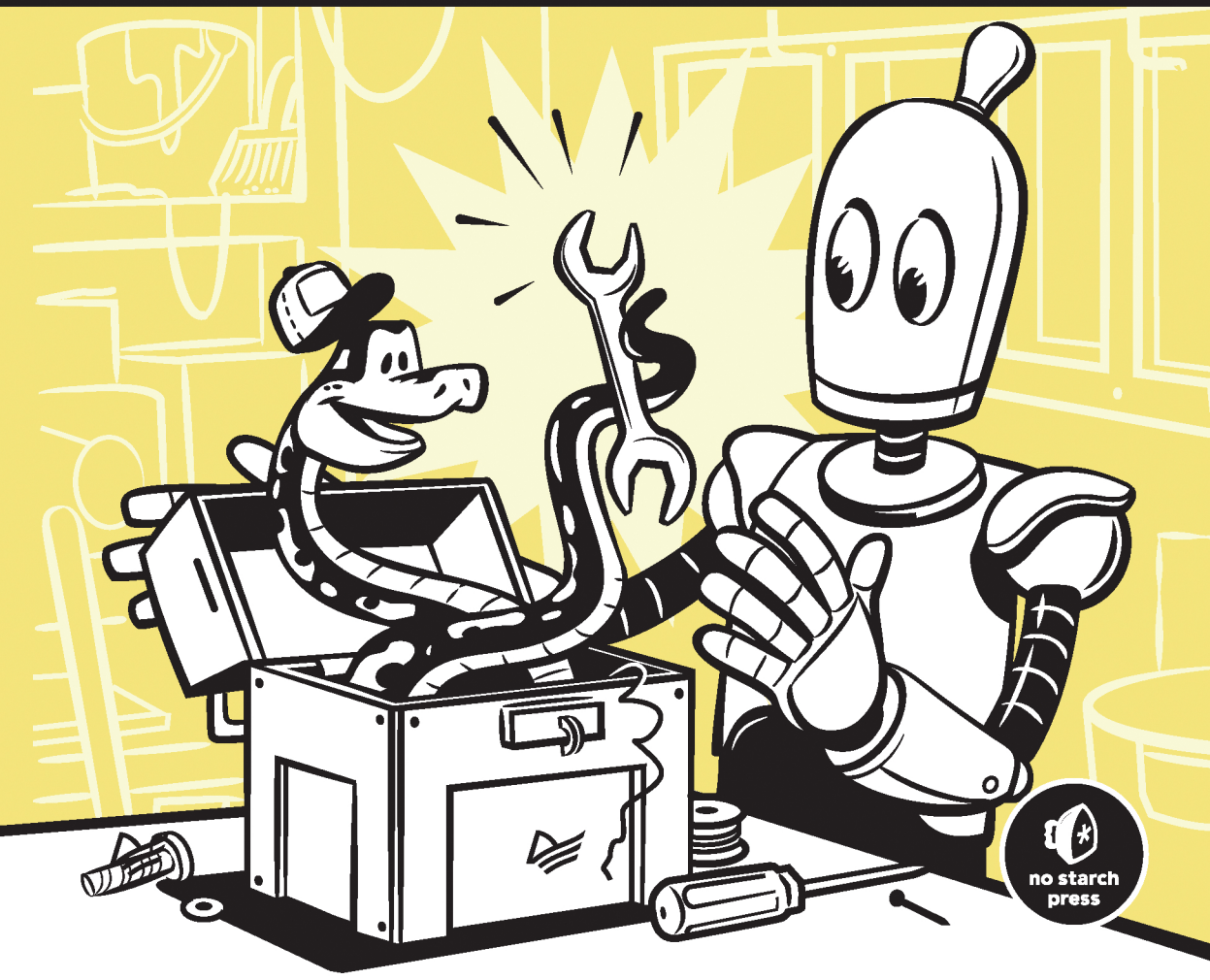
БОЛЬШАЯ КНИГА ПРОЕКТОВ PYTHON

ЭЛ СВЕЙГАРТ



THE BIG BOOK OF SMALL PYTHON PROJECTS

81 Easy Practice Programs

Al Sweigart

no starch
press

San Francisco

БОЛЬШАЯ КНИГА ПРОЕКТОВ PYTHON

ЭЛ СВЕЙГАРТ

ПИТЕР®

Санкт-Петербург • Москва • Минск

2022

ББК 32.973.2-018.1
УДК 004.43
С24

Свейгарт Эл

С24 Большая книга проектов Python. — СПб.: Питер, 2022. — 432 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1907-3

Вы уже освоили основы синтаксиса Python и готовы программировать? Отточите свои навыки на самых интересных задачах — графике, играх, анимации, расчетах и многом другом. Вы можете экспериментировать, добавляя к готовым проектам собственные детали.

В 256 строк кода поместится всё — «винтажная» экранная заставка, забег улиток на скорость, рекламный заголовок-приманка, вращающаяся спираль ДНК и так далее. Добавьте к этому пару строк своего кода, и вы сможете делиться собственными уникальными проектами в интернете.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-1718501249 англ.

ISBN 978-5-4461-1907-3

Оглавление

Об авторе 9

О научном редакторе 10

Введение 11

Проект 1. Бейглз 25

Проект 2. Парадокс дней рождения 30

Проект 3. Сообщение в виде битовой карты 35

Проект 4. Блек-джек 39

Проект 5. Отскакивающий от краев логотип DVD 47

Проект 6. Шифр Цезаря 53

Проект 7. Взлом шифра Цезаря 57

Проект 8. Генерация календарей 60

Проект 9. Морковка в коробке 65

Проект 10. Чо-хан 71

Проект 11. Генератор заголовков-приманок 75

Проект 12. Гипотеза Коллатца 80

Проект 13. Игра «Жизнь» Конвея 83

Проект 14. Обратный отсчет 87

Проект 15. Глубокая пещера 90

Проект 16. Ромбы 93

Проект 17. Арифметика с игральными костями ... 97
Проект 18. Выбрасыватель игральных костей ... 104
Проект 19. Цифровые часы ... 108
Проект 20. Цифровой поток ... 111
Проект 21. Визуализация ДНК ... 114
Проект 22. Утята ... 118
Проект 23. Гравировщик ... 124
Проект 24. Разложение на множители ... 130
Проект 25. Быстрый стрелок ... 134
Проект 26. Фибоначчи ... 137
Проект 27. Аквариум ... 141
Проект 28. Заливка ... 150
Проект 29. Моделирование лесного пожара ... 157
Проект 30. Четыре в ряд ... 162
Проект 31. Угадай число ... 168
Проект 32. Простак ... 171
Проект 33. Мини-игра со взломом ... 173
Проект 34. «Виселица» и «Гильотина» ... 179
Проект 35. Гексагональная сетка ... 185
Проект 36. Песочные часы ... 188
Проект 37. Голодные роботы ... 194
Проект 38. «Я обвиняю!» ... 202
Проект 39. Муравей Лэнгтона ... 211
Проект 40. П0г0в0рим (leetspeak) ... 217
Проект 41. Под счастливой звездой ... 220

Проект 42. Магический хрустальный шар ... 228

Проект 43. Манкала ... 231

Проект 44. Бегущий в лабиринте 2D ... 238

Проект 45. Бегущий в лабиринте 3D ... 244

Проект 46. Моделирование статистики за миллион бросков игральных костей ... 253

Проект 47. Генератор картин в стиле Мондриана ... 256

Проект 48. Парадокс Монти Холла ... 263

Проект 49. Таблица умножения ... 270

Проект 50. Девяносто девять бутылок ... 272

Проект 51. ДевяНосто деевяять буутылок ... 275

Проект 52. Счет в различных системах счисления ... 279

Проект 53. Периодическая таблица элементов ... 283

Проект 54. Поросячья латынь ... 287

Проект 55. Лотерея Powerball ... 290

Проект 56. Простые числа ... 295

Проект 57. Индикатор хода выполнения ... 298

Проект 58. Радуга ... 302

Проект 59. Камень, ножницы, бумага ... 306

Проект 60. Камень, ножницы, бумага (беспроигрышная версия) ... 310

Проект 61. Шифр ROT13 ... 314

Проект 62. Вращающийся куб ... 317

Проект 63. Царская игра Ура ... 324

Проект 64. Семисегментный модуль индикации ... 333

Проект 65. Ковер из «Сияния» ... 337

Проект 66. Простой шифр подстановки ... 340

Проект 67. Синусовидное сообщение ... 345
Проект 68. Игра в 15 ... 349
Проект 69. Бега улиток ... 355
Проект 70. Соробан — японский абак ... 359
Проект 71. Повторение музыки ... 365
Проект 72. Губкорегистр ... 369
Проект 73. Головоломка судоку ... 372
Проект 74. Преобразование текста в речь ... 379
Проект 75. Три карты Монте ... 381
Проект 76. Крестики-нолики ... 387
Проект 77. Ханойская башня ... 391
Проект 78. Вопросы с подвохом ... 396
Проект 79. Игра «2048» ... 403
Проект 80. Шифр Виженера ... 411
Проект 81. Головоломка с ведрами воды ... 416
Приложение А. Указатель тегов ... 422
Приложение Б. Таблица кодов символов ... 426

Об авторе

Эл Свейгарт (Al Sweigart) — разработчик программного обеспечения, автор и участник Python Software Foundation. Ранее работал руководителем по вопросам образования в Музее искусств и цифровых развлечений — музее компьютерных игр Окленда, Калифорния. Эл написал несколько книг по программированию, включая *Automate the Boring Stuff with Python*[1] и *Invent Your Own Computer Games with Python*[2]. Книги Эла свободно доступны под лицензией Creative Commons на его сайте https://inventwithpython.com/. Его кошка Зофи обожает снеки с водорослями нори.

[1] *Свейгарт Э.* Автоматизация рутинных задач с помощью Python. — М.: Вильямс, 2017.

[2] *Свейгарт Э.* Учим Python, делая крутые игры. — М.: Эксмо, 2021.

О научном редакторе

Сара Кучински (Sarah Kuchinsky) — магистр естественных наук, корпоративный инструктор и консультант. Она использует Python для множества целей, включая моделирование систем здравоохранения, разработку игр и автоматизацию задач. Сара — один из основателей конференции North Bay Python, председатель комиссии по обучающим пособиям конференции PyCon US и ведущий организатор группы PyLadies Silicon Valley. Защитила дипломы по теории управления, инженерии и математике.

Введение

Программировать легко, когда можно просто скопировать `print('Hello, world!')`. Вероятно, вам случалось читать хорошо структурированную книгу или проходить онлайн-курс для начинающих, прорабатывая упражнения и поддакивая жаргонным словечкам, которые вы (более или менее) понимали. Однако когда пришло время покидать гнездо и писать собственные программы, возможно, летать самостоятельно оказалось не так просто. Вы обнаружили, что пялитесь в пустое окно редактора и не знаете, как начать писать свои программы на Python.

Проблема в том, что следовать учебнику очень полезно для усвоения теории, но это далеко не всегда то же самое, что учиться писать новые программы с нуля. На данном этапе часто рекомендуют изучать программное обеспечение с открытым исходным кодом или работать над собственными проектами, но проекты с открытым исходным кодом далеко не всегда хорошо документированы или доступны для начинающих. И хотя работа над собственными проектами очень стимулирует, вы остаетесь совершенно без руководства.

В книге вы найдете практические примеры применения различных идей программирования в виде коллекции более чем 80 игр, имитационных моделей и объектов цифрового искусства. Они представляют собой не просто фрагменты кода, а полноценные работающие программы на Python. Вы можете скопировать их код, чтобы лучше познакомиться с тем, как они работают, поэкспериментировать, а затем попытаться воссоздать их самостоятельно в качестве практики. Вскоре вы найдете идеи для собственных программ и, главное, будете знать, как приступить к их реализации.

Проектирование маленьких программ

Программирование зарекомендовало себя как навык, открывающий большие возможности, в том числе по созданию технологических компаний стоимостью

миллиарды долларов и потрясающих технических достижений. Стремиться к большему при создании своего ПО легко, но когда переоцениваешь свои силы — в конце концов получаешь только незаконченные программы и разочарование. Однако вовсе не нужно быть компьютерным гением, чтобы создавать интересные и креативные программы.

Проекты на Python в этой книге отвечают нескольким основным принципам проектирования, чтобы упростить для начинающих понимание исходного кода.

- **Они маленькие** — большинство приведенных в книге программ не превышает 256 строк кода, а часто они намного короче. Благодаря таким ограничениям размера читателю проще понять эти программы. Число 256 выбрано случайно, но $256 = 2^8$, а степени двойки — счастливые для программистов числа.
- **Приводятся в виде текста** — текст проще, чем графика. Когда и исходный код, и выводимые программой результаты представляют собой текст, можно легко отследить, скажем, причинно-следственную связь между оператором `print('Thanks for playing!')` в коде и выводимой на экран надписью `Thanks for playing!`.
- **Не требуют установки** — все программы заключены в отдельные, самодостаточные файлы исходного кода Python с расширением `.py`, например `tictactoe.py`. Не нужно запускать программу установки и можно легко разместить такую программу в интернете, чтобы поделиться ею с другими.
- **Их много** — в книге приведена 81 программа. Вы обязательно найдете программы себе по вкусу среди настольных игр, карточных игр, художественных цифровых изображений, имитационных моделей, математических загадок, лабиринтов и развлекательных программ.
- **Они простые** — эти программы были написаны так, чтобы быть понятными даже для начинающих. Выбирая между кодом на основе сложных, высокопроизводительных алгоритмов и простым и ясным кодом, я всегда в этой книге склонялся к последнему.

Программы в текстовом формате могут показаться несколько старомодными, но подобный стиль программирования позволяет не отвлекаться на нюансы, связанные со скачиванием графических данных, установкой дополнительных библиотек и организацией каталогов проекта. Вместо этого можно сосредоточиться на самом коде.

Для кого эта книга

Книга написана для двух групп людей. В первую входят те, кто уже освоил основы Python и программирования вообще, но все еще не вполне представляет, как писать собственные программы. Этим людям может казаться, что программирование для них «не сложилось». Можно успешно решать практические упражнения из

учебников, но с трудом представлять себе, как выглядит полная программа. Благодаря сначала копированию, а затем и воссозданию игр из данной книги эти люди постепенно поймут, как изучаемые здесь понятия программирования компоновать во множество настоящих программ.

Во вторую группу входят новички в сфере программирования, достаточно азартные и настроенные на приключения, желающие погрузиться в работу с головой и сразу же начать создавать игры, имитационные модели и программы, обрабатывающие большие массивы числовых данных. Таких людей устраивает копирование кода и изучение его по ходу дела. Или, возможно, они уже умеют программировать на другом языке, но Python им внове. Эта книга, хотя и не заменяет полноценный вводный курс Python, кратко знакомит читателя с его основами и учит использовать отладчик для исследования внутренних механизмов работы программы во время выполнения.

Опытные программисты тоже могут развлечься с программами из этой книги, но учтите, что она написана все-таки для новичков.

Что можно найти в издании

Хотя основная часть книги посвящена конкретным программам, в ней вы найдете также дополнительные источники информации по общим вопросам программирования и Python.

- **Проекты** — перечислять здесь 81 проект будет слишком долго, но каждому из них посвящена отдельная глава, включающая название проекта, описание, пример результатов работы программы и ее исходный код. Вдобавок приводятся рекомендации относительно изменений, которые вы можете внести в код, чтобы адаптировать эти программы к своим требованиям.
- **Приложение А «Указатель тегов»** — перечислены все проекты, разбитые на категории по тегам проектов.
- **Приложение Б «Таблица кодов символов»** — список кодов символов для сердечек, линий, стрелок и блоков, которые могут выводить ваши программы.

Как научиться чему-то на программах из этой книги

Эта книга не учит языку Python или каким-либо понятиям программирования, подобно обычному учебнику. В ней применяется подход «учеба на практике», при котором читателя призывают вручную копировать программы, экспериментировать с ними и исследовать внутренние механизмы их работы путем запуска их под отладчиком.

Основная идея книги состоит не в подробном пояснении синтаксиса языка программирования, а в демонстрации полноценных примеров программ, реализующих нечто реальное: карточные ли игры, воспроизведение ли анимации, исследование ли математической загадки. Поэтому я рекомендую придерживаться следующих этапов.

1. Скачайте программу и запустите ее, чтобы посмотреть, что она делает.
2. Начав с пустого файла, скопируйте код игры из книги, вручную набрав его (не используйте команды копирования/вставки!).
3. Запустите программу снова, вернитесь и исправьте все опечатки и ошибки, которые вы могли случайно внести в код.
4. Запустите программу из-под отладчика, чтобы последовательно выполнить код построчно и разобраться, что делает каждая строка.
5. Найдите комментарии, отмеченные `(!)`, чтобы найти код, который можно изменить и посмотреть, как это повлияет на программу при следующем запуске.
6. Наконец, попробуйте воссоздать программу самостоятельно с нуля. Не обязательно точную ее копию; можете привнести в нее что-то свое.

При копировании кода из книги можете не набирать комментарии (текст в конце строк, следующий за символом `#`) — это примечания для программистов, игнорируемые Python. Однако старайтесь писать свой код на Python на строках с теми же номерами, что и программы в данном издании, чтобы упростить их сравнение. Если у вас не получается найти опечатки в своих программах, то можете сравнить свой код с кодом из книги с помощью онлайн-утилиты diff по адресу https://inventwithpython.com/bigbookpython/diff/.

Каждую программу описывает набор тегов, например `настольная игра`, `имитационная модель`, `художественная` и `для двух игроков`. Пояснения ко всем этим тегам и перекрестный указатель тегов и проектов приведены в приложении А.

Скачивание и установка Python

Python — название как языка программирования, так и интерпретатора, выполняющего код на языке Python. Утилита-интерпретатор совершенно бесплатна и свободно доступна для скачивания и использования. Можете проверить, не установлен ли уже в вашей системе Python, с помощью командной строки. В Windows откройте командную строку и введите **`py --version`**. Если выведено примерно следующее, значит, Python установлен:

```
C:\Users\Al>py --version
Python 3.9.1
```

В macOS и Linux откройте терминал и введите **python3 --version**. Если будет выведено примерно следующее, значит, Python установлен:

```
$ python3 --version
Python 3.9.1
```

В этой книге используется Python версии 3. При переходе от Python 2 к Python 3 было внесено несколько обратно несовместимых изменений, и для работы описанных здесь программ требуется как минимум Python версии 3.1.1 (выпущена в 2009 году). Если вы увидите сообщение об ошибке, гласящее, что Python не найден или версия Python — 2, можете скачать свежий установочный пакет Python для вашей операционной системы с сайта https://python.org/. В случае проблем с установкой Python дополнительные инструкции можно найти здесь: https://installpython3.com/.

Скачивание и установка редактора Mu

Код Python вы будете вводить в текстовом редакторе или интегрированной среде разработки (IDE) приложений. Я рекомендую использовать в качестве IDE редактор Mu, если вы новичок; он прост и не отвлекает ваше внимание множеством расширенных опций.

Откройте сайт https://codewith.mu/ в браузере. В Windows и macOS скачайте установочный пакет для соответствующей операционной системы и запустите его, дважды щелкнув на файле. В macOS запуск установочного пакета приведет к открытию окна, в котором необходимо перетащить пиктограмму Mu в каталог Applications для продолжения установки. В Ubuntu придется установить Mu в качестве пакета Python. В этом случае откройте новое окно терминала и выполните команду `pip3 install mu-editor` для установки и `mu-editor` — для запуска. Нажмите кнопку Instructions в разделе Python Package страницы загрузки для подробных инструкций.

Запуск редактора Mu

После установки запустите Mu:

- в Windows 7 или более поздней версии щелкните на пиктограмме Start в нижнем левом углу экрана, введите **mu** в поле поиска и выберите Mu, когда он появится;
- в macOS откройте окно Finder, щелкните на Applications, а затем на mu-editor;
- в Ubuntu нажмите Ctrl+Alt+T, чтобы открыть окно терминала, и введите команду **python3 -m mu**.

При первом запуске Mu появится окно Select Mode (Выберите режим) со следующими вариантами: Adafruit CircuitPython, BBC micro:bit, Pygame Zero и Python 3. Выберите Python 3. При желании позднее всегда можно изменить режим, нажав кнопку Mode вверху окна редактора.

Вы сможете вводить код в главном окне Mu, а затем сохранять его, открывать и запускать файлы с помощью кнопок вверху.

Запуск IDLE и других редакторов

Можете использовать какие угодно редакторы для написания кода на Python. Вместе с Python устанавливается ПО IDLE (Integrated Development and Learning Environment, интегрированная среда разработки и изучения), которое может служить альтернативным редактором, если по какой-либо причине вам не удалось установить Mu или заставить его работать. Запустим IDLE.

- В Windows 7 или более поздней версии щелкните на пиктограмме Start в нижнем левом углу экрана, введите **`idle`** в поле поиска и выберите IDLE (Python GUI).
- В macOS откройте окно Finder, нажмите Applications ▸ Python 3.9 ▸ IDLE.
- В Ubuntu выберите Applications ▸ Accessories ▸ Terminal и введите **`idle3`** (можете также щелкнуть на Applications вверху экрана, выбрать Programming, а затем щелкнуть на IDLE 3).
- На Raspberry Pi нажмите кнопку меню Raspberry Pi в левом верхнем углу; щелкните на Programming, а затем встав на Python 3 (IDLE). Можете также выбрать Thonny Python IDE из меню Programming.

Существует еще несколько бесплатных редакторов, с помощью которых можно вводить и выполнять код Python, например:

- Thonny, IDE Python для начинающих по адресу https://thonny.org/;
- PyCharm Community Edition — IDE Python, которую используют разработчики-профессионалы, по адресу https://www.jetbrains.com/pycharm/.

Установка модулей Python

Для большинства программ из этой книги требуется только стандартная библиотека Python, устанавливаемая автоматически вместе с Python. Однако для некоторых программ требуются сторонние модули, например `pyperclip`, `bext`, `playsound` и `pyttsx3`. Их все можно установить сразу, загрузив модуль `bigbookpython`.

Что касается редактора Mu, то необходимо установить версию 1.1.0-alpha (или более позднюю). По состоянию на 2020 год эту версию можно найти вверху страницы скачивания по адресу https://codewith.mu/en/download в разделе Try the Alpha of the Next Version of Mu (Попробуйте альфа-версию обновленного Mu). После установки нажмите значок с шестеренкой в левом нижнем углу окна, чтобы вызвать окно Mu Administration (Администрирование Mu). Выберите вкладку Third Party Packages (Сторонние пакеты), введите `bigbookpython` в текстовое поле и нажмите Ok. В результате этого будут установлены все сторонние модули, используемые программами из книги.

При использовании Visual Studio Code или редактора IDLE откройте редактор и выполните следующий код Python в интерактивной командной оболочке:

```
>>> import os, sys
>>> os.system(sys.executable + ' -m pip install --user bigbookpython')
0
```

Число `0`, выводимое после второй инструкции, означает, что все работает должным образом. В противном случае, если вы увидите сообщение об ошибке или другое число, попробуйте выполнить следующие команды без опции `--user`:

```
>>> import os, sys
>>> os.system(sys.executable + ' -m pip install bigbookpython')
0
```

При использовании любого редактора можете попробовать выполнить команду `import pyperclip` или `import bext`, чтобы убедиться, что установка прошла успешно. Если эти команды импорта не возвращают сообщения об ошибке, значит, соответствующие модули установлены правильно и вы сможете запускать использующие их проекты из книги.

Копирование кода из книги

Программирование — навык, совершенствуемый прежде всего практикой. Не стоит просто читать код в этой книге и копировать/вставлять его в свой компьютер. Потратьте немного времени, но наберите код в редакторе вручную. Обращайте внимание на номера строк, чтобы случайно не пропустить ничего. Если вы столкнетесь с ошибками, то воспользуйтесь онлайн-утилитой diff по адресу https://inventwithpython.com/bigbookpython/diff/, чтобы сравнить свой код с кодом из книги. Чтобы лучше разобраться в работе программ, попробуйте запустить их под отладчиком.

После того как исходный код будет введен и запущен несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы можете найти в комментариях, помеченных (!), вдобавок в каждом проекте есть список идей более крупных изменений.

Далее попробуйте воссоздать программу с самого начала, не глядя на исходный код в книге. Ваша программа не должна в точности воспроизводить программу из книги, вы можете придумать собственную версию.

Проработав все программы из книги, вероятно, вы захотите начать создавать собственные. Большинство современных компьютерных игр и прикладных программ достаточно сложны, их создание требует команды, состоящей из программистов, архитекторов и графических дизайнеров. Однако многие настольные, карточные игры и игры на бумаге достаточно просты, чтобы для них можно было разработать соответствующую программу. Многие из них относятся к категории абстрактных стратегических игр. Их список можно найти по адресу https://en.wikipedia.org/wiki/List_of_abstract_strategy_games.

Запуск программ из терминала

Использующие модуль `bext` программные проекты из книги выводят разноцветный текст. Однако эти цвета не отображаются, если запускать программы из Mu, IDLE или других редакторов, так что их необходимо запускать из окна *терминала* (командной строки). В Windows для этого запустите программу Command Prompt (Командная строка) из меню Пуск. В macOS запустите Terminal (Терминал) из Spotlight. В Ubuntu Linux запустите Terminal (Терминал) из Ubuntu Dash или нажмите Ctrl+Alt+T.

Когда откроется окно терминала, перейдите в каталог с вашими файлами `.py` с помощью команды `cd` (change directory — «сменить каталог»). Например, если я работаю под Windows и сохранил программы Python в каталог C:\Users\Al, то должен ввести следующую команду:

```
C:\>cd C:\Users\Al

C:\Users\Al>
```

Далее, чтобы запустить программы Python, введите команду `python` *`вашаПрограмма.py`* в Windows или `python3` *`вашаПрограмма.py`* в macOS или Linux, заменив *`вашаПрограмма.py`* на название соответствующей программы на Python:

```
C:\Users\Al>python guess.py
Угадайте число, (с) Эл Свейгарт al@inventwithpython.com
```

```
Я загадал число от 1 до 100.
У вас осталось 10 попыток. Угадывайте.
--сокращено--
```

Прервать выполнение программы можно из терминала нажатием Ctrl+C, вместо того чтобы закрывать само окно терминала.

Запуск программ со смартфона или планшета

Работать с ноутбука или стационарного компьютера с полноценной клавиатурой будет удобнее, поскольку набирать код на клавиатуре телефона или даже планшета очень утомительно. И хотя не существует официальных интерпретаторов Python для Android или iOS, есть сайты с интерактивными онлайн-оболочками, которые можно использовать из браузера. Они подходят и для ноутбуков/стационарных компьютеров на случай, если вы преподаватель, у которого нет прав на установку нового программного обеспечения на компьютерах в учебном классе.

Интерпретаторы Python на сайтах https://repl.it/languages/Python3/ и https://www.pythonanywhere.com/ можно свободно использовать в браузере. Эти сайты подходят для большинства проектов из данной книги, но не для программ, использующих сторонние модули, например `bext`, `pyperclip`, `pyttsx3` и `playsound`, и не для программ, читающих или записывающих файлы с помощью функции `open()`. Если вы увидите эти элементы в коде программы, значит, она не будет работать в указанных онлайн-интерпретаторах Python. Однако большинство программ из книги будет в них прекрасно работать.

Где получить помощь

Если вы не можете нанять частного преподавателя и у вас нет друга-программиста, который мог бы ответить на вопросы по программированию, то вам придется самостоятельно искать ответы на них. К счастью, эти вопросы практически наверняка кто-то уже задавал. Умение найти ответы самостоятельно — важный навык для любого программиста.

Не расстраивайтесь, если поймете, что постоянно ищете в интернете ответы на вопросы по программированию. Пока вы учитесь, нет ничего постыдного в том, чтобы искать что-то в интернете, вместо того чтобы запоминать все нюансы программирования с самого начала. Даже профессиональные разработчики ПО каждый день ищут что-то в Сети. В этом разделе вы узнаете, как задавать «умные» вопросы и искать ответы на них в интернете.

Когда программа пытается выполнить недопустимую инструкцию, отображается сообщение об ошибке: трассировка, описывающая тип произошедшей ошибки и то, на какой строке кода она произошла. Вот пример программы, в которой произошла ошибка во время вычисления того, сколько кусков пиццы должен получить каждый:

```
Traceback (most recent call last):
  File "pizza.py", line 5, in <module>
    print('Each person gets', (slices / people), ' slices of pizza.')
ZeroDivisionError: division by zero
```

Из этой трассировки не сразу ясно, что проблема вызвана переменной `people`, значение которой равно `0`, вследствие чего выражение `slices / people` приводит к ошибке деления на ноль. Сообщения об ошибках зачастую настолько коротки, что даже не являются предложениями. Поскольку программисты сталкиваются с этими сообщениями постоянно, они играют роль скорее оповещений, а не полноценных пояснений. Если вы встретили какое-то сообщение об ошибке впервые, то скопируйте его и поищите в интернете, почти наверняка вы найдете подробное пояснение, что означает эта ошибка и чем она могла быть вызвана.

Если в интернете найти решение проблемы не удалось, то можете выставить свой вопрос на онлайн-форуме или задать кому-либо по электронной почте. Для большей эффективности процесса задавайте конкретные, четко сформулированные вопросы. То есть приведите полный исходный код и полное сообщение об ошибке со всеми подробностями, расскажите, какие решения проблемы уже пробовали, а также какую операционную систему и версию Python используете. Изложенные на форуме ответы не только позволят решить вашу проблему, но и помогут в будущем другим программистам с тем же вопросом, которые найдут ваше сообщение.

Набор кода

Программисту не обязательно уметь быстро печатать на клавиатуре, но лишним это умение не будет. Многие люди печатают двумя пальцами, в то время как более быстрый набор может значительно облегчить написание программ. По мере работы над программами из книги вам будет удобнее смотреть на код, а не на клавиатуру. Существуют бесплатные сайты для обучения быстрому набору текста, например https://typingclub.com/ и https://www.typing.com/. Хорошая программа для обучения набору текста отображает клавиатуру и прозрачные руки на экране, чтобы вы могли практиковаться, избавляясь от вредной привычки смотреть на клавиатуру в поисках клавиш. Как и любой другой навык, набор — дело привычки, а написание кода даст вам множество возможностей привыкнуть к набору.

Выполнять различные действия намного быстрее, не перемещая указатель мыши к пункту меню, позволяют также сочетания горячих клавиш. Они часто записаны

в виде наподобие Ctrl+C, означающем, что нужно нажать одну из двух клавиш Ctrl и затем, не отпуская ее, нажать клавишу C. Но не нажать клавишу Ctrl, отпустить ее, а затем нажать клавишу C.

Выучить распространенные сочетания горячих клавиш, например Ctrl+C для копирования или Ctrl+S для сохранения, можно, открыв мышью меню вверху приложения (в Windows и Linux) или вверху экрана (в macOS). Время, потраченное на изучение этих сочетаний клавиш, окупится с лихвой.

Другие сочетания горячих клавиш не так очевидны. Например, Alt+Tab в Windows и Linux и Command+Tab в macOS позволяет переключиться на окно другого приложения. Чтобы выбрать конкретное окно, можно зажать Alt или Command и последовательно нажимать Tab.

Копирование и вставка

Буфер обмена (clipboard) — элемент операционной системы, предназначенный для временного хранения данных для вставки, которые могут представлять собой текст, изображения, файлы и другие типы информации, хотя в этом разделе мы будем говорить только о текстовых данных. При *копировании* (copying) текста копия выбранного в текущий момент времени текста попадает в буфер обмена. При *вставке* (pasting) текст из буфера вставляется в то место, где сейчас находится курсор, как если бы вы мгновенно набрали его сами. Копирование и вставка текста освобождают вас от необходимости заново набирать уже имеющийся на вашем компьютере текст, неважно, одну строку или сотни страниц.

Чтобы копировать и вставить текст, сначала выберите (*выделите*, highlight) текст, который будете копировать. Для этого можно нажать основную кнопку мыши (левую, если мышь предназначена для правшей) и перетащить указатель мыши по всему выбираемому тексту. Однако зачастую быстрее и точнее будет нажать кнопку Shift и переместить курсор с помощью сочетания горячих клавиш. Многие приложения позволяют мгновенно выделить целое слово путем двойного щелчка на нем. Можно также выделить целую строку или абзац тройным щелчком.

Следующий шаг: нажать Ctrl+C в Windows или Command+C в macOS, чтобы копировать выделенный текст в буфер обмена. В нем может содержаться только один элемент, так что копируемый текст замещает находившееся в буфере до этого.

Наконец, передвиньте курсор туда, куда нужно вставить текст, и нажмите Ctrl+V в Windows или Command+V в macOS. Вставлять текст можно столько раз, сколько нужно; он остается в буфере обмена до тех пор, пока вы не скопируете новый текст.

Поиск и замена текста

Дэн Рассел (Dan Russell), поисковый антрополог в Google, в статье 2011 года в *Atlantic* пояснил: при изучении привычек использования компьютеров людьми оказалось, что 90 % из них не знали, что можно нажать Ctrl+F (в Windows и Linux) или Command+F (в macOS) для поиска слов в приложениях. Это исключительно удобная возможность не только в редакторах кода, но и в текстовых редакторах, браузерах, приложениях электронных таблиц и практически во всех прочих программах, отображающих текст. Вы можете нажать Ctrl+F — и появится окно Find (Найти), куда можно ввести слово для поиска в программе. Нажатие клавиши F3 обычно позволяет повторить поиск и найти следующее вхождение слова. Эта возможность экономит колоссальное количество времени по сравнению с поиском слова с помощью просмотра документа вручную.

В редакторах также есть возможность поиска и замены текста, с которой обычно связано сочетание клавиш Ctrl+H (Command+H). Она позволяет находить вхождения какого-либо фрагмента текста и заменять его другим. Это очень удобно, например, для переименования переменной или функции. Однако возможность поиска и замены текста следует использовать с осторожностью, чтобы не заменить текст, случайно совпавший с критерием поиска.

Отладчик

Отладчик — утилита, выполняющая программы построчно, с возможностью просмотра текущего состояния переменных программы. Это ценный инструмент для поиска программных ошибок. В данном разделе я расскажу о возможностях отладчика редактора Mu. Не волнуйтесь: возможности всех отладчиков одинаковы, даже если интерфейсы пользователя различаются.

Чтобы запустить программу в отладчике, воспользуйтесь пунктом меню Debug (Отладка) в своей IDE вместо пункта меню Run (Запуск). Отладчик запустится в приостановленном состоянии на первой строке программы. У всех отладчиков есть кнопки Continue (Продолжить), Step In (Шаг с заходом), Step Over (Шаг с обходом), Step Out (Шаг с выходом) и Stop (Останов).

При нажатии кнопки Continue (Продолжить) программа выполняется как обычно, до тех пор пока не завершится или не достигнет точки останова (о них я расскажу позже в этом разделе). Если вы закончили отладку и хотите, чтобы программа далее выполнялась обычным образом, то нажмите Continue (Продолжить).

При нажатии кнопки Step In (Шаг с заходом) отладчик выполняет следующую строку кода, после чего останавливается. Если следующая строка кода представляет

собой вызов функции, то отладчик «заходит» в эту функцию и переходит к первой строке ее кода.

При нажатии кнопки Step Over (Шаг с обходом) отладчик выполняет следующую строку кода аналогично кнопке Step In (Шаг с заходом). Но если следующая строка кода представляет собой вызов функции, то отладчик «обходит» код этой функции. Функция выполняется с обычной скоростью, и отладчик приостанавливается после возврата из ее вызова. Кнопку Step Over (Шаг с обходом) используют чаще, чем кнопку Step In (Шаг с заходом).

При нажатии кнопки Step Out (Шаг с выходом) отладчик выполняет строки кода с обычной скоростью, пока не происходит возврат из текущей функции. Если вы «зашли» в вызов функции с помощью кнопки Step In (Шаг с заходом) и просто хотите выполнить оставшиеся инструкции, пока не выйдете обратно, то нажмите кнопку Step Out (Шаг с выходом), чтобы отладчик «вышел» из текущего вызова функции.

Если вы хотите полностью завершить сеанс отладки и не продолжать выполнение остатка программы, то нажмите кнопку Stop (Останов). Она немедленно завершает выполнение программы.

Можно установить *точку останова* (breakpoint) на конкретной строке, при этом программа будет выполняться с обычной скоростью, до тех пор пока не достигнет данной строки. Затем отладчик остановит выполнение программы, чтобы вы могли изучить значения переменных и продолжить пошаговое выполнение отдельных строк кода. В большинстве IDE установить точку останова можно с помощью двойного щелчка на номере строки в левой части окна.

В любом отладчике где-нибудь в окне отладки отображаются значения, хранящиеся в текущий момент в переменных программы. Впрочем, один из распространенных методов отладки программ — *отладка с выводом значений* (print debugging). Метод заключается в добавлении вызовов `print()` для отображения значений переменных и в повторном запуске программы. Этот подход к отладке прост и удобен, однако часто требует больше времени, чем использование отладчика. При отладке с выводом значений необходимо добавлять вызовы `print()`, перезапускать программу, а затем удалять эти вызовы. Однако после перезапуска программы часто оказывается, что нужно добавить дополнительные вызовы `print()`, чтобы узнать значения других переменных. А значит, приходится перезапускать программу еще раз, что может означать еще один цикл добавления вызовов `print()` и т. д. Кроме того, легко можно упустить какие-либо из этих вызовов, что потребует дополнительного цикла их удаления. Отладка с выводом значений удобна для простых ошибок, но в конечном счете использование настоящего отладчика экономит время.

Резюме

Программирование — интересный и творческий навык. Хотите ли вы овладеть основами синтаксиса Python или просто посмотреть на настоящие программы на Python — проекты в этой книге породят новые идеи о том, чего можно добиться с помощью всего нескольких страниц кода.

Лучший способ работы с этими программами — не просто читать их код и копировать его. Потратьте немного времени, но наберите код в редакторе вручную, чтобы выработать «мышечную память». Помимо всего прочего, это немного замедлит вас, так что вы поневоле будете внимательнее обдумывать каждую из строк кода в ходе их набора, а не просто пробегать по ним глазами. Старайтесь искать в интернете все незнакомые инструкции или пробовать выполнять их в интерактивной командной оболочке.

Наконец, не поленитесь повторить программу с нуля, а затем модифицировать ее, добавив что-то свое. Благодаря этим упражнениям вы освоите концепции, на основе которых создаются настоящие работающие программы, а главное, хорошо проведете время!

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На веб-сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1

Бейглз

В дедуктивной логической игре «Бейглз» необходимо по подсказкам угадать секретное число из трех цифр. В ответ на ваши попытки угадать игра выдает одну из трех подсказок: Pico, если вы угадали правильную цифру на неправильном месте, Fermi, если в вашей догадке есть правильная цифра на правильном месте, и Bagels, если в догадке не содержится правильных цифр. На угадывание секретного числа у вас десять попыток.

Программа в действии

Результат выполнения `bagels.py` выглядит следующим образом:

```
Bagels, a deductive logic game.
By Al Sweigart al@inventwithpython.com

I am thinking of a 3-digit number. Try to guess what it is.
Here are some clues:
When I say:    That means:
  Pico         One digit is correct but in the wrong position.
  Fermi        One digit is correct and in the right position.
  Bagels       No digit is correct.
I have thought up a number.
```

```
 You have 10 guesses to get it.
Guess #1:
> 123
Pico
Guess #2:
> 456
Bagels
Guess #3:
> 178
Pico Pico
--сокращено--
Guess #7:
> 791
Fermi Fermi
Guess #8:
> 701
You got it!
Do you want to play again? (yes or no)
> no
Thanks for playing!
```

Описание работы

Обратите внимание: в данной программе используются не целочисленные значения, а строковые, содержащие цифры. Например, `'426'` — вовсе не то же значение, что и `426`. Это необходимо, поскольку мы производим строковое сравнение с секретным числом, а не математическую операцию. Учтите, что `'0'` может быть значащей цифрой: строковое значение `'026'` отличается от `'26'`, в то время как целочисленное значение `026` то же самое, что `26`.

```
 1. """Бейглз, (c) Эл Свейгарт al@inventwithpython.com
 2. Дедуктивная логическая игра на угадывание числа по подсказкам.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Один из вариантов этой игры приведен в книге Invent Your Own
 5. Computer Games with Python на https://nostarch.com/inventwithpython
 6. Теги: короткая, игра, головоломка"""
 7.
 8. import random
 9.
10. NUM_DIGITS = 3  # (!) Попробуйте задать эту константу равной 1 или 10.
11. MAX_GUESSES = 10  # (!) Попробуйте задать эту константу равной 1 или 100.
12.
13.
14. def main():
15.     print('''Bagels, a deductive logic game.
16. By Al Sweigart al@inventwithpython.com
17.
```

```
18. I am thinking of a {}-digit number with no repeated digits.
19. Try to guess what it is. Here are some clues:
20. When I say:    That means:
21.   Pico         One digit is correct but in the wrong position.
22.   Fermi        One digit is correct and in the right position.
23.   Bagels       No digit is correct.
24.
25. For example, if the secret number was 248 and your guess was 843, the
26. clues would be Fermi Pico.'''.format(NUM_DIGITS))
27.
28.     while True:  # Основной цикл игры.
29.         # Переменная, в которой хранится секретное число, которое
30.         secretNum = getSecretNum() # должен угадать игрок
31.         print('I have thought up a number.')
32.         print(' You have {} guesses to get it.'.format(MAX_GUESSES))
33.
34.         numGuesses = 1
35.         while numGuesses <= MAX_GUESSES:
36.             guess = ''
37.             # Продолжаем итерации до получения правильной догадки:
38.             while len(guess) != NUM_DIGITS or not guess.isdecimal():
39.                 print('Guess #{}: '.format(numGuesses))
40.                 guess = input('> ')
41.
42.             clues = getClues(guess, secretNum)
43.             print(clues)
44.             numGuesses += 1
45.
46.             if guess == secretNum:
47.                 break  # Правильно, выходим из цикла.
48.             if numGuesses > MAX_GUESSES:
49.                 print('You ran out of guesses.')
50.                 print('The answer was {}.'.format(secretNum))
51.
52.         # Спрашиваем игрока, хочет ли он сыграть еще раз.
53.         print('Do you want to play again? (yes or no)')
54.         if not input('> ').lower().startswith('y'):
55.             break
56.     print('Thanks for playing!')
57.
58.
59. def getSecretNum():
60.     """Возвращает строку из NUM_DIGITS уникальных случайных цифр."""
61.     numbers = list('0123456789')  # Создает список цифр от 0 до 9.
62.     random.shuffle(numbers)  # Перетасовываем их случайным образом.
63.
64.     # Берем первые NUM_DIGITS цифр списка для нашего секретного числа:
65.     secretNum = ''
66.     for i in range(NUM_DIGITS):
67.         secretNum += str(numbers[i])
```

```
68.     return secretNum
69.
70.
71. def getClues(guess, secretNum):
72.     """Возвращает строку с подсказками pico, fermi и bagels
73.     для полученной на входе пары из догадки и секретного числа."""
74.     if guess == secretNum:
75.         return 'You got it!'
76.
77.     clues = []
78.
79.     for i in range(len(guess)):
80.         if guess[i] == secretNum[i]:
81.             # Правильная цифра на правильном месте.
82.             clues.append('Fermi')
83.         elif guess[i] in secretNum:
84.             # Правильная цифра на неправильном месте.
85.             clues.append('Pico')
86.     if len(clues) == 0:
87.         return 'Bagels'  # Правильных цифр нет вообще.
88.     else:
89.         # Сортируем подсказки в алфавитном порядке, чтобы их исходный
90.         # порядок ничего не выдавал.
91.         clues.sort()
92.         # Склеиваем список подсказок в одно строковое значение.
93.         return ' '.join(clues)
94.
95.
96. # Если программа не импортируется, а запускается, производим запуск:
97. if __name__ == '__main__':
98.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (`!`). Можете также сами попробовать придумать, как сделать следующее:

- изменить количество цифр в секретном числе, изменив константу `NUM_DIGITS`;
- изменить количество попыток угадывания, доступных игроку, изменив константу `MAX_GUESSES`;
- создать версию, в которой в секретном числе могут содержаться не только цифры, но и буквы.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и снова запустите программу, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если изменить константу `NUM_DIGITS`?
2. Что будет, если изменить константу `MAX_GUESSES`?
3. Что будет, если задать значение `NUM_DIGITS` больше `10`?
4. Что будет, если `secretNum = getSecretNum()` в строке 30 заменить на `secretNum = '123'`?
5. Какое сообщение об ошибке вы получите, если удалите или закомментируете `numGuesses = 1` в строке 34?
6. Что будет, если вы удалите или закомментируете `random.shuffle(numbers)` в строке 62?
7. Что будет, если вы удалите или закомментируете `if guess == secretNum:` в строке 74 и `return 'You got it!'` в строке 75?
8. Что будет, если вы закомментируете `numGuesses += 1` в строке 44?

2

Парадокс дней рождения

Парадокс дней рождения, также известный как задача о днях рождения, заключается в удивительно высокой вероятности того, что у двух человек совпадает день рождения даже в относительно небольшой группе людей. В группе из 70 человек вероятность совпадения дней рождения у двух людей составляет 99,9 %. Но даже в группе всего лишь из 23 человек вероятность совпадения дней рождения составляет 50 %. Приведенная программа производит несколько вероятностных экспериментов, чтобы определить процентные соотношения для групп различного размера. Подобные эксперименты с определением возможных исходов с помощью множества случайных испытаний называются экспериментами Монте-Карло.

Узнать больше о парадоксе дней рождения можно в соответствующей статье «Википедии»: https://ru.wikipedia.org/wiki/Парадокс_дней_рождения.

Программа в действии

Результат выполнения `birthdayparadox.py` выглядит следующим образом:

```
Birthday Paradox, by Al Sweigart al@inventwithpython.com
--сокращено--
How many birthdays shall I generate? (Max 100)
> 23
```

```
Here are 23 birthdays:
Oct 9, Sep 1, May 28, Jul 29, Feb 17, Jan 8, Aug 18, Feb 19, Dec 1, Jan 22,
May 16, Sep 25, Oct 6, May 6, May 26, Oct 11, Dec 19, Jun 28, Jul 29, Dec 6,
Nov 26, Aug 18, Mar 18

In this simulation, multiple people have a birthday on Jul 29

Generating 23 random birthdays 100,000 times...
Press Enter to begin...
Let's run another 100,000 simulations.
0 simulations run...
10000 simulations run...
--сокращено--
90000 simulations run...
100000 simulations run.
Out of 100,000 simulations of 23 people, there was a
matching birthday in that group 50955 times. This means
that 23 people have a 50.95 % chance of
having a matching birthday in their group.
That's probably more than you would think!
```

Описание работы

Выполнение 100 000 операций имитационного моделирования займет немало времени, поэтому в строках 94 и 95 выводятся сообщения о каждых 10 000 произведенных операций. Такая обратная связь демонстрирует пользователю, что программа не зависла. Обратите внимание на знаки подчеркивания в некоторых числах, например, `10_000` в строке 95 и `100_000` в строках 93 и 103. Никакого особого смысла у этих знаков подчеркивания нет, но Python позволяет их указывать, чтобы программисты могли упростить чтение чисел в их программах. Другими словами, число «сто тысяч» в форме `100_000` понятнее, чем `100000`.

```
 1. """Имитационное моделирование парадокса дней рождения, (c) Эл Свейгарт
 2. al@inventwithpython.com Изучаем неожиданные вероятности из "Парадокса
 3. дней рождения". Больше информации — в статье
 4. https://ru.wikipedia.org/wiki/Парадокс_дней_рождения
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, математическая, имитационное моделирование"""
 7. import datetime, random
 8.
 9.
10. def getBirthdays(numberOfBirthdays):
11.     """ Возвращаем список объектов дат для случайных дней рождения."""
12.     birthdays = []
13.     for i in range(numberOfBirthdays):
14.         # Год в нашем имитационном моделировании роли не играет, лишь
15.         # бы в объектах дней рождения он был одинаков.
16.         startOfYear = datetime.date(2001, 1, 1)
17.
```

```
        # Получаем случайный день года:
        randomNumberOfDays = datetime.timedelta(random.randint(0, 364))
        birthday = startOfYear + randomNumberOfDays
        birthdays.append(birthday)
    return birthdays


def getMatch(birthdays):
    """ Возвращаем объект даты дня рождения, встречающегося
    несколько раз в списке дней рождения."""
    if len(birthdays) == len(set(birthdays)):
        return None  # Все дни рождения различны, возвращаем None.

    # Сравниваем все дни рождения друг с другом попарно:
    for a, birthdayA in enumerate(birthdays):
        for b, birthdayB in enumerate(birthdays[a + 1 :]):
            if birthdayA == birthdayB:
                return birthdayA  # Возвращаем найденные соответствия.


# Отображаем вводную информацию:
print('''Birthday Paradox, by Al Sweigart al@inventwithpython.com

The Birthday Paradox shows us that in a group of N people, the odds
that two of them have matching birthdays is surprisingly large.
This program does a Monte Carlo simulation (that is, repeated random
simulations) to explore this concept.

(It's not actually a paradox, it's just a surprising result.)
''')

# Создаем кортеж названий месяцев по порядку:
MONTHS = ('Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun',
          'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec')

while True:  # Запрашиваем, пока пользователь не введет допустимое значение.
    print('How many birthdays shall I generate? (Max 100)')
    response = input('> ')
    if response.isdecimal() and (0 < int(response) <= 100):
        numBDays = int(response)
        break  # Пользователь ввел допустимое значение.
print()

# Генерируем и отображаем дни рождения:
print('Here are', numBDays, 'birthdays:')
birthdays = getBirthdays(numBDays)
for i, birthday in enumerate(birthdays):
    if i != 0:
        # Выводим запятую для каждого дня рождения после первого.
        print(', ', end='')
    monthName = MONTHS[birthday.month - 1]
```

```
    dateText = '{} {}'.format(monthName, birthday.day)
    print(dateText, end='')
print()
print()

# Выясняем, встречаются ли два совпадающих дня рождения.
match = getMatch(birthdays)

# Отображаем результаты:
print('In this simulation, ', end='')
if match != None:
    monthName = MONTHS[match.month - 1]
    dateText = '{} {}'.format(monthName, match.day)
    print('multiple people have a birthday on', dateText)
else:
    print('there are no matching birthdays.')
print()

# Производим 100 000 операций имитационного моделирования:
print('Generating', numBDays, 'random birthdays 100,000 times...')
input('Press Enter to begin...')

print('Let\'s run another 100,000 simulations.')
simMatch = 0  # Число операций моделирования с совпадающими днями рождения.
for i in range(100_000):
    # Отображаем сообщение о ходе выполнения каждые 10 000 операций:
    if i % 10_000 == 0:
        print(i, 'simulations run...')
    birthdays = getBirthdays(numBDays)
    if getMatch(birthdays) != None:
        simMatch = simMatch + 1
print('100,000 simulations run.')

# Отображаем результаты имитационного моделирования:
probability = round(simMatch / 100_000 * 100, 2)
print('Out of 100,000 simulations of', numBDays, 'people, there was a')
print('matching birthday in that group', simMatch, 'times. This means')
print('that', numBDays, 'people have a', probability, '% chance of')
print('having a matching birthday in their group.')
print('That\'s probably more than you would think!')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Как в программе представлены дни рождения? (Подсказка: взгляните на строку 16.)

2. Как убрать ограничение в 100 генерируемых дней рождения, установленное в программе?
3. Какое сообщение об ошибке вы получите, если удалите или закомментируете `numBDays = int(response)` в строке 57?
4. Как сделать так, чтобы программа отображала полные названия месяцев, например, `'January'` вместо `'Jan'`?
5. Как сделать так, чтобы надпись `'X simulations run...'` выводилась каждые 1000 операций имитационного моделирования, а не 10 000?

3

Сообщение в виде битовой карты

В этой программе *битовая карта* (bitmap) — двумерное изображение, каждый пиксел которого может быть одного из двух цветов, представлена в виде многострочного строкового значения и служит для определения способа отображения пользовательского сообщения. В нашей битовой карте пробелы соответствуют пустому пространству, а все прочие символы заменяются символами из пользовательского сообщения. Представленная битовая карта напоминает карту мира, но вы можете заменить ее на любую другую, какую пожелаете. Простота системы двоичного представления «пустое пространство или символы сообщения» делает ее идеально подходящей для начинающих. Попробуйте поэкспериментировать с различными сообщениями и посмотрите, что получится.

Программа в действии

Результат выполнения `bitmapmessage.py` выглядит следующим образом:

```
Bitmap Message, by Al Sweigart al@inventwithpython.com
Enter the message to display with the bitmap.
> Hello!

Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!He
   lo!Hello!Hello   l  !He lo  e      llo!Hello!Hello!Hello!Hello!He
  llo!Hello!Hello!Hello He lo H  l !Hello!Hello!Hello!Hello!Hello H
 el      lo!Hello!Hello!He       lo!Hello!Hello!Hello!Hello!Hel
          o!Hello!Hello          lo  e lo!H ll !Hello!Hello!H l
           !Hello!He            llo!Hel   Hello!Hello!Hell ! e
            Hello!He           ello!Hello!Hello!Hello!Hell  H
   l        H llo! ell         ello!Hello!Hell !Hello  el o
               lo!H  l         ello!Hello!Hell   ell !He  o
                 !Hello         llo!Hello!Hel    el   He  o
                 !Hello!H        lo!Hello!Hell    l  !H llo
                   ello!Hel         Hello!He          H llo Hell
                   ello!Hell         ello!H  l        Hell !H l o!
                   ello!Hell         ello!H l o           o!H l   H
                     lo!Hel          ello! el             o!Hel   H
                     lo!He            llo! e            llo!Hell
                    llo!H             llo!              llo!Hello
                    llo!              ll                 lo!Hell   e
                    llo                                      l    e
                    ll     l                   H
Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!Hello!He
```

Описание работы

Вместо того чтобы по отдельности вводить все символы шаблона карты мира, вы можете скопировать его целиком из https://inventwithpython.com/bitmapworld.txt и вставить. Строки из 68 точек вверху и внизу шаблона служат «линейками» для упрощения правильного выравнивания. Однако программа будет работать и при наличии каких-либо опечаток в этом шаблоне.

Вызываемый в строке 43 метод `bitmap.splitlines()` возвращает список отдельных строк из многострочного строкового значения `bitmap`. Благодаря использованию многострочного строкового значения можно легко заменить нашу битовую карту любым другим шаблоном. Программа заполнит все непробельные символы в шаблоне, так что нет разницы, какие это символы: звездочки, точки или любые другие символы.

Код `message[i % len(message)]` повторяет текст в `message`. Когда `i`, увеличиваясь с `0`, доходит до числа `len(message)`, выражение `i % len(message)` снова становится

равно `0`. В результате этого выражение `message[i % len(message)]` повторяет символы в `message` при росте `i`.

```
"""Сообщение в виде битовой карты, (c) Эл Свейгарт al@inventwithpython.com
Отображает текстовое сообщение в соответствии с указанной битовой картой.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: крошечная, для начинающих, графика"""

import sys

# (!) Попробуйте заменить это многострочное строковое значение на любое
# другое изображение
# Вверху и внизу строки расположено 68 точек:
# (Можете также скопировать и вставить это строковое значение из
# из https://inventwithpython.com/bitmapworld.txt)
bitmap = """
...................................................................
   **************   *  *** **  *      ******************************
  ********************* ** ** *  * ****************************** *
 **      *****************       ******************************
          *************          **  * **** ** ************** *
           *********            *******   ***************** * *
            ********           ************************** *
   *        * **** ***         *************** ******  ** *
              ****  *         ***************   *** ***  *
                 ******         *************    **   **  *
                 ********        *************    *  ** ***
                   ********          ********          * *** ****
                   *********         ******  *        **** ** * **
                   *********         ****** * *           *** *   *
                     ******          ***** **            *****   *
                     *****            **** *            ********
                    *****             ****              *********
                    ****              **                 *******   *
                    ***                                      *    *
                    **     *                   *
..................................................................."""

print('Bitmap Message, by Al Sweigart al@inventwithpython.com')
print('Enter the message to display with the bitmap.')
message = input('> ')
if message == '':
    sys.exit()

# Проходим в цикле по всем строкам битовой карты:
for line in bitmap.splitlines():
    # Проходим в цикле по всем символам строки:
    for i, bit in enumerate(line):
        if bit == ' ':
            # Выводим пустое пространство согласно пробелу в битовой карте:
            print(' ', end='')
```

```
49.         else:
50.             # Выводим символ сообщения:
51.             print(message[i % len(message)], end='')
52.     print()  # Выводим символ новой строки.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете, например, изменить строковое значение в `bitmap` и создать совершенно другие шаблоны.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если игрок введет в качестве сообщения пустую строку?
2. Играет ли какую-либо роль то, какие именно не пробельные символы указаны в переменной `bitmap`?
3. Каков смысл переменной `i`, созданной в строке 45?
4. Какая программная ошибка возникнет, если удалить или закомментировать `print()` в строке 52?

4

Блек-джек

Блек-джек, известный также как «двадцать одно», — карточная игра, в которой игроки пытаются набрать количество очков, как можно более близкое к 21, но не больше. В данной программе используются изображения, составленные из текстовых символов, так называемая *ASCII-графика*. ASCII (American Standard Code for Information Interchange, американский стандартный код обмена информацией) представляет собой таблицу соответствий текстовых символов числовым кодам, она применялась до того, как ее заменила кодировка Unicode[1]. Игральные карты в этой программе представляют собой пример ASCII-графики:

```
 ___  ___
|A  | |10 |
| ♣ | | ♦ |
|__A| |_10|
```

Прочие правила и историю этой карточной игры вы можете найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Блэкджек.

[1] Вообще-то нельзя сказать, что Unicode заменила ASCII, ASCII используется и по сей день. — *Примеч. пер.*

Программа в действии

Результат выполнения `blackjack.py` выглядит следующим образом:

```
Blackjack, by Al Sweigart al@inventwithpython.com
    Rules:
      Try to get as close to 21 without going over.
      Kings, Queens, and Jacks are worth 10 points.
      Aces are worth 1 or 11 points.
      Cards 2 through 10 are worth their face value.
      (H)it to take another card.
      (S)tand to stop taking cards.
      On your first play, you can (D)ouble down to increase your bet
      but must hit exactly one more time before standing.
      In case of a tie, the bet is returned to the player.
      The dealer stops hitting at 17.
Money: 5000
How much do you bet? (1-5000, or QUIT)
> 400
Bet: 400

DEALER: ???
 ___   ___
|## | |2  |
|###| | ♥ |
|_##| |__2|

PLAYER: 17
 ___   ___
|K  | |7  |
| ♠ | | ♦ |
|__K| |__7|


(H)it, (S)tand, (D)ouble down
> h
You drew a 4 of ♦.
--сокращено--
DEALER: 18
 ___   ___   ___
|K  | |2  | |6  |
| ♦ | | ♥ | | ♠ |
|__K| |__2| |__6|

PLAYER: 21
 ___   ___   ___
|K  | |7  | |4  |
| ♠ | | ♦ | | ♦ |
|__K| |__7| |__4|

You won $400!
--сокращено--
```

Описание работы

Символов для мастей карт на клавиатуре нет, поэтому нам приходится вызывать `chr()` для их отображения. Передаваемое в `chr()` целое число называется *кодом символа* (code point) кодировки Unicode, оно представляет собой уникальное число, идентифицирующее символ в соответствии со стандартом Unicode. Unicode часто понимают неправильно. Прекрасное введение в Unicode — доклад Неда Бачхелдера (Ned Batchelder) Pragmatic Unicode, or How Do I Stop the Pain?, сделанный на конференции PyCon US 2012. Вы можете найти его по адресу https://youtu.be/sgHbC6udIqc/. В приложении Б приведен полный список символов Unicode, которые можно использовать в программах Python.

```
 1. """Блек-джек, (c) Эл Свейгарт al@inventwithpython.com
 2. Классическая карточная игра, известная также как "двадцать одно".
 3. (В этой версии нет страхования или разбиения.)
 4. Больше информации — в статье https://ru.wikipedia.org/wiki/Блэкджек
 5. Код размещен на сайте https://nostarch.com/big-book-small-python-projects
 6. Теги: большая, игра, карточная игра"""
 7.
 8. import random, sys
 9.
10. # Задаем значения констант:
11. HEARTS   = chr(9829) # Символ 9829 — '♥'.
12. DIAMONDS = chr(9830) # Символ 9830 — '♦'.
13. SPADES   = chr(9824) # Символ 9824 — '♠'.
14. CLUBS    = chr(9827) # Символ 9827 — '♣'.
15. # (Список кодов chr можно найти в https://inventwithpython.com/charactermap)
16. BACKSIDE = 'backside'
17.
18.
19. def main():
20.     print('''Blackjack, by Al Sweigart al@inventwithpython.com
21.
22.     Rules:
23.       Try to get as close to 21 without going over.
24.       Kings, Queens, and Jacks are worth 10 points.
25.       Aces are worth 1 or 11 points.
26.       Cards 2 through 10 are worth their face value.
27.       (H)it to take another card.
28.       (S)tand to stop taking cards.
29.       On your first play, you can (D)ouble down to increase your bet
30.       but must hit exactly one more time before standing.
31.       In case of a tie, the bet is returned to the player.
32.       The dealer stops hitting at 17.''')
33.
34.     money = 5000
35.     while True:  # Основной цикл игры.
36.         # Проверяем, не закончились ли у игрока деньги:
37.         if money <= 0:
```

```
38.             print("You're broke!")
39.             print("Good thing you weren't playing with real money.")
40.             print('Thanks for playing!')
41.             sys.exit()
42.
43.         # Даем возможность игроку сделать ставку на раунд:
44.         print('Money:', money)
45.         bet = getBet(money)
46.
47.         # Сдаем дилеру и игроку по две карты из колоды:
48.         deck = getDeck()
49.         dealerHand = [deck.pop(), deck.pop()]
50.         playerHand = [deck.pop(), deck.pop()]
51.
52.         # Обработка действий игрока:
53.         print('Bet:', bet)
54.         while True:  # Выполняем цикл до тех пор, пока игрок
                         # не скажет "хватит" или у него не будет перебор.
55.             displayHands(playerHand, dealerHand, False)
56.             print()
57.
58.             # Проверка на перебор у игрока:
59.             if getHandValue(playerHand) > 21:
60.                 break
61.
62.             # Получаем ход игрока: H, S или D:
63.             move = getMove(playerHand, money - bet)
64.
65.             # Обработка действий игрока:
66.             if move == 'D':
67.                 # Игрок удваивает, он может увеличить ставку:
68.                 additionalBet = getBet(min(bet, (money - bet)))
69.                 bet += additionalBet
70.                 print('Bet increased to {}.'.format(bet))
71.                 print('Bet:', bet)
72.
73.             if move in ('H', 'D'):
74.                 # "Еще" или "удваиваю": игрок берет еще одну карту.
75.                 newCard = deck.pop()
76.                 rank, suit = newCard
77.                 print('You drew a {} of {}.'.format(rank, suit))
78.                 playerHand.append(newCard)
79.
80.                 if getHandValue(playerHand) > 21:
81.                     # Перебор у игрока:
82.                     continue
83.
84.             if move in ('S', 'D'):
85.                 # "Хватит" или "удваиваю": переход хода к следующему игроку
86.                 break
87.
```

```
        # Обработка действий дилера:
        if getHandValue(playerHand) <= 21:
            while getHandValue(dealerHand) < 17:
                # Дилер берет еще карту:
                print('Dealer hits...')
                dealerHand.append(deck.pop())
                displayHands(playerHand, dealerHand, False)

                if getHandValue(dealerHand) > 21:
                    break  # Перебор у дилера.
                input('Press Enter to continue...')
                print('\n\n')

        # Отображает итоговые карты на руках:
        displayHands(playerHand, dealerHand, True)

        playerValue = getHandValue(playerHand)
        dealerValue = getHandValue(dealerHand)
        # Проверяем, игрок выиграл, проиграл или сыграл вничью:
        if dealerValue > 21:
            print('Dealer busts! You win ${}!'.format(bet))
            money += bet
        elif (playerValue > 21) or (playerValue < dealerValue):
            print('You lost!')
            money -= bet
        elif playerValue > dealerValue:
            print('You won ${}!'.format(bet))
            money += bet
        elif playerValue == dealerValue:
            print('It\'s a tie, the bet is returned to you.')

        input('Press Enter to continue...')
        print('\n\n')

def getBet(maxBet):
    """Спрашиваем у игрока, сколько он ставит на этот раунд."""
    while True:  # Продолжаем спрашивать, пока не будет введено
                 # допустимое значение.
        print('How much do you bet? (1-{}, or QUIT)'.format(maxBet))
        bet = input('> ').upper().strip()
        if bet == 'QUIT':
            print('Thanks for playing!')
            sys.exit()

        if not bet.isdecimal():
            continue  # Если игрок не ответил — спрашиваем снова.

        bet = int(bet)
        if 1 <= bet <= maxBet:
            return bet  # Игрок ввел допустимое значение ставки.

```

```
139.
140. def getDeck():
141.     """Возвращаем список кортежей (номинал, масть) для всех 52 карт."""
142.     deck = []
143.     for suit in (HEARTS, DIAMONDS, SPADES, CLUBS):
144.         for rank in range(2, 11):
145.             deck.append((str(rank), suit))  # Добавляем числовые карты.
146.         for rank in ('J', 'Q', 'K', 'A'):
147.             deck.append((rank, suit))  # Добавляем фигурные карты и тузы.
148.     random.shuffle(deck)
149.     return deck
150.
151.
152. def displayHands(playerHand, dealerHand, showDealerHand):
153.     """Отображаем карты игрока и дилера. Скрываем первую карту дилера,
154.     если showDealerHand равно False."""
155.     print()
156.     if showDealerHand:
157.         print('DEALER:', getHandValue(dealerHand))
158.         displayCards(dealerHand)
159.     else:
160.         print('DEALER: ???')
161.         # Скрываем первую карту дилера:
162.         displayCards([BACKSIDE] + dealerHand[1:])
163.
164.     # Отображаем карты игрока:
165.     print('PLAYER:', getHandValue(playerHand))
166.     displayCards(playerHand)
167.
168.
169. def getHandValue(cards):
170.     """ Возвращаем стоимость карт. Фигурные карты стоят 10, тузы — 11
171.     или 1 очко (эта функция выбирает подходящую стоимость карты)."""
172.     value = 0
173.     numberOfAces = 0
174.
175.     # Добавляем стоимость карты — не туза:
176.     for card in cards:
177.         rank = card[0]  # карта представляет собой кортеж (номинал, масть)
178.         if rank == 'A':
179.             numberOfAces += 1
180.         elif rank in ('K', 'Q', 'J'):  # Фигурные карты стоят 10 очков.
181.             value += 10
182.         else:
183.             value += int(rank)  # Стоимость числовых карт равна их номиналу.
184.
185.     # Добавляем стоимость для тузов:
186.     value += numberOfAces  # Добавляем 1 для каждого туза.
187.     for i in range(numberOfAces):
188.         # Если можно добавить еще 10 с перебором, добавляем:
189.         if value + 10 <= 21:
```

```
            value += 10

    return value


def displayCards(cards):
    """Отображаем все карты из списка карт."""
    rows = ['', '', '', '', '']  # Отображаемый в каждой строке текст.

    for i, card in enumerate(cards):
        rows[0] += ' ___  '  # Выводим верхнюю строку карты.
        if card == BACKSIDE:
            # Выводим рубашку карты:
            rows[1] += '|## | '
            rows[2] += '|###| '
            rows[3] += '|_##| '
        else:
            # Выводим лицевую сторону карты:
            rank, suit = card  # Карта — структура данных типа кортеж.
            rows[1] += '|{} | '.format(rank.ljust(2))
            rows[2] += '| {} | '.format(suit)
            rows[3] += '|_{}| '.format(rank.rjust(2, '_'))

    # Выводим все строки на экран:
    for row in rows:
        print(row)

def getMove(playerHand, money):
    """Спрашиваем, какой ход хочет сделать игрок, и возвращаем 'H', если он
    хочет взять еще карту, 'S', если ему хватит, и 'D', если он удваивает."""
    while True:  # Продолжаем итерации цикла, пока игрок не сделает
                 # допустимый ход.
        # Определяем, какие ходы может сделать игрок:
        moves = ['(H)it', '(S)tand']

        # Игрок может удвоить при первом ходе, это ясно из того,
        # что у игрока ровно две карты:
        if len(playerHand) == 2 and money > 0:
            moves.append('(D)ouble down')

        # Получаем ход игрока:
        movePrompt = ', '.join(moves) + '> '
        move = input(movePrompt).upper()
        if move in ('H', 'S'):
            return move  # Игрок сделал допустимый ход.
        if move == 'D' and '(D)ouble down' in moves:
            return move  # Игрок сделал допустимый ход.


# Если программа не импортируется, а запускается, производим запуск:
if __name__ == '__main__':
    main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. У игры в блек-джек есть несколько необязательных правил, которые вы можете реализовать. Например, если стоимость первых двух карт одинакова, то игрок может разбить их на две «руки» и делать ставки на каждую по отдельности. Кроме того, если игрок получает блекджек (набор из туза пик и валета черной масти) в качестве первых двух карт, то выигрывает десять к одному. Больше об этой карточной игре вы можете узнать из статьи «Википедии»: https://ru.wikipedia.org/wiki/Блэкджек.

Исследование программы

Попробуйте ответить на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Как сделать так, чтобы игрок начинал с другой суммой денег?
2. Как наша программа препятствует тому, чтобы игрок поставил больше денег, чем у него есть?
3. Как в программе представляется отдельная карта?
4. Как в программе представляются все карты на руках кого-либо?
5. Чему соответствует каждое из строковых значений в списке `list` (созданном в строке 197)?
6. Что произойдет, если удалить или закомментировать `random.shuffle(deck)` в строке 148?
7. Что будет, если оператор `money -= bet` в строке 112 заменить на `money += bet`?
8. Что будет, если передавать значение `True` для `showDealerHand` в функцию `displayHands()`? И что происходит, если передавать `False`?

5

Отскакивающий от краев логотип DVD

Если вы старше определенного возраста, то наверняка помните эти древние технические устройства — DVD-проигрыватели. Во время простоя на экране появлялся путешествующий по диагонали логотип DVD, отражающийся от границ. Приведенная программа имитирует такой цветной логотип DVD, меняющий направление при каждом столкновении с краем экрана. Мы также отслеживаем количество таких столкновений. В результате получается интересная анимация, особенно в тот волшебный момент, когда логотип попадает точно в угол.

Эту программу нельзя запустить из среды интегрированной разработки (IDE) или редактора, поскольку в ней используется модуль `bext`. Следовательно, для правильного отображения ее необходимо запустить из командной строки или терминала. Больше информации о модуле `bext` можно найти на сайте https://pypi.org/.

Программа в действии

Результат выполнения `bouncingdvd.py` выглядит следующим образом (рис. 5.1).

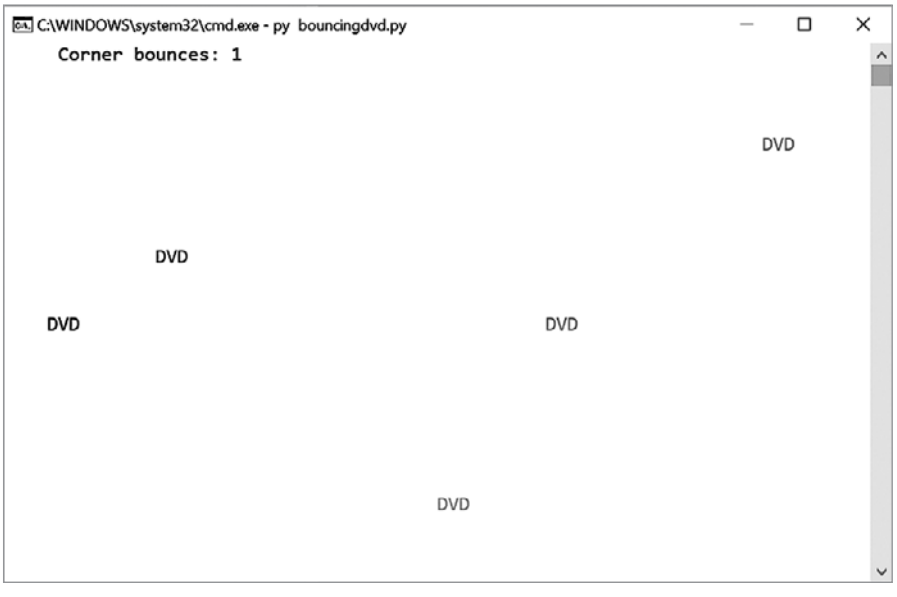


Рис. 5.1. Движущиеся по диагонали логотипы DVD в программе bouncingdvd.py

Описание работы

Наверняка из курса школьной математики вы помните, что такое декартовы координаты. В программировании координата *x* соответствует позиции объекта по горизонтали, а координата *y* — по вертикали, как и в математике. Однако, в отличие от математики, точка начала координат (0, 0) находится в верхнем левом углу экрана, и координата *y* растет по мере движения вниз. Координата *x* растет по мере движения объекта вправо, как и в математике. На рис. 5.2 показана система координат экрана.

Функция `goto()` модуля `bext` ведет себя аналогичным образом: при вызове `bext.goto(0, 0)` курсор ввода текста помещается в верхнюю левую точку окна терминала. Каждый отражающийся логотип представлен в нашей программе с помощью ассоциативного массива с ключами `'color'`, `'direction'`, `'x'` и `'y'`. Целочисленные значения `'x'` и `'y'` отражают местоположение логотипа на экране. Поскольку эти значения передаются в функцию `bext.goto()`, их увеличение будет сдвигать логотип вправо и вниз, а уменьшение — перемещать влево и вверх.

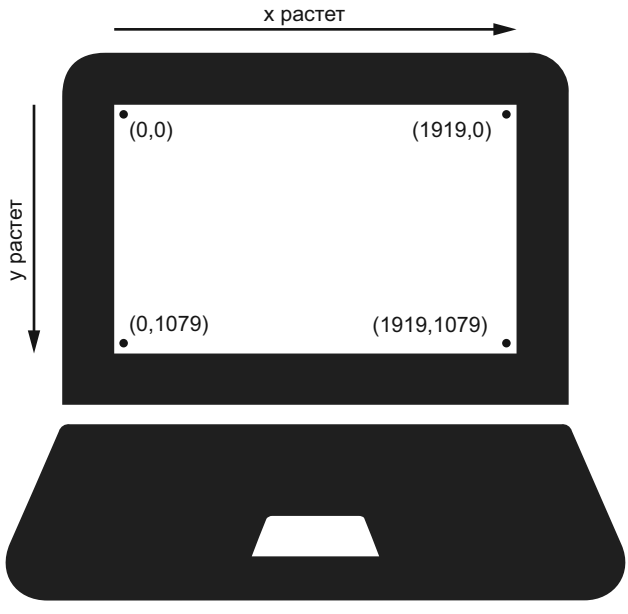


Рис. 5.2. Точка начала координат (0, 0) находится в верхнем левом углу экрана, и координаты x и y растут по мере движения вправо и вниз соответственно

```
"""Отскакивающий от краев логотип DVD, (c) Эл Свейгарт al@inventwithpython.com
Анимация отскакивающего логотипа DVD. Оценить ее по достоинству могут
только люди "определенного возраста". Для останова нажмите Ctrl+C.

Примечание: не меняйте размера окна терминала во время работы программы.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: короткая, графика, bext"""

import sys, random, time

try:
    import bext
except ImportError:
    print('This program requires the bext module, which you')
    print('can install by following the instructions at')
    print('https://pypi.org/project/Bext/')
    sys.exit()

# Задаем константы:
WIDTH, HEIGHT = bext.size()
# В Windows нельзя вывести символ в последний столбец без добавления
# автоматически символа новой строки, так что уменьшаем ширину на 1:
WIDTH -= 1

```

```
25. NUMBER_OF_LOGOS = 5  # (!) Попробуйте заменить на 1 или 100.
26. PAUSE_AMOUNT = 0.2  # (!) Попробуйте заменить на 1.0 или 0.0.
27. # (!) Попробуйте уменьшить количество цветов в этом списке:
28. COLORS = ['red', 'green', 'yellow', 'blue', 'magenta', 'cyan', 'white']
29.
30. UP_RIGHT   = 'ur'
31. UP_LEFT    = 'ul'
32. DOWN_RIGHT = 'dr'
33. DOWN_LEFT  = 'dl'
34. DIRECTIONS = (UP_RIGHT, UP_LEFT, DOWN_RIGHT, DOWN_LEFT)
35.
36. # Названия ключей для ассоциативных массивов логотипов:
37. COLOR = 'color'
38. X = 'x'
39. Y = 'y'
40. DIR = 'direction'
41.
42.
43. def main():
44.     bext.clear()
45.
46.     # Генерация логотипов.
47.     logos = []
48.     for i in range(NUMBER_OF_LOGOS):
49.         logos.append({COLOR: random.choice(COLORS),
50.                       X: random.randint(1, WIDTH - 4),
51.                       Y: random.randint(1, HEIGHT - 4),
52.                       DIR: random.choice(DIRECTIONS)})
53.         if logos[-1][X] % 2 == 1:
54.             # Гарантируем, что X четное число, для столкновения с углом.
55.             logos[-1][X] -= 1
56.
57.     cornerBounces = 0  # Считаем, сколько раз логотип столкнулся с углом.
58.     while True:  # Основной цикл программы.
59.         for logo in logos:  # Обрабатываем все логотипы в списке логотипов.
60.             # Очищаем место, где ранее находился логотип:
61.             bext.goto(logo[X], logo[Y])
62.             print('   ', end='')  # (!) Попробуйте закомментировать строку.
63.
64.             originalDirection = logo[DIR]
65.
66.             # Проверяем, не отскакивает ли логотип от угла:
67.             if logo[X] == 0 and logo[Y] == 0:
68.                 logo[DIR] = DOWN_RIGHT
69.                 cornerBounces += 1
70.             elif logo[X] == 0 and logo[Y] == HEIGHT - 1:
71.                 logo[DIR] = UP_RIGHT
72.                 cornerBounces += 1
73.             elif logo[X] == WIDTH - 3 and logo[Y] == 0:
74.                 logo[DIR] = DOWN_LEFT
75.                 cornerBounces += 1
76.             elif logo[X] == WIDTH - 3 and logo[Y] == HEIGHT - 1:
```

```
 77.                 logo[DIR] = UP_LEFT
 78.                 cornerBounces += 1
 79.
 80.             # Проверяем, не отскакивает ли логотип от левого края:
 81.             elif logo[X] == 0 and logo[DIR] == UP_LEFT:
 82.                 logo[DIR] = UP_RIGHT
 83.             elif logo[X] == 0 and logo[DIR] == DOWN_LEFT:
 84.                 logo[DIR] = DOWN_RIGHT
 85.
 86.             # Проверяем, не отскакивает ли логотип от правого края:
 87.             # (WIDTH - 3, поскольку 'DVD' состоит из трех букв.)
 88.             elif logo[X] == WIDTH - 3 and logo[DIR] == UP_RIGHT:
 89.                 logo[DIR] = UP_LEFT
 90.             elif logo[X] == WIDTH - 3 and logo[DIR] == DOWN_RIGHT:
 91.                 logo[DIR] = DOWN_LEFT
 92.
 93.             # Проверяем, не отскакивает ли логотип от верхнего края:
 94.             elif logo[Y] == 0 and logo[DIR] == UP_LEFT:
 95.                 logo[DIR] = DOWN_LEFT
 96.             elif logo[Y] == 0 and logo[DIR] == UP_RIGHT:
 97.                 logo[DIR] = DOWN_RIGHT
 98.
 99.             # Проверяем, не отскакивает ли логотип от нижнего края:
100.             elif logo[Y] == HEIGHT - 1 and logo[DIR] == DOWN_LEFT:
101.                 logo[DIR] = UP_LEFT
102.             elif logo[Y] == HEIGHT - 1 and logo[DIR] == DOWN_RIGHT:
103.                 logo[DIR] = UP_RIGHT
104.
105.             if logo[DIR] != originalDirection:
106.                 # Меняем цвет при отскакивании логотипа:
107.                 logo[COLOR] = random.choice(COLORS)
108.
109.             # Перемещаем логотип. (Координата X меняется на 2, поскольку
110.             # в терминале высота символов вдвое превышает ширину.)
111.             if logo[DIR] == UP_RIGHT:
112.                 logo[X] += 2
113.                 logo[Y] -= 1
114.             elif logo[DIR] == UP_LEFT:
115.                 logo[X] -= 2
116.                 logo[Y] -= 1
117.             elif logo[DIR] == DOWN_RIGHT:
118.                 logo[X] += 2
119.                 logo[Y] += 1
120.             elif logo[DIR] == DOWN_LEFT:
121.                 logo[X] -= 2
122.                 logo[Y] += 1
123.
124.         # Отображает количество отскакиваний от углов:
125.         bext.goto(5, 0)
126.         bext.fg('white')
127.         print('Corner bounces:', cornerBounces, end='')
128.
```

```
129.         for logo in logos:
130.             # Отрисовывает логотипы на новом месте:
131.             bext.goto(logo[X], logo[Y])
132.             bext.fg(logo[COLOR])
133.             print('DVD', end='')
134.
135.         bext.goto(0, 0)
136.
137.         sys.stdout.flush()  # (Нужно для программ, использующих модуль bext.)
138.         time.sleep(PAUSE_AMOUNT)
139.
140.
141. # Если программа не импортируется, а запускается, производим запуск:
142. if __name__ == '__main__':
143.     try:
144.         main()
145.     except KeyboardInterrupt:
146.         print()
147.         print('Bouncing DVD Logo, by Al Sweigart')
148.         sys.exit()  # При нажатии Ctrl+C завершаем выполнение программы.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- изменить NUMBER_OF_LOGOS, чтобы увеличить количество отражающихся логотипов на экране;
- изменить PAUSE_AMOUNT, чтобы ускорить или замедлить движение логотипов.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `WIDTH, HEIGHT = bext.size()` в строке 20 заменить на `WIDTH, HEIGHT = 10, 5`?
2. Что будет, если `DIR: random.choice(DIRECTIONS)` в строке 52 заменить на `DIR: DOWN_RIGHT`?
3. Как сделать, чтобы текст `'Corner bounces:'` не появлялся на экране?
4. Какое сообщение об ошибке вы получите, если удалите или закомментируете `cornerBounces = 0` в строке 57?

6

Шифр Цезаря

Шифр Цезаря — древний алгоритм шифрования, использовавшийся Юлием Цезарем. Буквы в нем шифруются путем сдвига их на определенное количество позиций в алфавите. Дистанция сдвига называется *ключом*. Например, если ключ равен 3, то A превращается в D, B — в E, C — в F и т. д. Для расшифровки сообщения необходимо выполнить сдвиг зашифрованных символов в противоположном направлении. Данная программа позволяет шифровать и расшифровывать сообщения в соответствии с приведенным алгоритмом.

Сейчас шифр Цезаря выглядит очень простым и именно потому подходит для начинающих. Программа из проекта 7 позволяет расшифровывать сообщения, даже если исходный ключ неизвестен, путем прямого перебора всех 26 возможных ключей. Кроме того, при шифровании сообщений с ключом 13 шифр Цезаря превращается в шифр ROT13 из проекта 61. Узнать больше о шифре Цезаря можно в статье «Википедии»: https://ru.wikipedia.org/wiki/Шифр_Цезаря. Если вам интересны шифры и их взлом вообще, то можете прочитать мою книгу *Cracking Codes with Python*[1] (издательство No Starch Press, 2018; https://nostarch.com/crackingcodes/).

[1] *Свейгарт Э.* Криптография и взлом шифров на Python. — Киев: Диалектика, 2019.

Программа в действии

Результат выполнения `caesarcipher.py` выглядит следующим образом:

```
Caesar Cipher, by Al Sweigart al@inventwithpython.com
Do you want to (e)ncrypt or (d)ecrypt?
> e
Please enter the key (0 to 25) to use.
> 4
Enter the message to encrypt.
> Meet me by the rose bushes tonight.
QIIX QI FC XLI VSWI FYWLIW XSRMKLX.
Full encrypted text copied to clipboard.

Caesar Cipher, by Al Sweigart al@inventwithpython.com
Do you want to (e)ncrypt or (d)ecrypt?
> d
Please enter the key (0 to 26) to use.
> 4
Enter the message to decrypt.
> QIIX QI FC XLI VSWI FYWLIW XSRMKLX.
MEET ME BY THE ROSE BUSHES TONIGHT.
Full decrypted text copied to clipboard.
```

Описание работы

Подобно большинству программ шифрования, шифр Цезаря преобразует буквы в числа, производит над ними определенные математические операции и преобразует их обратно в буквы текста. В контексте шифрования мы будем называть элементы текста *символами*. Символы могут быть буквами, цифрами и знаками препинания, каждому из которых соответствует уникальное целочисленное значение. В случае программы для шифра Цезаря все символы представляют собой буквы, а целочисленные значения — их позиции в строке `SYMBOLS`: `'ABCDEFGHIJKLMNOPQRSTUVWXYZ'`.

```
 1. """Шифр Цезаря, (c) Эл Свейгарт al@inventwithpython.com
 2. Шифр Цезаря – шифр сдвига, в котором шифрование и дешифровка букв
 3. производятся путем сложения и вычитания соответствующих чисел.
 4. Дополнительная информация: https://ru.wikipedia.org/wiki/Шифр_Цезаря
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, для начинающих, криптография, математическая"""
 7.
 8. try:
 9.     import pyperclip  # pyperclip копирует текст в буфер обмена.
10. except ImportError:
11.     pass  # Если библиотека pyperclip не установлена, ничего не делаем.
12.
13. # Все возможные символы для шифрования/дешифровки:
14. # (!) Можете добавить также символы для цифр и знаков препинания,
15. # чтобы шифровать и их.
```

```
16. SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
17.
18. print('Caesar Cipher, by Al Sweigart al@inventwithpython.com')
19. print('The Caesar cipher encrypts letters by shifting them over by a')
20. print('key number. For example, a key of 2 means the letter A is')
21. print('encrypted into C, the letter B encrypted into D, and so on.')
22. print()
23.
24. # Спрашиваем у пользователя, хочет ли он шифровать или расшифровывать:
25. while True:  # Повторяем вопрос, пока пользователь не введет e или d.
26.     print('Do you want to (e)ncrypt or (d)ecrypt?')
27.     response = input('> ').lower()
28.     if response.startswith('e'):
29.         mode = 'encrypt'
30.         break
31.     elif response.startswith('d'):
32.         mode = 'decrypt'
33.         break
34.     print('Please enter the letter e or d.')
35.
36. # Просим пользователя ввести ключ шифрования:
37. while True:  # Повторяем вопрос, пока пользователь не введет допустимый ключ.
38.     maxKey = len(SYMBOLS) - 1
39.     print('Please enter the key (0 to {}) to use.'.format(maxKey))
40.     response = input('> ').upper()
41.     if not response.isdecimal():
42.         continue
43.
44.     if 0 <= int(response) < len(SYMBOLS):
45.         key = int(response)
46.         break
47.
48. # Просим пользователя ввести сообщение для шифрования/расшифровки:
49. print('Enter the message to {}.'.format(mode))
50. message = input('> ')
51.
52. # Шифр Цезаря работает только с символами в верхнем регистре:
53. message = message.upper()
54.
55. # Для хранения зашифрованного/расшифрованного варианта сообщения:
56. translated = ''
57.
58. # Зашифровываем/расшифровываем каждый символ сообщения:
59. for symbol in message:
60.     if symbol in SYMBOLS:
61.         # Получаем зашифрованное (расшифрованное) числовое значение для символа.
62.         num = SYMBOLS.find(symbol)  # Получаем числовое значение символа
63.         if mode == 'encrypt':
64.             num = num + key
65.         elif mode == 'decrypt':
66.             num = num - key
```

```
67.
68.         # Производим переход по кругу, если число больше длины SYMBOLS
69.         # или меньше 0:
70.         if num >= len(SYMBOLS):
71.             num = num - len(SYMBOLS)
72.         elif num < 0:
73.             num = num + len(SYMBOLS)
74.         # Добавляем соответствующий числу зашифрованный/расшифрованный символ
75.         # в translated:
76.         translated = translated + SYMBOLS[num]
77.     else:
78.         # Просто добавляем символ без шифрования/расшифровки:
79.         translated = translated + symbol
80.
81. # Выводим зашифрованную/расшифрованную строку на экран:
82. print(translated)
83.
84. try:
85.     pyperclip.copy(translated)
86.     print('Full {}ed text copied to clipboard.'.format(mode))
87. except:
88.     pass  # Если pyperclip не установлена, ничего не делаем.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (!). При желании можете расширить спектр шифруемых символов, добавив дополнительные в строковое значение SYMBOLS.

Исследование программы

Попробуйте ответить на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'` в строке 16 заменить на `SYMBOLS = 'ABC'`?
2. Что будет, если зашифровать сообщение с ключом `0`?
3. Какое сообщение об ошибке вы получите, если удалите или закомментируете `translated = ''` в строке 56?
4. Какое сообщение об ошибке вы получите, если удалите или закомментируете `key = int(response)` в строке 45?
5. Что будет, если `translated = translated + SYMBOLS[num]` в строке 76 заменить на `translated = translated + symbol`?

7

Взлом шифра Цезаря

Эта программа способна взламывать сообщения, зашифрованные с помощью шифра Цезаря из проекта 6, даже если ключ неизвестен. Существует всего 26 возможных ключей для шифра Цезаря, так что компьютер может легко перебрать все возможные расшифровки и отобразить пользователю результаты. В криптографии подобная методика называется *атакой методом подбора* (brute-force attack). Если вам интересно узнать больше о шифрах и их взломе, то можете прочитать мою книгу *Cracking Codes with Python*.

Программа в действии

Результат выполнения `caesarhacker.py` выглядит следующим образом:

```
Caesar Cipher Hacker, by Al Sweigart al@inventwithpython.com
Enter the encrypted Caesar cipher message to hack.
> QIIX QI FC XLI VSWI FYWLIW XSRMKLX.
Key #0: QIIX QI FC XLI VSWI FYWLIW XSRMKLX.
Key #1: PHHW PH EB WKH URVH EXVKHV WRQLJKW.
Key #2: OGGV OG DA VJG TQUG DWUJGU VQPKIJV.
Key #3: NFFU NF CZ UIF SPTF CVTIFT UPOJHIU.
Key #4: MEET ME BY THE ROSE BUSHES TONIGHT.
Key #5: LDDS LD AX SGD QNRD ATRGDR SNMHFGS.
Key #6: KCCR KC ZW RFC PMQC ZSQFCQ RMLGEFR.
--сокращено--
```

Описание работы

Обратите внимание, что строки с 20-й по 36-ю данной программы практически идентичны строкам с 55-й по 78-ю программы для шифра Цезаря. В программе взлома шифра Цезаря реализован тот же код дешифрования, только в цикле `for`, в целях выполнения этого кода для всех возможных ключей.

К сожалению, наша программа взлома не настолько совершенна, чтобы определить, какой из ключей — нужный, поэтому выведенные результаты должен прочитать человек и определить, какая из расшифровок представляет собой исходное сообщение на английском (или каком-либо другом письменном языке, который был зашифрован). В главе 11 книги *Cracking Codes with Python* подробнее рассказывается, как написать код на Python для выявления сообщений на английском.

```
 1. """Взлом шифра Цезаря, (c) Эл Свейгарт al@inventwithpython.com
 2. Данная программа взламывает сообщения, зашифрованные шифром Цезаря,
 3. путем прямого перебора всех возможных ключей.
 4. Больше информации – по адресу
 5. https://ru.wikipedia.org/wiki/Шифр_Цезаря#Взлом_шифра
 6. Код размещен на https://nostarch.com/big-book-small-python-projects
 7. Теги: крошечная, для начинающих, криптография, математическая"""
 8.
 9. print('Caesar Cipher Hacker, by Al Sweigart al@inventwithpython.com')
10.
11. # Просим пользователя ввести сообщение для взлома:
12. print('Enter the encrypted Caesar cipher message to hack.')
13. message = input('> ')
14.
15. # Все возможные символы для шифрования/дешифровки:
16. # (должно совпадать с набором SYMBOLS, использовавшимся при шифровании)
17. SYMBOLS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
18.
19. for key in range(len(SYMBOLS)):  # Проходим в цикле по всем возможным ключам.
20.     translated = ''
21.
22.     # Расшифровываем каждый символ в сообщении:
23.     for symbol in message:
24.         if symbol in SYMBOLS:
25.             num = SYMBOLS.find(symbol)  # Получаем числовое значение символа.
26.             num = num - key  # Расшифровываем числовое значение.
27.
28.             # Выполняем переход по кругу, если число меньше 0:
29.             if num < 0:
30.                 num = num + len(SYMBOLS)
31.             # Добавляем соответствующий числу расшифрованный символ
32.             # в translated:
33.             translated = translated + SYMBOLS[num]
34.         else:
35.             # Просто добавляем символ без расшифровки:
```

```
            translated = translated + symbol

    # Выводим проверяемый ключ вместе с расшифрованным на его основе текстом:
    print('Key #{}: {}'.format(key, translated))
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Учтите, что строковое значение в переменной `SYMBOLS` должно совпадать с переменной `SYMBOLS` в сгенерировавшей исходный текст программе шифрования.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если удалите или закомментируете `translated = ''` в строке 20?
2. Что будет, если `translated = translated + SYMBOLS[num]` в строке 33 заменить на `translated = translated + symbol`?
3. Что будет, если ввести в программу взлома шифра Цезаря незашифрованный текст?

8

Генерация календарей

Эта программа генерирует подходящие для распечатки текстовые файлы с календарями на выбранный месяц (в выбранном году). Даты и календари — каверзный вопрос в программировании, поскольку существует очень много различных правил для определения количества дней в месяце, високосных годов, а также того, каким днем недели является конкретная дата. К счастью, модуль `datetime` решает все эти проблемы вместо вас. Данная программа генерирует многострочное строковое значение, содержащее страницу календаря на месяц.

Программа в действии

Результат выполнения `calendarmaker.py` выглядит следующим образом:

```
Calendar Maker, by Al Sweigart al@inventwithpython.com
Enter the year for the calendar:
> 2029
Enter the month for the calendar, 1-12:
> 12
```

```
                              December 2029
...Sunday.....Monday....Tuesday...Wednesday...Thursday....Friday....Saturday..
+----------+----------+----------+----------+----------+----------+----------+
|25        |26        |27        |28        |29        |30        | 1        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
| 2        | 3        | 4        | 5        | 6        | 7        | 8        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
| 9        |10        |11        |12        |13        |14        |15        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
|16        |17        |18        |19        |20        |21        |22        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
|23        |24        |25        |26        |27        |28        |29        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
|30        |31        | 1        | 2        | 3        | 4        | 5        |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
|          |          |          |          |          |          |          |
+----------+----------+----------+----------+----------+----------+----------+
Saved to calendar_2029_12.txt
```

Описание работы

Обратите внимание, что функция `getCalendarFor()` возвращает гигантское многострочное строковое значение с календарем на заданный месяц в заданном году. Упомянутое значение в этой функции хранится в переменной `calText`, в которую постепенно добавляются линии, пробелы и даты. Чтобы можно было отслеживать даты, переменная `currentDate` содержит объект `datetime.date()`, которому присваивается значение следующей или предыдущей даты путем сложения или вычитания объектов `datetime.timedelta()`. Узнать больше о модулях Python для работы с временем и датами вы можете из главы 17 моей книги *Automate the Boring Stuff with Python* по адресу https://automatetheboringstuff.com/2e/chapter17/.

```
"""Генерация календарей, (c) Эл Свейгарт al@inventwithpython.com
Создает календари на месяц, готовые для распечатки, и сохраняет их
в текстовом файле. Код размещен на https://nostarch.com/big-book-small-
python-projects Теги: короткая"""

import datetime

# Задаем константы:
DAYS = ('Sunday', 'Monday', 'Tuesday', 'Wednesday', 'Thursday',
        'Friday', 'Saturday')
MONTHS = ('January', 'February', 'March', 'April', 'May', 'June', 'July',
          'August', 'September', 'October', 'November', 'December')

print('Calendar Maker, by Al Sweigart al@inventwithpython.com')

while True:  # Цикл для запроса у пользователя года.
    print('Enter the year for the calendar:')
    response = input('> ')

    if response.isdecimal() and int(response) > 0:
        year = int(response)
        break

    print('Please enter a numeric year, like 2023.')
    continue

while True:  # Цикл для запроса у пользователя месяца.
    print('Enter the month for the calendar, 1-12:')
    response = input('> ')

    if not response.isdecimal():
        print('Please enter a numeric month, like 3 for March.')
        continue

    month = int(response)
    if 1 <= month <= 12:
        break

    print('Please enter a number from 1 to 12.')


def getCalendarFor(year, month):
    calText = ''  # calText содержит строковое значение с календарем.

    # Размещаем месяц и год вверху календаря:
    calText += (' ' * 34) + MONTHS[month - 1] + ' ' + str(year) + '\n'

    # Добавляем в календарь метки дней недели:
    # (!) Попробуйте заменить их аббревиатурами: SUN, MON, TUE и т. д.
    calText += '...Sunday.....Monday....Tuesday...Wednesday...
    Thursday....Friday....Saturday..\n'
```

```
51.
52.     # Горизонтальная линия — разделитель недель:
53.     weekSeparator = ('+----------' * 7) + '+\n'
54.
55.     # Пустые строки содержат по десять пробелов между разделителями дней |:
56.     blankRow = ('|          ' * 7) + '|\n'
57.
58.     # Получаем первую дату месяца. (Модуль datetime берет на себя
59.     # все сложные нюансы календарных вычислений.)
60.     currentDate = datetime.date(year, month, 1)
61.
62.     # Отнимаем от currentDate по дню, пока не дойдем до воскресенья.
63.     # (weekday() возвращает для воскресенья 6, а не 0.)
64.     while currentDate.weekday() != 6:
65.         currentDate -= datetime.timedelta(days=1)
66.
67.     while True:  # Проходим в цикле по всем неделям в месяце.
68.         calText += weekSeparator
69.
70.         # dayNumberRow — строка с метками номеров дней:
71.         dayNumberRow = ''
72.         for i in range(7):
73.             dayNumberLabel = str(currentDate.day).rjust(2)
74.             dayNumberRow += '|' + dayNumberLabel + (' ' * 8)
75.             currentDate += datetime.timedelta(days=1) # Переходим к следующему
76.                                                       # дню.
77.         dayNumberRow += '|\n'  # Добавляем вертикальную линию после субботы.
78.         # Добавляем в текст календаря строку с номерами дней и 3 пустых строки.
79.         calText += dayNumberRow
80.         for i in range(3):  # (!) Попробуйте заменить 3 на 5 или 10.
81.             calText += blankRow
82.
83.         # Проверяем, закончили ли мы обработку месяца:
84.         if currentDate.month != month:
85.             break
86.
87.     # Добавляем горизонтальную линию в самом низу календаря.
88.     calText += weekSeparator
89.     return calText
90.
91.
92. calText = getCalendarFor(year, month)
93. print(calText)  # Выводим календарь.
94.
95. # Сохраняем календарь в текстовом файле:
96. calendarFilename = 'calendar_{}_{}.txt'.format(year, month)
97. with open(calendarFilename, 'w') as fileObj:
98.     fileObj.write(calText)
99.
100. print('Saved to ' + calendarFilename)
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте воссоздать эту программу с нуля, не глядя на исходный код в книге. Программа не должна в точности совпадать с нашей; можете придумать собственную версию! Можете также сами попробовать придумать, как сделать следующее:

- добавить текст внутрь части прямоугольников для праздничных дней;
- добавить текст внутрь части прямоугольников повторяющихся событий;
- вывести мини-календарь с датами без ограничивающих прямоугольников.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Как сделать так, чтобы в календарь выводились сокращенные названия месяцев? Например, `'Jan'` вместо `'January'`.
2. Какое сообщение об ошибке вы получите, если удалите или закомментируете `year = int (response)` в строке 21?
3. Как сделать так, чтобы вверху календаря не отображались дни недели?
4. Как сделать так, чтобы программа не сохраняла календарь в файл?
5. Что будет, если удалить или закомментировать `print(calText)` в строке 93?

9

Морковка в коробке

Это простая несерьезная игра с блефом для двух игроков. У каждого игрока есть коробка; в одной из них лежит морковка, которую хочет заполучить каждый игрок. Первый заглядывает в свою коробку и говорит второму, есть ли там морковка. А второй должен решить: меняться коробками или нет[1].

Из-за ASCII-графики набор данной программы занимает немало времени (хотя процесс можно ускорить путем копирования и вставки ASCII-графики), но этот проект хорошо подходит для начинающих, поскольку прост, содержит минимум циклов и не требует описания функций.

Программа в действии

Результат выполнения `carrotinabox.py` выглядит следующим образом:

```
Carrot in a Box, by Al Sweigart al@inventwithpython.com
--сокращено--
Human player 1, enter your name: Alice
Human player 2, enter your name: Bob
```

[1] Не заглядывая в свою коробку, конечно. — *Примеч. пер.*

```
HERE ARE TWO BOXES:
  __________     __________
 /         /|   /         /|
+---------+ |  +---------+ |
|   RED   | |  |   GOLD  | |
|   BOX   | /  |   BOX   | /
+---------+/   +---------+/
   Alice           Bob

Alice, you have a RED box in front of you.
Bob, you have a GOLD box in front of you.
Press Enter to continue...
--сокращено--
When Bob has closed their eyes, press Enter...
Alice here is the inside of your box:

   ___VV____
  |   VV    |
  |   VV    |
  |___||____|    __________
 /    ||   /|   /         /|
+---------+ |  +---------+ |
|   RED   | |  |   GOLD  | |
|   BOX   | /  |   BOX   | /
+---------+/   +---------+/
 (carrot!)
   Alice           Bob
Press Enter to continue...
--сокращено--
```

Описание работы

В программе предполагается, что второй игрок закрывает глаза и не видит содержимого коробки первого. Чтобы второй игрок случайно не увидел содержимое коробки после этого, необходимо каким-либо образом очищать экран. В строке 83 мы делаем это с помощью оператора `print('\n' * 100)`, который выводит 100 символов новой строки, в результате чего ранее выведенное содержимое уходит вверх, вне поля зрения. Так что второй игрок не увидит случайно то, что предназначено лишь для первого. И хотя второй игрок всегда может прокрутить экран вверх и посмотреть текст, первый сразу это заметит, ведь они сидят рядом.

В строках 114, 130 и 142 размещение вертикальных линий может показаться неправильным, но программа заменяет фигурные скобки на строку символов `'RED '` (с пробелом в конце) или `'GOLD'`. Поскольку эти строки состоят из четырех символов, все вертикальные линии данной коробки выровняются с остальной частью ASCII-изображения.

```
 1. """Морковка в коробке, (c) Эл Свейгарт al@inventwithpython.com
 2. Несерьезная игра с блефом для двух игроков-людей. В ее основе лежит
 3. игра из телепрограммы 8 Out of 10 Cats
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: большая, для начинающих, игра, для двух игроков"""
 6.
 7. import random
 8.
 9. print('''Carrot in a Box, by Al Sweigart al@inventwithpython.com
10.
11. This is a bluffing game for two human players. Each player has a box.
12. One box has a carrot in it. To win, you must have the box with the
13. carrot in it.
14.
15. This is a very simple and silly game.
16.
17. The first player looks into their box (the second player must close
18. their eyes during this). The first player then says "There is a carrot
19. in my box" or "There is not a carrot in my box". The second player then
20. gets to decide if they want to swap boxes or not.
21. ''')
22. input('Press Enter to begin...')
23.
24. p1Name = input('Human player 1, enter your name: ')
25. p2Name = input('Human player 2, enter your name: ')
26. playerNames = p1Name[:11].center(11) + '    ' + p2Name[:11].center(11)
27.
28. print('''HERE ARE TWO BOXES:
29.   __________     __________
30.  /         /|   /         /|
31. +---------+ |  +---------+ |
32. |   RED   | |  |   GOLD  | |
33. |   BOX   | /  |   BOX   | /
34. +---------+/   +---------+/''')
35.
36. print()
37. print(playerNames)
38. print()
39. print(p1Name + ', you have a RED box in front of you.')
40. print(p2Name + ', you have a GOLD box in front of you.')
41. print()
42. print(p1Name + ', you will get to look into your box.')
43. print(p2Name.upper() + ', close your eyes and don\'t look!!!')
44. input('When ' + p2Name + ' has closed their eyes, press Enter...')
45. print()
46.
47. print(p1Name + ' here is the inside of your box:')
48.
49. if random.randint(1, 2) == 1:
50.     carrotInFirstBox = True
51. else:
```

```
    carrotInFirstBox = False

if carrotInFirstBox:
    print('''
    ___VV____
   |   VV    |
   |   VV    |
   |___||____|    __________
  /    ||   /|   /         /|
 +---------+ |  +---------+ |
 |   RED   | |  |   GOLD  | |
 |   BOX   | /  |   BOX   | /
 +---------+/   +---------+/
  (carrot!)''')
    print(playerNames)
else:
    print('''
     _________
   |         |
   |         |
   |_________|    __________
  /         /|   /         /|
 +---------+ |  +---------+ |
 |   RED   | |  |   GOLD  | |
 |   BOX   | /  |   BOX   | /
 +---------+/   +---------+/
 (no carrot!)''')
    print(playerNames)

input('Press Enter to continue...')

print('\n' * 100)  # Очищаем экран, выводя несколько символов новой строки.
print(p1Name + ', tell ' + p2Name + ' to open their eyes.')
input('Press Enter to continue...')

print()
print(p1Name + ', say one of the following sentences to ' + p2Name + '.')
print('  1) There is a carrot in my box.')
print('  2) There is not a carrot in my box.')
print()
input('Then press Enter to continue...')

print()
print(p2Name + ', do you want to swap boxes with ' + p1Name + '? YES/NO')
while True:
    response = input('> ').upper()
    if not (response.startswith('Y') or response.startswith('N')):
        print(p2Name + ', please enter "YES" or "NO".')
    else:
        break

```

```
firstBox = 'RED '  # Обратите внимание на пробел после "D".
secondBox = 'GOLD'

if response.startswith('Y'):
    carrotInFirstBox = not carrotInFirstBox
    firstBox, secondBox = secondBox, firstBox

print('''HERE ARE THE TWO BOXES:
  __________     __________
 /         /|   /         /|
+---------+ |  +---------+ |
|   {}  | |  |   {}  | |
|   BOX   | /  |   BOX   | /
+---------+/   +---------+/'''.format(firstBox, secondBox))
print(playerNames)

input('Press Enter to reveal the winner...')
print()

if carrotInFirstBox:
    print('''
    ___VV____       _________
   |   VV    |     |         |
   |   VV    |     |         |
   |___||____|     |_________|
 /     ||   /|    /         /|
+---------+ |  +---------+ |
|   {}  | |  |   {}  | |
|   BOX   | /  |   BOX   | /
+---------+/   +---------+/'''.format(firstBox, secondBox))

else:
    print('''
    _________       ___VV____
   |         |     |   VV    |
   |         |     |   VV    |
   |_________|     |___||____|
 /         /|    /     ||   /|
+---------+ |  +---------+ |
|   {}  | |  |   {}  | |
|   BOX   | /  |   BOX   | /
+---------+/   +---------+/'''.format(firstBox, secondBox))

print(playerNames)

# Данная модификация возможна благодаря переменной 'carrotInFirstBox'
if carrotInFirstBox:
    print(p1Name + ' is the winner!')
else:
    print(p2Name + ' is the winner!')

print('Thanks for playing!')
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- заменить ASCII-графику для коробок и морковок на что-то более изящное;
- добавить возможность «Хотите сыграть еще раз?», чтобы игроки могли продолжить игру с сохранением счета;
- добавить третьего игрока, перед которым должен блефовать второй.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Обратите внимание на код `p1Name[:11]` и `p2Name[:11]` в строке 26. Введите имя, превышающее десять символов. Какие особенности отображения программой этого имени вы видите?
2. Что будет, если пропустить пробел в конце `firstBox = 'RED '` в строке 103?
3. Что будет, если удалить или закомментировать `print('\n' * 100)` в строке 83?
4. Что будет, если удалить или закомментировать `else:` в строке 100 и `break` в строке 101?

10

Чо-хан

Чо-хан — традиционная игра в кости, распространенная в игорных домах феодальной Японии. Две шестигранные игральные кости выбрасываются из чашки, а игроки должны угадать, окажется сумма четной (чо) или нечетной (хан). Игорный дом берет себе небольшую часть всех выигрышей. Благодаря простоте генерации случайных чисел и арифметики, используемой для определения того, четной или нечетной будет сумма, этот проект особенно хорошо подходит для начинающих. Больше информации о чо-хан можно найти в статье «Википедии»: https://en.wikipedia.org/wiki/Cho-han.

Программа в действии

Результат выполнения `chohan.py` выглядит следующим образом:

```
Cho-Han, by Al Sweigart al@inventwithpython.com

In this traditional Japanese dice game, two dice are rolled in a bamboo
cup by the dealer sitting on the floor. The player must guess if the
dice total to an even (cho) or odd (han) number.

You have 5000 mon. How much do you bet? (or QUIT)
> 400
The dealer swirls the cup and you hear the rattle of dice.
The dealer slams the cup on the floor, still covering the
```

```
dice and asks for your bet.

    CHO (even) or HAN (odd)?
> cho
The dealer lifts the cup to reveal:
   GO - GO
     5 - 5
You won! You take 800 mon.
The house collects a 40 mon fee.
--сокращено--
```

Описание работы

Вызов `random.randint(1, 6)` возвращает случайное целое число между `1` и `6`, а потому идеально подходит для представления броска шестигранной кости. Однако нам нужно также отображать японские слова для чисел от одного до шести. Вместо оператора `if`, за которым следует пять операторов `elif`, мы воспользуемся хранимым в переменной `JAPANESE_NUMBERS` ассоциативным массивом соответствий чисел от `1` до `6` строковым представлениям японских слов. Благодаря этому `JAPANESE_NUMBERS[dice1]` и `JAPANESE_NUMBERS[dice2]` в строке 57 позволяют вывести на экран японские слова для результатов бросков костей в одной строке кода.

```
 1. """Чо-хан, (c) Эл Свейгарт al@inventwithpython.com
 2. Традиционная японская игра в кости типа чет-нечет.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, для начинающих, игра"""
 5.
 6. import random, sys
 7.
 8. JAPANESE_NUMBERS = {1: 'ICHI', 2: 'NI', 3: 'SAN',
 9.                     4: 'SHI', 5: 'GO', 6: 'ROKU'}
10.
11. print('''Cho-Han, by Al Sweigart al@inventwithpython.com
12.
13. In this traditional Japanese dice game, two dice are rolled in a bamboo
14. cup by the dealer sitting on the floor. The player must guess if the
15. dice total to an even (cho) or odd (han) number.
16. ''')
17.
18. purse = 5000
19. while True:  # Основной цикл игры.
20.     # Делайте ставки, господа:
21.     print('You have', purse, 'mon. How much do you bet? (or QUIT)')
22.     while True:
23.         pot = input('> ')
24.         if pot.upper() == 'QUIT':
25.             print('Thanks for playing!')
26.             sys.exit()
```

```
27.         elif not pot.isdecimal():
28.             print('Please enter a number.')
29.         elif int(pot) > purse:
30.             print('You do not have enough to make that bet.')
31.         else:
32.             # Допустимая ставка.
33.             pot = int(pot)  # Преобразуем pot в тип integer.
34.             break  # Выходим из цикла после размещения допустимой ставки.
35.
36.     # Бросаем кости.
37.     dice1 = random.randint(1, 6)
38.     dice2 = random.randint(1, 6)
39.
40.     print('The dealer swirls the cup and you hear the rattle of dice.')
41.     print('The dealer slams the cup on the floor, still covering the')
42.     print('dice and asks for your bet.')
43.     print()
44.     print('    CHO (even) or HAN (odd)?')
45.
46.     # Спрашиваем у игрока, на что он ставит: на чо или на хан:
47.     while True:
48.         bet = input('> ').upper()
49.         if bet != 'CHO' and bet != 'HAN':
50.             print('Please enter either "CHO" or "HAN".')
51.             continue
52.         else:
53.             break
54.
55.     # Открываем результаты броска костей:
56.     print('The dealer lifts the cup to reveal:')
57.     print('  ', JAPANESE_NUMBERS[dice1], '-', JAPANESE_NUMBERS[dice2])
58.     print('    ', dice1, '-', dice2)
59.
60.     # Определяем, выиграл ли игрок:
61.     rollIsEven = (dice1 + dice2) % 2 == 0
62.     if rollIsEven:
63.         correctBet = 'CHO'
64.     else:
65.         correctBet = 'HAN'
66.
67.     playerWon = bet == correctBet
68.
69.     # Отображаем результаты ставок:
70.     if playerWon:
71.         print('You won! You take', pot, 'mon.')
72.         purse = purse + pot  # Прибавляем приз к кошельку.
73.         print('The house collects a', pot // 10, 'mon fee.')
74.         purse = purse - (pot // 10)  # Сбор игрального дома 10%.
75.     else:
76.         purse = purse - pot  # Вычитаем ставку из кошелька.
77.         print('You lost!')
```

```
78.
79.     # Проверяем, не закончились ли у игрока деньги:
80.     if purse == 0:
81.         print('You have run out of money!')
82.         print('Thanks for playing!')
83.         sys.exit()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- реализовать один из вариантов игры, описанных в вышеупомянутой статье «Википедии», в которых несколько игроков делают ставки друг против друга; ввести в игру управляемых компьютером игроков с их собственными кошельками;
- добавить бонусы для определенных бросков, например 7 или «змеиных глаз»[1];
- предоставить игрокам возможность делать ставки на определенные числа и получать в случае успеха бонус.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Как изменить сумму денег, с которой игрок начинает игру?
2. Как программа предотвращает ставки игроков на сумму, превышающую их наличность в кошельке?
3. Как программа определяет, четная или нечетная сумма значений на двух костях?
4. Что будет, если `random.randint(1, 6)` в строке 37 заменить на `random.randint(1, 1)`?
5. Будет ли игральный дом получать 10%-ный сбор, если `pot // 10` в строке 73 (не 74) заменить на `0`?
6. Что будет, если удалить или закомментировать строки 80, 81, 82 и 83?

[1] Когда на обеих костях выпадает 1. — *Примеч. пер.*

11

Генератор заголовков-приманок

Нашему сайту нужно как-то заманивать людей смотреть рекламу! Но придумать интересный оригинальный контент непросто. К счастью, с помощью генератора заголовков-приманок компьютер сможет создать миллионы шокирующих фиктивных заголовков. Конечно, их качество — очень низкое, но читателей, похоже, это не смущает. Данная программа генерирует любое нужное количество заголовков на основе шаблона в стиле игры Mad Libs[1].

В этой программе много текста для шаблонов заголовков, но сам код прост и вполне доступен для начинающих.

Программа в действии

Результат выполнения `clickbait.py` выглядит следующим образом:

```
Clickbait Headline Generator
By Al Sweigart al@inventwithpython.com

Our website needs to trick people into looking at ads!
Enter the number of clickbait headlines to generate:
> 1000
```

[1] Игра, в которой в текст с пропусками нужно подставлять слова из заранее заготовленного списка, в результате чего получается смешной рассказ. — *Примеч. пер.*

```
Big Companies Hate Him! See How This New York Cat Invented a Cheaper Robot
What Telephone Psychics Don't Want You To Know About Avocados
You Won't Believe What This North Carolina Shovel Found in Her Workplace
--сокращено--
14 Reasons Why Parents Are More Interesting Than You Think (Number 1 Will Surprise
You!)
What Robots Don't Want You To Know About Cats
This Florida Telephone Psychic Didn't Think Robots Would Take Her Job. She Was Wrong.
```

Описание работы

Программа включает несколько функций для генерации различных видов заголовков-приманок, получающих случайные слова из `STATES`, `NOUNS`, `PLACES`, `WHEN` и других списков. Далее эти функции вставляют полученные слова в строковое значение шаблона с помощью строкового метода `format()`, а затем возвращают это значение. Все очень похоже на книгу для игры в Mad Libs, но пробелы заполняет компьютер, благодаря чему программа может генерировать тысячи заголовков-приманок в секунду.

```
 1. """Генератор заголовков-приманок, (c) Эл Свейгарт al@inventwithpython.com
 2. Генератор заголовков-приманок для сайта со скучным контентом
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: большая, для начинающих, юмор, слова"""
 5.
 6. import random
 7.
 8. # Задаем константы:
 9. OBJECT_PRONOUNS = ['Her', 'Him', 'Them']
10. POSSESIVE_PRONOUNS = ['Her', 'His', 'Their']
11. PERSONAL_PRONOUNS = ['She', 'He', 'They']
12. STATES = ['California', 'Texas', 'Florida', 'New York', 'Pennsylvania',
13.           'Illinois', 'Ohio', 'Georgia', 'North Carolina', 'Michigan']
14. NOUNS = ['Athlete', 'Clown', 'Shovel', 'Paleo Diet', 'Doctor', 'Parent',
15.          'Cat', 'Dog', 'Chicken', 'Robot', 'Video Game', 'Avocado',
16.          'Plastic Straw','Serial Killer', 'Telephone Psychic']
17. PLACES = ['House', 'Attic', 'Bank Deposit Box', 'School', 'Basement',
18.           'Workplace', 'Donut Shop', 'Apocalypse Bunker']
19. WHEN = ['Soon', 'This Year', 'Later Today', 'RIGHT NOW', 'Next Week']
20.
21.
22. def main():
23.     print('Clickbait Headline Generator')
24.     print('By Al Sweigart al@inventwithpython.com')
25.     print()
26.
27.     print('Our website needs to trick people into looking at ads!')
28.     while True:
29.         print('Enter the number of clickbait headlines to generate:')
```

```
30.         response = input('> ')
31.         if not response.isdecimal():
32.             print('Please enter a number.')
33.         else:
34.             numberOfHeadlines = int(response)
35.             break  # Выходим из цикла, когда будет введено допустимое число.
36.
37.     for i in range(numberOfHeadlines):
38.         clickbaitType = random.randint(1, 8)
39.
40.         if clickbaitType == 1:
41.             headline = generateAreMillennialsKillingHeadline()
42.         elif clickbaitType == 2:
43.             headline = generateWhatYouDontKnowHeadline()
44.         elif clickbaitType == 3:
45.             headline = generateBigCompaniesHateHerHeadline()
46.         elif clickbaitType == 4:
47.             headline = generateYouWontBelieveHeadline()
48.         elif clickbaitType == 5:
49.             headline = generateDontWantYouToKnowHeadline()
50.         elif clickbaitType == 6:
51.             headline = generateGiftIdeaHeadline()
52.         elif clickbaitType == 7:
53.             headline = generateReasonsWhyHeadline()
54.         elif clickbaitType == 8:
55.             headline = generateJobAutomatedHeadline()
56.
57.         print(headline)
58.     print()
59.
60.     website = random.choice(['wobsite', 'blag', 'Facebuuk', 'Googles',
61.                              'Facesbook', 'Tweedie', 'Pastagram'])
62.     when = random.choice(WHEN).lower()
63.     print('Post these to our', website, when, 'or you\'re fired!')
64.
65.
66. # Все эти функции возвращают заголовки различных типов:
67. def generateAreMillennialsKillingHeadline():
68.     noun = random.choice(NOUNS)
69.     return 'Are Millennials Killing the {} Industry?'.format(noun)
70.
71.
72. def generateWhatYouDontKnowHeadline():
73.     noun = random.choice(NOUNS)
74.     pluralNoun = random.choice(NOUNS) + 's'
75.     when = random.choice(WHEN)
76.     return 'Without This {}, {} Could Kill You {}'.format(noun, pluralNoun,
77.     when)
78.
79. def generateBigCompaniesHateHerHeadline():
80.     pronoun = random.choice(OBJECT_PRONOUNS)
```

```
    state = random.choice(STATES)
    noun1 = random.choice(NOUNS)
    noun2 = random.choice(NOUNS)
    return 'Big Companies Hate {}! See How This {} {} Invented a Cheaper {}'.
    format(pronoun, state, noun1, noun2)

def generateYouWontBelieveHeadline():
    state = random.choice(STATES)
    noun = random.choice(NOUNS)
    pronoun = random.choice(POSSESIVE_PRONOUNS)
    place = random.choice(PLACES)
    return 'You Won\'t Believe What This {} {} Found in {} {}'.format(state,
    noun, pronoun, place)

def generateDontWantYouToKnowHeadline():
    pluralNoun1 = random.choice(NOUNS) + 's'
    pluralNoun2 = random.choice(NOUNS) + 's'
    return 'What {} Don\'t Want You To Know About {}'.format(pluralNoun1,
    pluralNoun2)


def generateGiftIdeaHeadline():
    number = random.randint(7, 15)
    noun = random.choice(NOUNS)
    state = random.choice(STATES)
    return '{} Gift Ideas to Give Your {} From {}'.format(number, noun, state)

def generateReasonsWhyHeadline():
    number1 = random.randint(3, 19)
    pluralNoun = random.choice(NOUNS) + 's'
    # number2 should be no larger than number1:
    number2 = random.randint(1, number1)
    return '{} Reasons Why {} Are More Interesting Than You Think (Number {}
    Will Surprise You!)'.format(number1, pluralNoun, number2)


def generateJobAutomatedHeadline():
    state = random.choice(STATES)
    noun = random.choice(NOUNS)

    i = random.randint(0, 2)
    pronoun1 = POSSESIVE_PRONOUNS[i]
    pronoun2 = PERSONAL_PRONOUNS[i]
    if pronoun1 == 'Their':
        return 'This {} {} Didn\'t Think Robots Would Take {} Job. {} Were
        Wrong.'.format(state, noun, pronoun1, pronoun2)
    else:
        return 'This {} {} Didn\'t Think Robots Would Take {} Job. {} Was
        Wrong.'.format(state, noun, pronoun1, pronoun2)


```

```
129. # Если программа не импортируется, а запускается, производим запуск:
130. if __name__ == '__main__':
131.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- добавить еще несколько типов заголовков-приманок;
- добавить новые категории слов, помимо `NOUNS`, `STATES` и т. д.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если удалите или закомментируете `numberOfHeadlines = int(response)` в строке 34?
2. Какое сообщение об ошибке вы получите, если замените `int(response)` на `response` в строке 34?
3. Какое сообщение об ошибке вы получите, если замените строку 19 на `WHEN = []`?

12

Гипотеза Коллатца

Гипотеза Коллатца, называемая также дилеммой 3n + 1, — простейшая из нерешенных задач математики (не волнуйтесь, сама программа достаточно проста для начинающих). Начиная с числа *n*, следующие члены последовательности формируются в соответствии с тремя правилами.

1. Если *n* — четное, то следующий член последовательности равен *n* / 2.
2. Если *n* — нечетное, то следующий член равен *n* * 3 + 1.
3. Если *n* равно 1, то прекращаем. В противном случае повторяем.

Существует (математически не доказанная) гипотеза, что любая такая последовательность завершается 1. Больше информации о гипотезе Коллатца можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Гипотеза_Коллатца.

Программа в действии

Результат выполнения `collatz.py` выглядит следующим образом:

```
Collatz Sequence, or, the 3n + 1 Problem
By Al Sweigart al@inventwithpython.com

The Collatz sequence is a sequence of numbers produced from a starting
number n, following three rules:
--сокращено--
```

```
Enter a starting number (greater than 0) or QUIT:
> 26
26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Collatz Sequence, or, the 3n + 1 Problem
By Al Sweigart al@inventwithpython.com
--сокращено--
Enter a starting number (greater than 0) or QUIT:
> 27
27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 121,
364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350, 175, 526,
263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754,
377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158,
1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051,
6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1300, 650, 325, 976,
488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 80, 40, 20, 10, 5,
16, 8, 4, 2, 1
```

Описание работы

Определить, четное число или нечетное, можно с помощью оператора деления по модулю `%`. Напомню, что этот оператор относится к числу операторов «взятия остатка от деления»: 23 делится на 7 — равно 3 с остатком 2, то есть 23 по модулю 7 равно просто 2. Четные числа делятся на 2 без остатка, а нечетные — с остатком 1. При четном `n` выражение в `if n % 2 == 0`: в строке 33 равно `True`. При нечетном же оно равно `False`.

```
 1. """Гипотеза Коллатца, (c) Эл Свейгарт al@inventwithpython.com
 2. Генерирует члены последовательности Коллатца по заданному начальному числу.
 3. Больше информации — по адресу https://ru.wikipedia.org/wiki/Гипотеза_Коллатца
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, для начинающих, математическая"""
 6.
 7. import sys, time
 8.
 9. print('''Collatz Sequence, or, the 3n + 1 Problem
10. By Al Sweigart al@inventwithpython.com
11.
12. The Collatz sequence is a sequence of numbers produced from a starting
13. number n, following three rules:
14.
15. 1) If n is even, the next number n is n / 2.
16. 2) If n is odd, the next number n is n * 3 + 1.
17. 3) If n is 1, stop. Otherwise, repeat.
18.
19. It is generally thought, but so far not mathematically proven, that
20. every starting number eventually terminates at 1.
21. ''')
```

```
22.
23. print('Enter a starting number (greater than 0) or QUIT:')
24. response = input('> ')
25.
26. if not response.isdecimal() or response == '0':
27.     print('You must enter an integer greater than 0.')
28.     sys.exit()
29.
30. n = int(response)
31. print(n, end='', flush=True)
32. while n != 1:
33.     if n % 2 == 0:  # Если n — четное...
34.         n = n // 2
35.     else:  # В противном случае n — нечетное...
36.         n = 3 * n + 1
37.
38.     print(', ' + str(n), end='', flush=True)
39.     time.sleep(0.1)
40. print()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Сколько членов содержит последовательность Коллатца, начинающаяся с 32?
2. Сколько членов содержит последовательность Коллатца, начинающаяся с 33?
3. Всегда ли последовательности Коллатца, начальные члены которых являются степенями двойки (2, 4, 8, 16, 32, 64, 128 и т. д.), состоят только из четных чисел (не считая конечного числа 1)?
4. Что будет, если ввести `0` в качестве начального числа?

13

Игра «Жизнь» Конвея

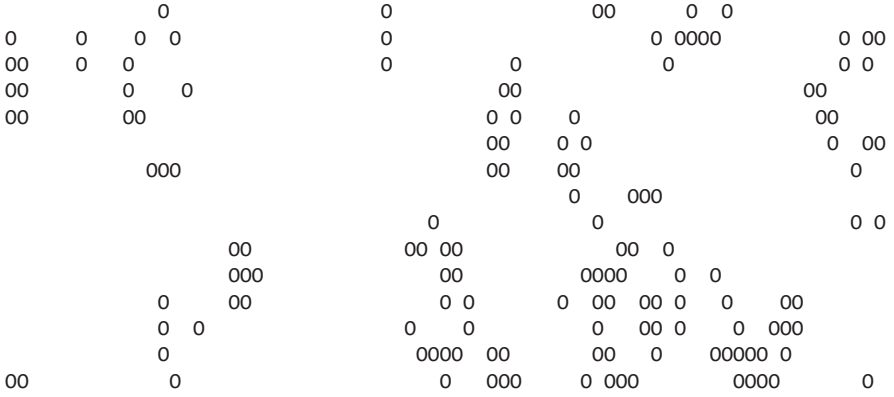
Игра «Жизнь» Конвея — клеточный автомат для имитационного моделирования, позволяющий создавать интересные узоры на основе простых правил. Изобрел ее математик Джон Конвей в 1970 году, а сделала известной рубрика «Математические игры» Мартина Гарднера в журнале *Scientific American*. Программисты и специалисты по вычислительной технике сегодня очень любят эту скорее интересную визуализацию, чем настоящую «игру». Двумерная доска содержит сетку из «клеток», которые все следуют трем простым правилам:

- живые клетки с двумя или тремя соседями остаются живыми на следующем шаге моделирования;
- мертвые клетки с ровно тремя соседями оживают на следующем шаге моделирования;
- все прочие клетки умирают или остаются мертвыми на следующем шаге моделирования.

Состояние клетки на следующем шаге моделирования (живая или мертвая) зависит только от текущего состояния. Клетки не «помнят» старых состояний. Существует множество исследований, посвященных узорам, генерируемым на основе этих простых правил. К сожалению, профессор Конвей умер от осложнений COVID-19 в апреле 2020 года. Больше информации об игре «Жизнь» вы можете найти в статье «Википедии» https://ru.wikipedia.org/wiki/Игра_«Жизнь», а больше информации о Мартине Гарднере — в статье https://ru.wikipedia.org/wiki/Гарднер,_Мартин.

Программа в действии

Результат выполнения `conwaysgameoflife.py` выглядит следующим образом:



Описание работы

Состояние клеток хранится в ассоциативных массивах в переменных `cells` и `nextCells`. Роль ключей обоих массивов играют кортежи `(x, y)`, где `x` и `y` — целочисленные, `'O'` для живых клеток и `' '` для мертвых. В строках с 40-й по 44-ю представление этих ассоциативных массивов выводится на экран. Ассоциативный массив в переменной `cells` отражает текущее состояние клеток, а `nextCells` содержит ассоциативный массив для клеток на следующем шаге моделирования.

```
 1. """Игра "Жизнь" Конвея, (c) Эл Свейгарт al@inventwithpython.com
 2. Клеточный автомат для имитационного моделирования. Нажмите Ctrl+C для останова.
 3. Больше информации — в статье https://ru.wikipedia.org/wiki/Игра_«Жизнь»
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: короткая, графика, имитационное моделирование"""
 6.
 7. import copy, random, sys, time
 8.
 9. # Задаем константы:
10. WIDTH = 79   # Ширина сетки клеток.
11. HEIGHT = 20  # Высота сетки клеток.
12.
13. # (!) Попробуйте заменить значение ALIVE на '#' или другой символ:
14. ALIVE = 'O'  # Символ, соответствующий живой клетке.
15. # (!) Попробуйте заменить значение DEAD на '.' или другой символ:
16. DEAD = ' '   # Символ, соответствующий мертвой клетке.
17.
18. # (!) Попробуйте заменить значение ALIVE на '|', а DEAD на '-'.
19.
20. # cells и nextCells — ассоциативные массивы для хранения состояния игры.
```

```
# Роль их ключей играют кортежи (x, y), а значения представляют собой
# одно из значений ALIVE или DEAD.
nextCells = {}
# Вставляем в nextCells случайные живые и мертвые клетки:
for x in range(WIDTH):  # Проходим в цикле по всем столбцам.
    for y in range(HEIGHT):  # Проходим в цикле по всем строкам.
        # Исходные клетки могут быть живыми или мертвыми с вероятностью 50 %.
        if random.randint(0, 1) == 0:
            nextCells[(x, y)] = ALIVE  # Добавляем живую клетку.
        else:
            nextCells[(x, y)] = DEAD  # Добавляем мертвую клетку.

while True:  # Основной цикл программы.
    # Итерации этого цикла соответствуют шагам моделирования.

    print('\n' * 50)  # Разделяем шаги символами новой строки.
    cells = copy.deepcopy(nextCells)

    # Выводим клетки на экран:
    for y in range(HEIGHT):
        for x in range(WIDTH):
            print(cells[(x, y)], end='')  # Выводим # или пробел.
        print()  # Выводим символ новой строки в конце строки.
    print('Press Ctrl-C to quit.')

    # Вычисляем клетки следующего шага исходя из клеток на текущем шаге:
    for x in range(WIDTH):
        for y in range(HEIGHT):
            # Получаем координаты (x, y) соседних клеток, даже если они
            # выходят за границы:
            left  = (x - 1) % WIDTH
            right = (x + 1) % WIDTH
            above = (y - 1) % HEIGHT
            below = (y + 1) % HEIGHT

            # Подсчитываем количество живых соседей:
            numNeighbors = 0
            if cells[(left, above)] == ALIVE:
                numNeighbors += 1  # Сосед вверху слева жив.
            if cells[(x, above)] == ALIVE:
                numNeighbors += 1  # Сосед вверху жив.
            if cells[(right, above)] == ALIVE:
                numNeighbors += 1  # Сосед вверху справа жив.
            if cells[(left, y)] == ALIVE:
                numNeighbors += 1  # Сосед слева жив.
            if cells[(right, y)] == ALIVE:
                numNeighbors += 1  # Сосед справа жив.
            if cells[(left, below)] == ALIVE:
                numNeighbors += 1  # Сосед внизу слева жив.
            if cells[(x, below)] == ALIVE:
                numNeighbors += 1  # Сосед внизу жив.
            if cells[(right, below)] == ALIVE:
                numNeighbors += 1  # Сосед внизу справа жив.
```

```
74.
75.             # Устанавливаем состояние клеток в соответствии с правилами игры
76.             if cells[(x, y)] == ALIVE and (numNeighbors == 2
77.                 or numNeighbors == 3):
78.                     # Живые клетки с двумя или тремя соседями остаются живыми:
79.                     nextCells[(x, y)] = ALIVE
80.             elif cells[(x, y)] == DEAD and numNeighbors == 3:
81.                 # Мертвые клетки с тремя соседями становятся живыми:
82.                 nextCells[(x, y)] = ALIVE
83.             else:
84.                 # Все остальные клетки умирают или остаются мертвыми:
85.                 nextCells[(x, y)] = DEAD
86.
87.     try:
88.         time.sleep(1)  # Добавляем паузу в 1 секунду, чтобы уменьшить мигание.
89.     except KeyboardInterrupt:
90.         print("Conway's Game of Life")
91.         print('By Al Sweigart al@inventwithpython.com')
92.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- задать вместо 50 % различный процент клеток, начинающих с «живого» состояния;
- добавить возможность чтения начального состояния из текстового файла, чтобы пользователи могли редактировать начальные состояния ячеек вручную.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `WIDTH = 79` в строке 10 заменить на `WIDTH = 7`?
2. Что будет, если удалить или закомментировать `print('\n' * 50)` в строке 36?
3. Что будет, если `random.randint(0, 1)` в строке 28 заменить на `random.randint(0, 10)`?
4. Что будет, если `nextCells[(x, y)] = DEAD` в строке 85 заменить на `nextCells[(x, y)] = ALIVE`?

14

Обратный отсчет

Программа выводит цифровой таймер, отсчитывающий время в обратном направлении. Вместо того чтобы непосредственно визуализировать цифры, мы генерируем их изображения с помощью модуля `sevseg.py` из проекта 64. Чтобы программа обратного отсчета работала, необходимо сначала создать этот файл. Далее вы можете установить таймер обратного отсчета на любое количество секунд, минут и часов. Данная программа аналогична проекту 19.

Программа в действии

Результат выполнения `countdown.py` выглядит следующим образом:

```
 __   __   __   __   __   __
|  | |  |  *  |  | |  |  *   __| |__|
|__| |__|  *  |__| |__|  *  |__   __|
Press Ctrl-C to quit.
```

Описание работы

После выполнения оператора `import sevseg` можно вызвать функцию `sevseg.getSevSegStr()`, возвращающую многострочное строковое значение с семисегментными цифрами. Однако программа обратного отсчета должна отображать

составленное из звездочек двоеточие в качестве разделителя часов, минут и секунд. Для этого необходимо разбить три строки многострочного строкового значения с цифрами на три отдельных строковых значения с помощью метода `splitlines()`.

```
 1. """Обратный отсчет, (c) Эл Свейгарт al@inventwithpython.com
 2. Отображает динамическое изображение таймера обратного отсчета с помощью
 3. семисегментного индикатора. Для останова нажмите Ctrl+C. Больше информации —
 4. в статье https://ru.wikipedia.org/wiki/Семисегментный_индикатор
 5. Требует наличия в том же каталоге файла sevseg.py.
 6. Код размещен на https://nostarch.com/big-book-small-python-projects
 7. Теги: крошечная, графика"""
 8.
 9. import sys, time
10. import sevseg  # Импорт программы sevseg.py.
11.
12. # (!) Можете заменить это значение на любое количество секунд:
13. secondsLeft = 30
14.
15. try:
16.     while True:  # Основной цикл программы.
17.         # Очищаем экран, выводя несколько символов новой строки:
18.         print('\n' * 60)
19.
20.         # Берем часы/минуты/секунды из secondsLeft:
21.         # Например: 7265 равно 2 часам 1 минуте 5 секундам.
22.         # 7265 // 3600 равно 2 часам:
23.         hours = str(secondsLeft // 3600)
24.         # 7265 % 3600 равно 65, и 65 // 60 равно 1 минуте:
25.         minutes = str((secondsLeft % 3600) // 60)
26.         # А 7265 % 60 равно 5 секундам:
27.         seconds = str(secondsLeft % 60)
28.
29.         # Получаем из модуля sevseg строковые значения для цифр:
30.         hDigits = sevseg.getSevSegStr(hours, 2)
31.         hTopRow, hMiddleRow, hBottomRow = hDigits.splitlines()
32.
33.         mDigits = sevseg.getSevSegStr(minutes, 2)
34.         mTopRow, mMiddleRow, mBottomRow = mDigits.splitlines()
35.
36.         sDigits = sevseg.getSevSegStr(seconds, 2)
37.         sTopRow, sMiddleRow, sBottomRow = sDigits.splitlines()
38.
39.         # Отображаем цифры:
40.         print(hTopRow    + '     ' + mTopRow    + '     ' + sTopRow)
41.         print(hMiddleRow + '  *  ' + mMiddleRow + '  *  ' + sMiddleRow)
42.         print(hBottomRow + '  *  ' + mBottomRow + '  *  ' + sBottomRow)
43.
44.         if secondsLeft == 0:
45.             print()
46.             print('    * * * * BOOM * * * *')
```

```
47.             break
48.
49.         print()
50.         print('Press Ctrl-C to quit.')
51.
52.         time.sleep(1)  # Вставляем паузу на 1 секунду.
53.         secondsLeft -= 1
54. except KeyboardInterrupt:
55.     print('Countdown, by Al Sweigart al@inventwithpython.com')
56.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- запросить у пользователя начальное время обратного отсчета;
- запросить у пользователя сообщение для отображения в конце обратного отсчета.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `secondsLeft = 30` в строке 13 заменить на `secondsLeft = 30.5`?
2. Что будет, если `2` в строках 30, 33 и 36 заменить на `1`?
3. Что будет, если `time.sleep(1)` в строке 52 заменить на `time.sleep(0.1)`?
4. Что будет, если `secondsLeft -= 1` в строке 53 заменить на `secondsLeft -= 2`?
5. Что будет, если вы удалите или закомментируете `print('\n' * 60)` в строке 18?
6. Какое сообщение об ошибке вы получите, если удалите или закомментируете `import sevseg` в строке 10?

15

Глубокая пещера

Программа представляет собой динамическое изображение глубокой пещеры, ведущей до самого центра Земли. Хотя и будучи короткой, программа создает интересную бесконечную визуализацию благодаря природе экрана компьютера, основанной на прокрутке, доказывая тем самым, что для создания интересного зрелища не нужно много кода. Данная программа аналогична проекту 58.

Программа в действии

Результат выполнения `deepcave.py` выглядит следующим образом:

```
Deep Cave, by Al Sweigart al@inventwithpython.com
Press Ctrl-C to stop.
####################          ########################################
####################         ########################################
####################          ########################################
####################          ########################################
#####################         ########################################
######################        ########################################
####################          ########################################
####################          ########################################
####################         ########################################
--сокращено--
```

Описание работы

Эта программа активно использует тот факт, что при выводе на экран новых строк старые передвигаются вверх. Благодаря выводу на экран чуть отличающегося зазора программа создает у зрителя впечатление, что он движется вниз.

Количество символов «решетка» слева отслеживается в переменной `leftWidth`. Количество пробелов посередине — в переменной `gapWidth`. Количество символов «решетка» справа вычисляется по формуле `WIDTH - gapWidth - leftWidth`, что гарантирует одинаковую ширину всех строк.

```
"""Глубокая пещера, (c) Эл Свейгарт al@inventwithpython.com
Динамическое изображение глубокой пещеры, ведущей до самого центра Земли.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: крошечная, для начинающих, прокрутка, графика"""


import random, sys, time

# Задаем константы:
WIDTH = 70  # (!) Попробуйте заменить на 10 или 30.
PAUSE_AMOUNT = 0.05  # (!) Попробуйте заменить на 0 или 1.0.

print('Deep Cave, by Al Sweigart al@inventwithpython.com')
print('Press Ctrl-C to stop.')
time.sleep(2)

leftWidth = 20
gapWidth = 10

while True:
    # Отображает фрагмент туннеля:
    rightWidth = WIDTH - gapWidth - leftWidth
    print(('#' * leftWidth) + (' ' * gapWidth) + ('#' * rightWidth))

    # Проверяем, не нажато ли Ctrl+C, во время короткой паузы:
    try:
        time.sleep(PAUSE_AMOUNT)
    except KeyboardInterrupt:
        sys.exit() # Если нажато сочетание клавиш Ctrl+C — завершаем программу.

    # Подбираем ширину левой части:
    diceRoll = random.randint(1, 6)
    if diceRoll == 1 and leftWidth > 1:
        leftWidth = leftWidth - 1  # Уменьшаем ширину левой части.
    elif diceRoll == 2 and leftWidth + gapWidth < WIDTH - 1:
        leftWidth = leftWidth + 1  # Увеличиваем ширину левой части.
    else:
        pass  # Ничего не делаем; ширина левой части не меняется.

```

```
40.     # Подбираем ширину зазора:
41.     # (!) Попробуйте раскомментировать весь следующий код:
42.     #diceRoll = random.randint(1, 6)
43.     #if diceRoll == 1 and gapWidth > 1:
44.     #    gapWidth = gapWidth - 1  # Уменьшаем.
45.     #elif diceRoll == 2 and leftWidth + gapWidth < WIDTH - 1:
46.     #    gapWidth = gapWidth + 1  # Увеличиваем ширину зазора.
47.     #else:
48.     #    pass  # Ничего не делаем; ширина зазора не меняется.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных `(!)`.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `(' ' * gapWidth)` в строке 23 заменить на `('.' * gapWidth)`?
2. Что будет, если `random.randint(1, 6)` в строке 32 заменить на `random.randint(1, 1)`?
3. Что будет, если `random.randint(1, 6)` в строке 32 заменить на `random.randint(2, 2)`?
4. Какое сообщение об ошибке вы получите, если удалите или закомментируете `leftWidth = 20` в строке 17?
5. Что будет, если `WIDTH = 70` в строке 10 заменить на `WIDTH = -70`?
6. Какое сообщение об ошибке вы получите, если `PAUSE_AMOUNT = 0.05` в строке 11 замените на `PAUSE_AMOUNT = -0.05`?

16

Ромбы

Данная программа иллюстрирует небольшой алгоритм для рисования ромбов различного размера с помощью ASCII-графики. Программа включает функции для рисования как контуров, так и заполненного ромба указанного размера. Эти функции — хорошее упражнение для начинающих; попробуйте понять закономерности рисунков ромбов по мере увеличения их размера.

Программа в действии

Результат выполнения `diamonds.py` выглядит следующим образом:

```
Diamonds, by Al Sweigart al@inventwithpython.com

/\
\/

/\
\/

 /\
/  \
\  /
 \/

 /\
//\\
\\//
 \/

  /\
 /  \
/    \
\    /
 \  /
  \/

  /\
 //\\
///\\\
\\\///
 \\//
  \/
--сокращено--
```

Описание работы

Для создания подобной программы самостоятельно не помешает сначала «нарисовать» ромбы различного размера в редакторе и выяснить закономерности их построения. Такая методика поможет вам понять, что каждая строка контура ромба состоит из четырех частей: нескольких ведущих пробелов, внешней прямой косой черты, нескольких внутренних пробелов и внешней обратной косой черты. Заполненные ромбы вместо внутренних пробелов включают несколько внутренних прямых и обратных косых черт. Только разобравшись с этими закономерностями, я смог написать `diamonds.py`.

```
"""Ромбы, (c) Эл Свейгарт al@inventwithpython.com
Рисует ромбы различного размера.
Код размещен на https://nostarch.com/big-book-small-python-projects
                           /\       /\
                          /  \     //\\
            /\     /\    /    \   ///\\\
           /  \   //\\  /      \ ////\\\\
 /\   /\  /    \ ///\\\ \      / \\\\////
/  \ //\\ \    / \\\///  \    /   \\\///
\  / \\//  \  /   \\//    \  /     \\//
 \/   \/    \/     \/      \/       \/
Теги: крошечная, для начинающих, графика"""

def main():
    print('Diamonds, by Al Sweigart al@inventwithpython.com')

    # Отображает ромбы размера с 0 по 6:
    for diamondSize in range(0, 6):
        displayOutlineDiamond(diamondSize)
        print()  # Выводит символ новой строки.
        displayFilledDiamond(diamondSize)
        print()  # Выводит символ новой строки.


def displayOutlineDiamond(size):
    # Отображает верхнюю половину ромба:
    for i in range(size):
        print(' ' * (size - i - 1), end='')  # Пробелы слева.
        print('/', end='')  # Левая сторона ромба.
        print(' ' * (i * 2), end='')  # Внутренность ромба.
        print('\\')  # Правая сторона ромба.

    # Отображает нижнюю половину ромба:
    for i in range(size):
        print(' ' * i, end='')  # Пробелы слева.
        print('\\', end='')  # Левая сторона ромба.
        print(' ' * ((size - i - 1) * 2), end='')  # Внутренность ромба.
        print('/')  # Правая сторона ромба.


def displayFilledDiamond(size):
    # Отображает верхнюю половину ромба:
    for i in range(size):
        print(' ' * (size - i - 1), end='')  # Пробелы слева.
        print('/' * (i + 1), end='')  # Левая сторона ромба.
        print('\\' * (i + 1))  # Правая сторона ромба.

    # Отображает нижнюю половину ромба:
    for i in range(size):
        print(' ' * i, end='')  # Пробелы слева.
        print('\\' * (size - i), end='')  # Левая сторона ромба.
```

```
52.         print('/' * (size - i))  # Правая сторона ромба.
53.
54.
55. # Если программа не импортируется, а запускается, производим запуск:
56. if __name__ == '__main__':
57.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- нарисовать другие фигуры: треугольники, прямоугольники и фигуры в виде различных ромбов;
- выводить нарисованные фигуры в текстовый файл, а не на экран.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `print('\\')` в строке 31 заменить на `print('@')`?
2. Что будет, если `print(' ' * (i * 2), end='')` в строке 30 заменить на `print('@' * (i * 2), end='')`?
3. Что будет, если `range(0, 6)` в строке 18 заменить на `range(0, 30)`?
4. Что будет, если удалить или закомментировать `for i in range(size):` в строке 34 или в строке 49?

17

Арифметика с игральными костями

Эта программа — математическая головоломка — выбрасывает от двух до шести игральных костей, сумму очков на которых вы должны вычислить так быстро, как только можете. Но данная программа — не просто автоматизированные обучающие карточки; выпавшие верхние (лицевые) стороны костей отображаются в случайных местах на экране. Подобная ASCII-графика позволяет превратить упражнения в арифметике в развлечение.

Программа в действии

Результат выполнения `dicemath.py` выглядит следующим образом:

```
Dice Math, by Al Sweigart al@inventwithpython.com

Add up the sides of all the dice displayed on the screen. You have
30 seconds to answer as many as possible. You get 4 points for each
correct answer and lose 1 point for each incorrect answer.

Press Enter to begin...
                                  +-------+
                                  | O   O |
                                  |   O   |
                                  | O   O |
                                  +-------+


                              +-------+
          +-------+           | O   O |  +-------+
          |     O |           |       |  | O     |
          |       |           | O   O |  |       |
          | O     |           +-------+  |     O |
          +-------+                      +-------+
Enter the sum: 13
--сокращено--
```

Описание работы

Отображаемые на экране кости представлены ассоциативным массивом, хранящимся в переменной `canvas`. В Python кортежи аналогичны спискам, но их содержимое нельзя менять. Ключи этого ассоциативного массива представляют собой кортежи `(x, y)`, соответствующие позиции левого верхнего угла кости, а значения — один из «кортежей костей» в `ALL_DICE`. Как видно из строк с 28-й по 80-ю, каждый кортеж кости содержит список строковых значений, графически отражающих одну из возможных лицевых сторон кости, и целого числа, отражающего количество точек на этой лицевой стороне. Далее программа на основе данной информации отображает кость и вычисляет общую сумму очков.

В строках с 174-й по 177-ю данные из ассоциативного массива `canvas` визуализируются на экране, подобно тому как в проекте 13 мы визуализировали клетки на экране.

```
1. """Арифметика с игральными костями, (c) Эл Свейгарт al@inventwithpython.com
2. Игра с обучающими карточками на сложение, в которой нужно
   ➥ суммировать все очки на выброшенных игральных костях
3. Код размещен на https://nostarch.com/big-book-small-python-projects
```

```
 4. Теги: большая, графика, игра, математическая"""
 5.
 6. import random, time
 7.
 8. # Задаем константы:
 9. DICE_WIDTH = 9
10. DICE_HEIGHT = 5
11. CANVAS_WIDTH = 79
12. CANVAS_HEIGHT = 24 - 3  # -3, чтобы было куда ввести сумму внизу
13.
14. # Длительность в секундах:
15. QUIZ_DURATION = 30  # (!) Попробуйте заменить это значение на 10 или 60.
16. MIN_DICE = 2  # (!) Попробуйте заменить это значение на 1 или 5.
17. MAX_DICE = 6  # (!) Попробуйте заменить это значение на 14.
18.
19. # (!) Попробуйте заменить эти значения на различные другие:
20. REWARD = 4  # (!) Очки, полученные за правильные ответы.
21. PENAltY = 1  # (!) Очки, отнятые за неправильные ответы.
22. # (!) Попробуйте задать отрицательное значение PENAltY, чтобы давать
23. # очки за неправильные ответы!
24.
25. # Если все кости не помещаются на экране, программа зависает:
26. assert MAX_DICE <= 14
27.
28. D1 = (['+-------+',
29.        '|       |',
30.        '|   O   |',
31.        '|       |',
32.        '+-------+'], 1)
33.
34. D2a = (['+-------+',
35.         '| O     |',
36.         '|       |',
37.         '|     O |',
38.         '+-------+'], 2)
39.
40. D2b = (['+-------+',
41.         '|     O |',
42.         '|       |',
43.         '| O     |',
44.         '+-------+'], 2)
45.
46. D3a = (['+-------+',
47.         '| O     |',
48.         '|   O   |',
49.         '|     O |',
50.         '+-------+'], 3)
51.
52. D3b = (['+-------+',
53.         '|     O |',
54.         '|   O   |',
```

```
 55.         '| O     |',
 56.         '+-------+'], 3)
 57.
 58. D4 = (['+-------+',
 59.        '| O   O |',
 60.        '|       |',
 61.        '| O   O |',
 62.        '+-------+'], 4)
 63.
 64. D5 = (['+-------+',
 65.        '| O   O |',
 66.        '|   O   |',
 67.        '| O   O |',
 68.        '+-------+'], 5)
 69.
 70. D6a = (['+-------+',
 71.         '| O   O |',
 72.         '| O   O |',
 73.         '| O   O |',
 74.         '+-------+'], 6)
 75.
 76. D6b = (['+-------+',
 77.         '| O O O |',
 78.         '|       |',
 79.         '| O O O |',
 80.         '+-------+'], 6)
 81.
 82. ALL_DICE = [D1, D2a, D2b, D3a, D3b, D4, D5, D6a, D6b]
 83.
 84. print('''Dice Math, by Al Sweigart al@inventwithpython.com
 85.
 86. Add up the sides of all the dice displayed on the screen. You have
 87. {} seconds to answer as many as possible. You get {} points for each
 88. correct answer and lose {} point for each incorrect answer.
 89. '''.format(QUIZ_DURATION, REWARD, PENAltY))
 90. input('Press Enter to begin...')
 91.
 92. # Отслеживаем количество правильных и неправильных ответов:
 93. correctAnswers = 0
 94. incorrectAnswers = 0
 95. startTime = time.time()
 96. while time.time() < startTime + QUIZ_DURATION:  # Основной цикл игры.
 97.     # Выбираем кость для отображения:
 98.     sumAnswer = 0
 99.     diceFaces = []
100.     for i in range(random.randint(MIN_DICE, MAX_DICE)):
101.         die = random.choice(ALL_DICE)
102.         # die[0] содержит список лицевых сторон костей в виде строк:
103.         diceFaces.append(die[0])
```

```
        # die[1] содержит количество точек на лицевой стороне в виде чисел:
        sumAnswer += die[1]

    # Содержит кортежи (x, y) с местоположением верхнего левого угла кости.
    topLeftDiceCorners = []

    # Определяем, где должна быть размещена кость:
    for i in range(len(diceFaces)):
        while True:
            # Находим случайное место на холсте для размещения кости:
            left = random.randint(0, CANVAS_WIDTH  - 1 - DICE_WIDTH)
            top  = random.randint(0, CANVAS_HEIGHT - 1 - DICE_HEIGHT)

            # Получаем координаты x, y всех четырех углов:
            #      left
            #      v
            #top > +-------+ ^
            #      | O     | |
            #      |   O   | DICE_HEIGHT (5)
            #      |     O | |
            #      +-------+ v
            #      <-------->
            #      DICE_WIDTH (9)
            topLeftX = left
            topLeftY = top
            topRightX = left + DICE_WIDTH
            topRightY = top
            bottomLeftX = left
            bottomLeftY = top + DICE_HEIGHT
            bottomRightX = left + DICE_WIDTH
            bottomRightY = top + DICE_HEIGHT

            # Проверяем, не пересекается ли эта игральная кость с предыдущей.
            overlaps = False
            for prevDieLeft, prevDieTop in topLeftDiceCorners:
                prevDieRight = prevDieLeft + DICE_WIDTH
                prevDieBottom = prevDieTop + DICE_HEIGHT
                # Проверяем все углы этой кости, не входят ли они
                # в область, занимаемую предыдущей костью:
                for cornerX, cornerY in ((topLeftX, topLeftY),
                                         (topRightX, topRightY),
                                         (bottomLeftX, bottomLeftY),
                                         (bottomRightX, bottomRightY)):
                    if (prevDieLeft <= cornerX < prevDieRight
                        and prevDieTop <= cornerY < prevDieBottom):
                            overlaps = True
            if not overlaps:
                # Если не пересекается, можем ее тут разместить:
                topLeftDiceCorners.append((left, top))
```

```
153.                 break
154.
155.     # Отрисовываем кость на холсте:
156.
157.     # Ключи представляют собой кортежи (x, y) целочисленных значений,
158.     # значения — символы на соответствующем месте холста:
159.     canvas = {}
160.     # Проходим в цикле по всем костям:
161.     for i, (dieLeft, dieTop) in enumerate(topLeftDiceCorners):
162.         # Проходим в цикле по всем символам лицевой стороны кости:
163.         dieFace = diceFaces[i]
164.         for dx in range(DICE_WIDTH):
165.             for dy in range(DICE_HEIGHT):
166.                 # Копируем символ в соответствующее место холста:
167.                 canvasX = dieLeft + dx
168.                 canvasY = dieTop + dy
169.                 # Обратите внимание, что в dieFace, списке строковых
170.                 # значений, x и y поменяны местами:
171.                 canvas[(canvasX, canvasY)] = dieFace[dy][dx]
172.
173.     # Выводим холст на экран:
174.     for cy in range(CANVAS_HEIGHT):
175.         for cx in range(CANVAS_WIDTH):
176.             print(canvas.get((cx, cy), ' '), end='')
177.         print()  # Выводим символ новой строки.
178.
179.     # Даем игроку возможность ввести свой ответ:
180.     response = input('Enter the sum: ').strip()
181.     if response.isdecimal() and int(response) == sumAnswer:
182.         correctAnswers += 1
183.     else:
184.         print('Incorrect, the answer is', sumAnswer)
185.         time.sleep(2)
186.         incorrectAnswers += 1
187.
188. # Отображаем итоговый счет:
189. score = (correctAnswers * REWARD) - (incorrectAnswers * PENAltY)
190. print('Correct:  ', correctAnswers)
191. print('Incorrect:', incorrectAnswers)
192. print('Score:    ', score)
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- переделать ASCII-графику для лицевых сторон костей;
- добавить лицевые стороны костей с семью, восемью и девятью точками.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если строку 82 заменить на `ALL_DICE = [D1]`?
2. Что будет, если `get((cx, cy), ' ')` в строке 176 заменить на `get((cx, cy), '.')`?
3. Что будет, если `correctAnswers += 1` в строке 182 заменить на `correctAnswers += 0`?
4. Какое сообщение об ошибке вы получите, если удалите или закомментируете `correctAnswers = 0` в строке 93?

18

Выбрасыватель игральных костей

В Dungeons & Dragons и других настольных ролевых играх используются специальные игральные кости с 4, 8, 10, 12 или даже 20 гранями. В этих играх есть также специальные обозначения для бросков различных костей. Например, `3d6` означает выбрасывание трех шестигранных костей, а `1d10+2` — выбрасывание одной десятигранной кости с добавлением к броску бонуса в два очка. Представленная ниже программа моделирует подобные броски костей на случай, если вы забыли захватить с собой свои. Она также моделирует выбрасывание не существующих физически костей, например 38-гранных.

Программа в действии

Результат выполнения `diceroller.py` выглядит следующим образом:

```
Dice Roller, by Al Sweigart al@inventwithpython.com
--сокращено--
> 3d6
7 (3, 2, 2)
> 1d10+2
9 (7, +2)
> 2d38-1
32 (20, 13, -1)
> 100d6
```

```
364 (3, 3, 2, 4, 2, 1, 4, 2, 4, 6, 4, 5, 4, 3, 3, 3, 2, 5, 1, 5, 6, 6, 6, 4,
5, 5, 1, 5, 2, 2, 2, 5, 1, 1, 2, 1, 4, 5, 6, 2, 4, 3, 4, 3, 5, 2, 2, 1, 1, 5,
1, 3, 6, 6, 6, 6, 5, 2, 6, 5, 4, 4, 5, 1, 6, 6, 6, 4, 2, 6, 2, 6, 2, 2, 4, 3,
6, 4, 6, 4, 2, 4, 3, 3, 1, 6, 3, 3, 4, 4, 5, 5, 5, 6, 2, 3, 6, 1, 1, 1)
--сокращено--
```

Описание работы

Бóльшая часть кода этой программы посвящена проверке форматирования вводимых пользователем данных. Сам же случайный бросок игральных костей представляет собой всего лишь вызов `random.randint()`. У этой функции отсутствует систематическая ошибка: все целые числа из указанного диапазона возвращаются с равной вероятностью, так что `random.randint()` идеально подходит для моделирования выбрасывания костей.

```
 1. """Выбрасыватель игральных костей, (c) Эл Свейгарт al@inventwithpython.com
 2. Моделирует выбрасывание костей, в нотации Dungeons & Dragons
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, имитационное моделирование"""
 5.
 6. import random, sys
 7.
 8. print('''Dice Roller, by Al Sweigart al@inventwithpython.com
 9.
10. Enter what kind and how many dice to roll. The format is the number of
11. dice, followed by "d", followed by the number of sides the dice have.
12. You can also add a plus or minus adjustment.
13.
14. Examples:
15.   3d6 rolls three 6-sided dice
16.   1d10+2 rolls one 10-sided die, and adds 2
17.   2d38-1 rolls two 38-sided die, and subtracts 1
18.   QUIT quits the program
19. ''')
20.
21. while True:  # Основной цикл программы:
22.     try:
23.         diceStr = input('> ')  # Приглашение ввести описание игральных костей.
24.         if diceStr.upper() == 'QUIT':
25.             print('Thanks for playing!')
26.             sys.exit()
27.
28.         # Очищаем строку описания игральных костей:
29.         diceStr = diceStr.lower().replace(' ', '')
30.
31.         # Ищем "d" в строке описания игральных костей:
32.         dIndex = diceStr.find('d')
33.         if dIndex == -1:
```

```
34.             raise Exception('Missing the "d" character.')
35.
36.         # Выясняем количество костей. ("3" в "3d6+1"):
37.         numberOfDice = diceStr[:dIndex]
38.         if not numberOfDice.isdecimal():
39.             raise Exception('Missing the number of dice.')
40.         numberOfDice = int(numberOfDice)
41.
42.         # Выясняем, присутствует ли модификатор в виде знака плюс или минус:
43.         modIndex = diceStr.find('+')
44.         if modIndex == -1:
45.             modIndex = diceStr.find('-')
46.
47.         # Выясняем количество граней ("6" в "3d6+1"):
48.         if modIndex == -1:
49.             numberOfSides = diceStr[dIndex + 1 :]
50.         else:
51.             numberOfSides = diceStr[dIndex + 1 : modIndex]
52.         if not numberOfSides.isdecimal():
53.             raise Exception('Missing the number of sides.')
54.         numberOfSides = int(numberOfSides)
55.
56.         # Выясняем числовое значение модификатора (The "1" in "3d6+1"):
57.         if modIndex == -1:
58.             modAmount = 0
59.         else:
60.             modAmount = int(diceStr[modIndex + 1 :])
61.             if diceStr[modIndex] == '-':
62.                 # Меняем числовое значение модификатора на отрицательное:
63.                 modAmount = -modAmount
64.
65.         # Моделируем бросок игральных костей:
66.         rolls = []
67.         for i in range(numberOfDice):
68.             rollResult = random.randint(1, numberOfSides)
69.             rolls.append(rollResult)
70.
71.         # Отображаем итоговую сумму очков:
72.         print('Total:', sum(rolls) + modAmount, '(Each die:', end='')
73.
74.         # Отображаем отдельные броски:
75.         for i, roll in enumerate(rolls):
76.             rolls[i] = str(roll)
77.         print(', '.join(rolls), end='')
78.
79.         # Отображаем числовое значение модификатора:
80.         if modAmount != 0:
81.             modSign = diceStr[modIndex]
82.             print(', {}{}'.format(modSign, abs(modAmount)), end='')
83.         print(')')
84.
```

```
    except Exception as exc:
        # Перехватываем все исключения и отображаем сообщение пользователю:
        print('Invalid input. Enter something like "3d6" or "1d10+2".')
        print('Input was invalid because: ' + str(exc))
        continue  # Возвращаемся к приглашению ввести описание игральных костей.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- добавить модификатор знака умножения, в дополнение к модификаторам знаков сложения и вычитания;
- добавить возможность автоматически исключать бросок костей с минимальной суммой очков.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `rolls.append(rollResult)` в строке 69?
2. Что будет, если `rolls.append(rollResult)` в строке 69 заменить на `rolls.append(-rollResult)`?
3. Что будет, если удалить или закомментировать `print(', '.join(rolls), end='')` в строке 77?
4. Что будет, если вместо строки описания броска костей не ввести ничего?

19

Цифровые часы

Эта программа выводит цифровые часы, показывающие текущее время. Вместо того чтобы непосредственно визуализировать цифры, мы генерируем их изображения с помощью модуля `sevseg.py` из проекта 64. Данная программа аналогична проекту 14.

Программа в действии

Результат выполнения `digitalclock.py` выглядит следующим образом:

```
 _   _       _   _       _   _
| | |_|  *  _|  _|  *  |_| |_
|_|  _|  *  _|  _|  *   _| |_|
Press Ctrl-C to quit.
```

Описание работы

Наша программа для цифровых часов напоминает программу из проекта 14. Они обе не только импортируют модуль `sevseg.py`, но и разбивают возвращаемые функцией `sevseg.getSevSegStr()` многострочные строковые значения с помощью метода `splitlines()`. Это позволяет отображать составленное из звездочек двоеточие в качестве разделителя часов, минут и секунд на часах. Сравните данный код с кодом проекта 14 и посмотрите, в чем они схожи, а чем различаются.

```
"""Цифровые часы, (с) Эл Свейгарт al@inventwithpython.com
Отображает показывающие текущее время цифровые часы с семисегментным
индикатором. Нажмите Ctrl+C для останова.
Код размещен на https://nostarch.com/big-book-small-python-projects
Требует наличия в том же каталоге файла sevseg.py
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: крошечная, графика"""

import sys, time
import sevseg  # Импорт программы sevseg.py.

try:
    while True:  # Основной цикл программы.
        # Очищаем экран, выводя несколько символов новой строки:
        print('\n' * 60)

        # Получаем текущее время из системных часов компьютера:
        currentTime = time.localtime()
        # % 12, поскольку мы используем 12-, а не 24-часовые часы:
        hours = str(currentTime.tm_hour % 12)
        if hours == '0':
            hours = '12'  # 12-часовые часы показывают 12:00, а не 00:00.
        minutes = str(currentTime.tm_min)
        seconds = str(currentTime.tm_sec)

        # Получаем из модуля sevseg строковые значения для цифр:
        hDigits = sevseg.getSevSegStr(hours, 2)
        hTopRow, hMiddleRow, hBottomRow = hDigits.splitlines()

        mDigits = sevseg.getSevSegStr(minutes, 2)
        mTopRow, mMiddleRow, mBottomRow = mDigits.splitlines()

        sDigits = sevseg.getSevSegStr(seconds, 2)
        sTopRow, sMiddleRow, sBottomRow = sDigits.splitlines()

        # Отображаем цифры:
        print(hTopRow    + '     ' + mTopRow    + '     ' + sTopRow)
        print(hMiddleRow + '  *  ' + mMiddleRow + '  *  ' + sMiddleRow)
        print(hBottomRow + '  *  ' + mBottomRow + '  *  ' + sBottomRow)
        print()
        print('Press Ctrl-C to quit.')

        # Продолжаем выполнение цикла до перехода на новую секунду:
        while True:
            time.sleep(0.01)
            if time.localtime().tm_sec != currentTime.tm_sec:
                break
except KeyboardInterrupt:
    print('Digital Clock, by Al Sweigart al@inventwithpython.com')
    sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `time.sleep(0.01)` в строке 45 заменить на `time.sleep(2)`?
2. Что будет, если `2` в строках 27, 30 и 33 заменить на `1`?
3. Что будет, если удалить или закомментировать `print('\n' * 60)` в строке 15?
4. Какое сообщение об ошибке вы получите, если удалите или закомментируете `import sevseg` в строке 10?

20

Цифровой поток

Эта программа имитирует визуализацию «цифрового потока» из научно-фантастического фильма «Матрица». Случайные цепочки из двоичного «дождя» поднимаются от низа экрана, формируя красивую хакерскую визуализацию. (К сожалению, из-за того, что текст движется по мере прокрутки экрана вниз, невозможно создать падающие вниз потоки без модулей наподобие `bext`.)

Программа в действии

Результат выполнения `digitalstream.py` выглядит следующим образом:

```
Digital Stream Screensaver, by Al Sweigart al@inventwithpython.com
Press Ctrl-C to quit.
                    0                     0
                    0                     0
   1            0    0    1              1 0                          1
   0            0    0    1         0    0 0         0                 0
   0            1    0    0         0    1 0 0       1              0    1
   0            1    0    0         1    011 1       1              0    1 0
   0            1    0    0         0    000 11      0              0  1 1 0
   1     1      0 1  0    1         1    110 10  1   0              1  0 1 0
         1    101 0       0         1    000 11  1   1              11 1 1 1
         0    100 1       0              11  00  0   1              01     0
      1  1    001 1       1              0   1  10   0              10     0
      0  0    010 0       1                  1  11  11              0      0
--сокращено--
```

Описание работы

Как и в проекте 15, для создания динамического изображения в этой программе используется прокрутка на основе вызовов `print()`. Каждому столбцу соответствует целое число в списке `columns`: `columns[0]` — число, соответствующее крайнему слева столбцу, `columns[1]` — число, соответствующее столбцу непосредственно справа от него, и т. д. Начальные значения этих чисел программа устанавливает в `0`, то есть выводит `' '` (пустое строковое значение с пробелом) вместо потока в данном столбце. А затем меняет каждое из этих чисел случайным образом на значение в диапазоне от `MIN_STREAM_LENGTH` до `MAX_STREAM_LENGTH`. Далее с каждой выведенной строкой это число уменьшается на `1`. И до тех пор, пока соответствующее столбцу число превышает `0`, программа выводит в данном столбце случайным образом `1` или `0`. В результате на экране создается эффект цифрового потока.

```
"""Цифровой поток, (c) Эл Свейгарт al@inventwithpython.com
Экранная заставка в стиле визуальных эффектов фильма "Матрица".
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: крошечная, графика, для начинающих, прокрутка"""

import random, shutil, sys, time

# Задаем константы:
MIN_STREAM_LENGTH = 6  # (!) Попробуйте заменить это значение на 1 или 50.
MAX_STREAM_LENGTH = 14  # (!) Попробуйте заменить это значение на 100.
PAUSE = 0.1  # (!) Попробуйте заменить это значение на 0.0 или 2.0.
STREAM_CHARS = ['0', '1']  # (!) Попробуйте заменить их на другие символы.

# Плотность может варьироваться от 0.0 до 1.0:
DENSITY = 0.02  # (!) Попробуйте заменить это значение на 0.10 или 0.30.

# Получаем размер окна терминала:
WIDTH = shutil.get_terminal_size()[0]
# В Windows нельзя вывести что-либо в последнем столбце без добавления
# автоматически символа новой строки, поэтому уменьшаем ширину на 1:
WIDTH -= 1

print('Digital Stream, by Al Sweigart al@inventwithpython.com')
print('Press Ctrl-C to quit.')
time.sleep(2)

try:
    # Если для столбца счетчик равен 0, поток не отображается.
    # В противном случае он показывает, сколько раз должны отображаться
    # в этом столбце 1 или 0.
    columns = [0] * WIDTH
    while True:
        # Задаем счетчики для каждого из столбцов:
        for i in range(WIDTH):
            if columns[i] == 0:
```

```
36.                 if random.random() <= DENSITY:
37.                     # Перезапускаем поток для этого столбца.
38.                     columns[i] = random.randint(MIN_STREAM_LENGTH,
39.                                                 MAX_STREAM_LENGTH)
40.
41.             # Выводим пробел или символ 1/0.
42.             if columns[i] > 0:
43.                 print(random.choice(STREAM_CHARS), end='')
44.                 columns[i] -= 1
45.             else:
46.                 print(' ', end='')
47.         print()  # Выводим символ новой строки в конце строки столбцов.
48.         sys.stdout.flush()  # Обеспечиваем появление текста на экране.
49.         time.sleep(PAUSE)
50. except KeyboardInterrupt:
51.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- добавить отображение и других символов, кроме единиц и нулей;
- добавить другие фигуры, помимо строк, включая прямоугольники, треугольники и ромбы.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `print(' ', end='')` в строке 46 заменить на `print('.', end='')`?
2. Какое сообщение об ошибке вы получите, если `PAUSE = 0.1` в строке 11 замените на `PAUSE = -0.1`?
3. Что будет, если `columns[i] > 0` в строке 42 заменить на `columns[i] < 0`?
4. Что будет, если `columns[i] > 0` в строке 42 заменить на `columns[i] <= 0`?
5. Что будет, если `columns[i] -= 1` в строке 44 заменить на `columns[i] += 1`?

21

Визуализация ДНК

Дезоксирибонуклеиновая кислота — крошечная молекула, присутствующая в каждой клетке нашего тела и содержащая черновой план его развития. Она представляет собой *двойную спираль* (своего рода винтовая лестница) пар молекул нуклеотидов: гуанина, цитозина, аденина и тимина. Для их обозначения используются буквы G, C, A и T. ДНК — очень длинная молекула; она микроскопическая, но если вытянуть ее в длину, то цепочка пар оснований достигает двух метров! Программа ниже представляет собой простое динамическое изображение ДНК.

Программа в действии

Результат выполнения `dna.py` выглядит следующим образом:

```
DNA Animation, by Al Sweigart al@inventwithpython.com
Press Ctrl-C to quit...
        #G-C#
       #C---G#
      #T-----A#
     #T------A#
    #A------T#
    #G-----C#
     #G---C#
     #C-G#
      ##
     #T-A#
     #C---G#
    #G-----C#
    #G------C#
     #T------A#
      #A-----T#
       #C---G#
        #G-C#
         ##
        #T-A#
       #T---A#
      #A-----T#
--сокращено--
```

Описание работы

Подобно программам проектов 15 и 20, эта программа создает динамическое изображение с прокруткой, выводя на экран строковые значения из списка `ROWS`. В каждое строковое значение пары AT и CG вставляются с помощью метода `format()`.

```
 1. """ДНК, (c) Эл Свейгарт al@inventwithpython.com. Простое динамическое
 2. изображение двойной спирали ДНК. Нажмите Ctrl+C для останова.
 3. Вдохновлено созданным matoken сценарием https://asciinema.org/a/155441
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: короткая, графика, прокрутка, научная"""
 6.
 7. import random, sys, time
 8.
 9. PAUSE = 0.15  # (!) Попробуйте заменить это значение на 0.5 или 0.0.
10.
11. # Отдельные строки динамического изображения DNA:
12. ROWS = [
13.     #123456789 <- Для наглядной оценки количества пробелов:
```

```
    '         ##',  # У индекса 0 нет нуклеотидов {}.
    '        #{}-{}#',
    '       #{}---{}#',
    '      #{}-----{}#',
    '     #{}------{}#',
    '    #{}------{}#',
    '    #{}-----{}#',
    '     #{}---{}#',
    '     #{}-{}#',
    '      ##',  # У индекса 9 нет нуклеотидов {}.
    '     #{}-{}#',
    '     #{}---{}#',
    '    #{}-----{}#',
    '    #{}------{}#',
    '     #{}------{}#',
    '      #{}-----{}#',
    '       #{}---{}#',
    '        #{}-{}#']
    #123456789 <- Для наглядной оценки количества пробелов:

try:
    print('DNA Animation, by Al Sweigart al@inventwithpython.com')
    print('Press Ctrl-C to quit...')
    time.sleep(2)
    rowIndex = 0

    while True:  # Основной цикл программы.
        # Увеличиваем rowIndex на 1 для отрисовки следующей строки:
        rowIndex = rowIndex + 1
        if rowIndex == len(ROWS):
            rowIndex = 0

        # У строк с индексами 0 и 9 нет нуклеотидов:
        if rowIndex == 0 or rowIndex == 9:
            print(ROWS[rowIndex])
            continue

        # Выбираем случайные пары оснований гуанин-цитозин
        # и аденин-тимин:
        randomSelection = random.randint(1, 4)
        if randomSelection == 1:
            leftNucleotide, rightNucleotide = 'A', 'T'
        elif randomSelection == 2:
            leftNucleotide, rightNucleotide = 'T', 'A'
        elif randomSelection == 3:
            leftNucleotide, rightNucleotide = 'C', 'G'
        elif randomSelection == 4:
            leftNucleotide, rightNucleotide = 'G', 'C'

        # Выводим строку на экран.
        print(ROWS[rowIndex].format(leftNucleotide, rightNucleotide))
```

```
65.         time.sleep(PAUSE)  # Вставляем небольшую паузу.
66. except KeyboardInterrupt:
67.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `rowIndex = rowIndex + 1` в строке 42 заменить на `rowIndex = rowIndex + 2`?
2. Что будет, если `random.randint(1, 4)` в строке 53 заменить на `random.randint(1, 2)`?
3. Какое сообщение об ошибке вы получите, если `PAUSE = 0.15` в строке 9 замените на `PAUSE = -0.15`?

22

Утята

Программа создает перемещающийся вверх фон с утятами. Они незначительно отличаются друг от друга: смотрят влево или вправо, обладают одним из двух размеров тела, четырех типов глаз, двух разновидностей ртов и трех положений крыльев. В результате получается 96 возможных вариантов утят, непрерывно выводимых программой.

Программа в действии

Результат выполнения `ducklings.py` выглядит следующим образом:

```
Duckling Screensaver, by Al Sweigart al@inventwithpython.com
Press Ctrl-C to quit...
                                          =" )
=")                                       (  v)=")
( ^)                                       ^ ^ ( v) >'')
 ^^                                             ^^  (  ^)
                                                     ^ ^
                          >")
                          ( v)      =^^)
 ("<  ("<               >")   ^^      (  >)
(^ ) (< )               ( ^)           ^ ^
 ^^   ^^             ("<  ^^                      (``<>^^)
 (^^=               (^ )                         (<  )(  ^)
(v  ) ( "<           ^^                           ^ ^  ^ ^
--сокращено--
```

Описание работы

Утята в этой программе представлены с помощью класса `Duckling`. В методе `__init__()` данного класса случайным образом выбираются черты каждого утенка, а различные части тела утят возвращаются методами `getHeadStr()`, `getBodyStr()` и `getFeetStr()`.

```
"""Утята, (c) Эл Свейгарт al@inventwithpython.com
Экранная заставка со множеством утят

>" )   =^^)    (``=   ("=  >")    ("=
(  >)  (  ^)  (v  )  (^ )  ( >)  (v )
 ^ ^    ^ ^    ^ ^    ^^    ^^    ^^

Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, графика, объектно-ориентированная, прокрутка"""

import random, shutil, sys, time

# Задаем константы:
PAUSE = 0.2  # (!) Попробуйте заменить это значение на 1.0 или 0.0.
DENSITY = 0.10  # (!) Замените это значение на любое из диапазона от 0.0 до 1.0.

DUCKLING_WIDTH = 5
LEFT = 'left'
RIGHT = 'right'
BEADY = 'beady'
WIDE = 'wide'
HAPPY = 'happy'
ALOOF = 'aloof'
CHUBBY = 'chubby'
VERY_CHUBBY = 'very chubby'
OPEN = 'open'
CLOSED = 'closed'
OUT = 'out'
DOWN = 'down'
UP = 'up'
HEAD = 'head'
BODY = 'body'
FEET = 'feet'

# Получаем размер окна терминала:
WIDTH = shutil.get_terminal_size()[0]
# В Windows нельзя вывести что-либо в последнем столбце без добавления
# автоматически символа новой строки, поэтому уменьшаем ширину на 1:
WIDTH -= 1


def main():
    print('Duckling Screensaver, by Al Sweigart')
```

```
    print('Press Ctrl-C to quit...')
    time.sleep(2)

    ducklingLanes = [None] * (WIDTH // DUCKLING_WIDTH)

    while True:  # Основной цикл программы.
        for laneNum, ducklingObj in enumerate(ducklingLanes):
            # Проверяем, имеет ли смысл создавать утенка на этой полоске:
            if (ducklingObj == None and random.random() <= DENSITY):
                    # Размещаем утенка на этой полоске:
                    ducklingObj = Duckling()
                    ducklingLanes[laneNum] = ducklingObj

            if ducklingObj != None:
                # Если на этой полоске есть утенок — отрисовываем:
                print(ducklingObj.getNextBodyPart(), end='')
                # Удаляем утенка по завершении его отрисовки:
                if ducklingObj.partToDisplayNext == None:
                    ducklingLanes[laneNum] = None
            else:
                # Выводим пять пробелов, раз утенка тут нет.
                print(' ' * DUCKLING_WIDTH, end='')

        print()  # Выводим символ новой строки.
        sys.stdout.flush()  # Обеспечиваем вывод текста на экран.
        time.sleep(PAUSE)


class Duckling:
    def __init__(self):
        """Создаем нового утенка со случайными отличительными чертами."""
        self.direction = random.choice([LEFT, RIGHT])
        self.body = random.choice([CHUBBY, VERY_CHUBBY])
        self.mouth = random.choice([OPEN, CLOSED])
        self.wing = random.choice([OUT, UP, DOWN])

        if self.body == CHUBBY:
            # У упитанных утят могут быть только глаза-бусинки.
            self.eyes = BEADY
        else:
            self.eyes = random.choice([BEADY, WIDE, HAPPY, ALOOF])

        self.partToDisplayNext = HEAD

    def getHeadStr(self):
        """Возвращает строковое значение с головой утенка."""
        headStr = ''
        if self.direction == LEFT:
            # Рот:
            if self.mouth == OPEN:
                headStr += '>'
```

```
 95.             elif self.mouth == CLOSED:
 96.                 headStr += '='
 97.
 98.             # Глаза:
 99.             if self.eyes == BEADY and self.body == CHUBBY:
100.                 headStr += '"'
101.             elif self.eyes == BEADY and self.body == VERY_CHUBBY:
102.                 headStr += '" '
103.             elif self.eyes == WIDE:
104.                 headStr += "'"
105.             elif self.eyes == HAPPY:
106.                 headStr += '^^'
107.             elif self.eyes == ALOOF:
108.                 headStr += '``'
109.
110.             headStr += ') '  # Затылок.
111.
112.         if self.direction == RIGHT:
113.             headStr += ' ('  # Затылок.
114.
115.             # Глаза:
116.             if self.eyes == BEADY and self.body == CHUBBY:
117.                 headStr += '"'
118.             elif self.eyes == BEADY and self.body == VERY_CHUBBY:
119.                 headStr += ' "'
120.             elif self.eyes == WIDE:
121.                 headStr += "'"
122.             elif self.eyes == HAPPY:
123.                 headStr += '^^'
124.             elif self.eyes == ALOOF:
125.                 headStr += '``'
126.
127.             # Рот:
128.             if self.mouth == OPEN:
129.                 headStr += '<'
130.             elif self.mouth == CLOSED:
131.                 headStr += '='
132.
133.         if self.body == CHUBBY:
134.             # Дополнительное пустое место, чтобы ширина упитанных
135.             # и очень упитанных утят совпадала.
136.             headStr += ' '
137.
138.         return headStr
139.
140.     def getBodyStr(self):
141.         """Возвращает строковое значение с телом утенка."""
142.         bodyStr = '('  # Левая сторона тела.
143.         if self.direction == LEFT:
144.             # Внутреннее пространство тела:
145.             if self.body == CHUBBY:
```

```
146.                 bodyStr += ' '
147.             elif self.body == VERY_CHUBBY:
148.                 bodyStr += '  '
149.
150.             # Крыло:
151.             if self.wing == OUT:
152.                 bodyStr += '>'
153.             elif self.wing == UP:
154.                 bodyStr += '^'
155.             elif self.wing == DOWN:
156.                 bodyStr += 'v'
157.
158.         if self.direction == RIGHT:
159.             # Крыло:
160.             if self.wing == OUT:
161.                 bodyStr += '<'
162.             elif self.wing == UP:
163.                 bodyStr += '^'
164.             elif self.wing == DOWN:
165.                 bodyStr += 'v'
166.
167.             # Внутреннее пространство тела:
168.             if self.body == CHUBBY:
169.                 bodyStr += ' '
170.             elif self.body == VERY_CHUBBY:
171.                 bodyStr += '  '
172.
173.         bodyStr += ')'  # Правая сторона тела.
174.
175.         if self.body == CHUBBY:
176.             # Дополнительное место, чтобы ширина упитанных и очень
177.             # упитанных утят совпадала.
178.             bodyStr += ' '
179.
180.         return bodyStr
181.
182.     def getFeetStr(self):
183.         """Возвращает строковое значение с лапками утенка."""
184.         if self.body == CHUBBY:
185.             return ' ^^  '
186.         elif self.body == VERY_CHUBBY:
187.             return ' ^ ^ '
188.
189.     def getNextBodyPart(self):
190.         """Вызываем соответствующий метод вывода для следующей
191.         отображаемой части тела утенка. По завершении
192.         partToDisplayNext задается равной None."""
193.         if self.partToDisplayNext == HEAD:
194.             self.partToDisplayNext = BODY
195.             return self.getHeadStr()
196.         elif self.partToDisplayNext == BODY:
```

```
197.             self.partToDisplayNext = FEET
198.             return self.getBodyStr()
199.         elif self.partToDisplayNext == FEET:
200.             self.partToDisplayNext = None
201.             return self.getFeetStr()
202.
203.
204.
205. # Если программа не импортируется, а запускается, производим запуск:
206. if __name__ == '__main__':
207.     try:
208.         main()
209.     except KeyboardInterrupt:
210.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!).

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `random.choice([LEFT, RIGHT])` в строке 75 заменить на `random.choice([LEFT])`?
2. Что будет, если `self.partToDisplayNext = BODY` в строке 194 заменить на `self.partToDisplayNext = None`?
3. Что будет, если `self.partToDisplayNext = FEET` в строке 197 заменить на `self.partToDisplayNext = BODY`?
4. Что будет, если `return self.getHeadStr()` в строке 195 заменить на `self.getFeetStr()`?

23

Гравировщик

При перемещении кончика пера по экрану с помощью клавиш WASD наша программа-гравировщик формирует картинку, проводя непрерывную линию, подобно игре «Волшебный экран». Откройте в себе художника и посмотрите, какие картины вы сможете нарисовать! Эта программа также дает возможность сохранять рисунки в текстовый файл, чтобы делать распечатки в будущем. Кроме того, вы можете копировать и вставлять в данную программу жесты WASD для других рисунков, например команды WASD для фрактальной кривой Гильберта, представленной в строках с 6-й по 14-ю исходного кода.

Программа в действии

Результат выполнения `etchingdrawer.py` выглядит следующим образом (рис. 23.1).

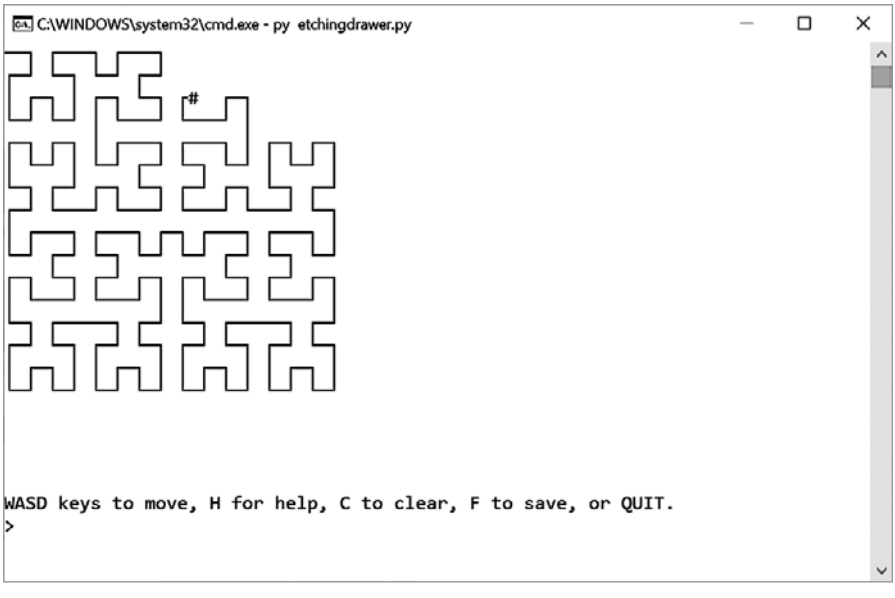


Рис. 23.1. Рисунок, сделанный в программе-гравировщике

Описание работы

Как и в программе проекта 17, в данной программе в переменной `canvas` содержится ассоциативный массив с линиями рисунка. Ключи этого ассоциативного массива представляют собой кортежи `(x, y)`, а значениями служат строковые значения, описывающие формы линий, которые нужно нарисовать на координатах `(x, y)` на экране. Полный список символов Unicode, которые можно использовать в программах на Python, приведен в приложении Б.

В строке 126 вызывается функция `open()` с аргументом `encoding='utf-8'`. Причины, почему это необходимо, выходят за рамки данной книги, но это нужно в Windows для записи символов линий в текстовый файл.

```
1. """Гравировщик, (c) Эл Свейгарт al@inventwithpython.com
2. Графическая программа, рисующая непрерывную линию на экране с помощью
3. клавиш WASD. Создана под влиянием игры "Волшебный экран".
4.
5. Например, нарисовать фрактальную кривую Гильберта можно с помощью:
```

```
SDWDDSASDSAAWASSDSASSDWDSDWWAWDDDSASSDWDSDWWAWDWWASAAWDWAWDDSDW

Или даже еще большую фрактальную кривую Гильберта с помощью:
DDSAASSDDWDDSDDWWAAWDDDDSDDWDDDDSAASDDSAAAAWAASSSDDWDDDDSAASDDSAAAAWA
ASAAAAWDDWWAASAAWAASSDDSAASSDDWDDDDSAASDDSAAAAWAASSDDSAASSDDWDDSDDWWA
AWDDDDDDSAASSDDWDDSDDWWAAWDDWWAASAAAAWDDWAAWDDDDSDDWDDSDDWDDDDSAASDDS
AAAAWAASSDDSAASSDDWDDSDDWWAAWDDDDDDSAASSDDWDDSDDWWAAWDDWWAASAAAAWDDWA
AWDDDDSDDWWAAWDDWWAASAAWAASSDDSAAAAWAASAAAAWDDWAAWDDDDSDDWWWAASAAAAWD
DWAAWDDDDSDDWDDDDSAASSDDWDDSDDWWAAWDD

Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, графика"""

import shutil, sys

# Задаем константы для символов линий:
UP_DOWN_CHAR         = chr(9474)  # Символ 9474 — '│'
LEFT_RIGHT_CHAR      = chr(9472)  # Символ 9472 — '─'
DOWN_RIGHT_CHAR      = chr(9484)  # Символ 9484 —'┌'
DOWN_LEFT_CHAR       = chr(9488)  # Символ 9488 — '┐'
UP_RIGHT_CHAR        = chr(9492)  # Символ 9492 — '└'
UP_LEFT_CHAR         = chr(9496)  # Символ 9496 — '┘'
UP_DOWN_RIGHT_CHAR   = chr(9500)  # Символ 9500 — '├'
UP_DOWN_LEFT_CHAR    = chr(9508)  # Символ 9508 — '┤'
DOWN_LEFT_RIGHT_CHAR = chr(9516)  # Символ 9516 — '┬'
UP_LEFT_RIGHT_CHAR   = chr(9524)  # Символ 9524 — '┴'
CROSS_CHAR           = chr(9532)  # Символ 9532 — '┼'
# Список кодов chr() можно найти в https://inventwithpython.com/chr

# Получаем размер окна терминала:
CANVAS_WIDTH, CANVAS_HEIGHT = shutil.get_terminal_size()
# В Windows нельзя вывести что-либо в последнем столбце без добавления
# автоматически символа новой строки, поэтому уменьшаем ширину на 1:
CANVAS_WIDTH -= 1
# Оставляем место в нескольких нижних строках для информации о команде.
CANVAS_HEIGHT -= 5

"""Ключи ассоциативного массива canvas представляют собой целочисленные
кортежи (x, y) координат, а значение — набор букв W, A, S, D,
описывающих тип отрисовываемой линии."""
canvas = {}
cursorX = 0
cursorY = 0


def getCanvasString(canvasData, cx, cy):
    """Возвращает многострочное значение рисуемой в canvasData линии."""
    canvasStr = ''

    """canvasData — ассоциативный массив с ключами (x, y) и значениями
    в виде множеств из строк символов 'W', 'A', 'S' и/или 'D',
    описывающих, в каком направлении идет линия в каждой точке xy."""
```

```
    for rowNum in range(CANVAS_HEIGHT):
        for columnNum in range(CANVAS_WIDTH):
            if columnNum == cx and rowNum == cy:
                canvasStr += '#'
                continue

            # Добавляем символ линии для данной точки в canvasStr.
            cell = canvasData.get((columnNum, rowNum))
            if cell in (set(['W', 'S']), set(['W']), set(['S'])):
                canvasStr += UP_DOWN_CHAR
            elif cell in (set(['A', 'D']), set(['A']), set(['D'])):
                canvasStr += LEFT_RIGHT_CHAR
            elif cell == set(['S', 'D']):
                canvasStr += DOWN_RIGHT_CHAR
            elif cell == set(['A', 'S']):
                canvasStr += DOWN_LEFT_CHAR
            elif cell == set(['W', 'D']):
                canvasStr += UP_RIGHT_CHAR
            elif cell == set(['W', 'A']):
                canvasStr += UP_LEFT_CHAR
            elif cell == set(['W', 'S', 'D']):
                canvasStr += UP_DOWN_RIGHT_CHAR
            elif cell == set(['W', 'S', 'A']):
                canvasStr += UP_DOWN_LEFT_CHAR
            elif cell == set(['A', 'S', 'D']):
                canvasStr += DOWN_LEFT_RIGHT_CHAR
            elif cell == set(['W', 'A', 'D']):
                canvasStr += UP_LEFT_RIGHT_CHAR
            elif cell == set(['W', 'A', 'S', 'D']):
                canvasStr += CROSS_CHAR
            elif cell == None:
                canvasStr += ' '
        canvasStr += '\n'  # Добавляем в конце строк символ новой строки.
    return canvasStr


moves = []
while True:  # Основной цикл программы.
    # Отрисовываем линии, исходя из содержащихся в canvas данных:
    print(getCanvasString(canvas, cursorX, cursorY))

    print('WASD keys to move, H for help, C to clear, '
        + 'F to save, or QUIT.')
    response = input('> ').upper()

    if response == 'QUIT':
        print('Thanks for playing!')
        sys.exit()  # Выходим из программы.
    elif response == 'H':
        print('Enter W, A, S, and D characters to move the cursor and')
        print('draw a line behind it as it moves. For example, ddd')
```

```
109.         print('draws a line going right and sssdddwwwaaa draws a box.')
110.         print()
111.         print('You can save your drawing to a text file by entering F.')
112.         input('Press Enter to return to the program...')
113.         continue
114.     elif response == 'C':
115.         canvas = {}  # Очищаем canvas.
116.         moves.append('C')  # Записываем движение.
117.     elif response == 'F':
118.         # Сохраняем строковое значение с холстом в текстовый файл:
119.         try:
120.             print('Enter filename to save to:')
121.             filename = input('> ')
122.
123.             # Проверяем, чтобы имя файла оканчивалось на .txt:
124.             if not filename.endswith('.txt'):
125.                 filename += '.txt'
126.             with open(filename, 'w', encoding='utf-8') as file:
127.                 file.write(''.join(moves) + '\n')
128.                 file.write(getCanvasString(canvas, None, None))
129.         except:
130.             print('ERROR: Could not save file.')
131.
132.     for command in response:
133.         if command not in ('W', 'A', 'S', 'D'):
134.             continue  # Игнорируем букву и переходим к следующей.
135.         moves.append(command)  # Фиксируем данное движение.
136.
137.         # Первая добавляемая линия должна формировать полную строку:
138.         if canvas == {}:
139.             if command in ('W', 'S'):
140.                 # Делаем первую линию горизонтальной:
141.                 canvas[(cursorX, cursorY)] = set(['W', 'S'])
142.             elif command in ('A', 'D'):
143.                 # Делаем первую линию вертикальной:
144.                 canvas[(cursorX, cursorY)] = set(['A', 'D'])
145.
146.         # Обновляем значения x и y:
147.         if command == 'W' and cursorY > 0:
148.             canvas[(cursorX, cursorY)].add(command)
149.             cursorY = cursorY - 1
150.         elif command == 'S' and cursorY < CANVAS_HEIGHT - 1:
151.             canvas[(cursorX, cursorY)].add(command)
152.             cursorY = cursorY + 1
153.         elif command == 'A' and cursorX > 0:
154.             canvas[(cursorX, cursorY)].add(command)
155.             cursorX = cursorX - 1
156.         elif command == 'D' and cursorX < CANVAS_WIDTH - 1:
157.             canvas[(cursorX, cursorY)].add(command)
158.             cursorX = cursorX + 1
159.         else:
```

```
            # Если курсор не двигается, чтобы не выйти за пределы холста,
            # то не меняем множество в canvas[(cursorX, cursorY)]
            # canvas[(cursorX, cursorY)].
            continue

        # Если не существует множества для (cursorX, cursorY), добавляем
        # пустое множество:
        if (cursorX, cursorY) not in canvas:
            canvas[(cursorX, cursorY)] = set()

        # Добавляем строку с направлением во множество для этой точки xy:
        if command == 'W':
            canvas[(cursorX, cursorY)].add('S')
        elif command == 'S':
            canvas[(cursorX, cursorY)].add('W')
        elif command == 'A':
            canvas[(cursorX, cursorY)].add('D')
        elif command == 'D':
            canvas[(cursorX, cursorY)].add('A')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `response = input('> ').upper()` в строке 101 заменить на `response = input('> ')`?
2. Что будет, если `canvasStr += '#'` в строке 61 заменить на `canvasStr += '@'`?
3. Что будет, если `canvasStr += ' '` в строке 89 заменить на `canvasStr + = '.'`?
4. Что будет, если выражение `moves = []` в строке 94 заменить на `moves = list ('SDWDDSASDSAAWASSDSAS')`?

24

Разложение на множители

Множители заданного числа — другие[1] два числа, произведение которых дает это заданное число. Например, 2 × 13 = 26, так что 2 и 13 — множители 26. Кроме того, 1 × 26 = 26, так что 1 и 26 — также множители 26. Следовательно, у 26 четыре множителя: 1, 2, 13 и 26.

Число, у которого только два множителя: 1 и оно само, называется простым. В противном случае оно называется составным. Найдите новые простые числа с помощью нашей программы разложения на множители![2] (Подсказка: простые числа всегда заканчиваются на нечетную цифру, но не 5.) Вдобавок в проекте 56 мы покажем, как вычислить их с помощью компьютера.

Математика в этой программе не слишком сложна, так что она идеально подходит для начинающих.

---

[1] Обычно само число также считается своим множителем, и далее автор тоже это упоминает. — *Примеч. пер.*

[2] Термин «факторизация», полагаю, здесь не подходит, так как под ним обычно понимают разложение именно на простые множители. — *Примеч. пер.*

Программа в действии

Результат выполнения `factorfinder.py` выглядит следующим образом:

```
Factor Finder, by Al Sweigart al@inventwithpython.com
--сокращено--
Enter a number to factor (or "QUIT" to quit):
> 26
1, 2, 13, 26
Enter a number to factor (or "QUIT" to quit):
> 4352784
1, 2, 3, 4, 6, 8, 12, 16, 24, 29, 48, 53, 58, 59, 87, 106, 116, 118, 159,
174, 177, 212, 232, 236, 318, 348, 354, 424, 464, 472, 636, 696, 708, 848,
944, 1272, 1392, 1416, 1537, 1711, 2544, 2832, 3074, 3127, 3422, 4611, 5133,
6148, 6254, 6844, 9222, 9381, 10266, 12296, 12508, 13688, 18444, 18762, 20532,
24592, 25016, 27376, 36888, 37524, 41064, 50032, 73776, 75048, 82128, 90683,
150096, 181366, 272049, 362732, 544098, 725464, 1088196, 1450928, 2176392,
4352784
Enter a number to factor (or "QUIT" to quit):
> 9787
1, 9787
Enter a number to factor (or "QUIT" to quit):
> quit
```

Описание работы

Узнать, является ли число множителем другого числа, можно, проверив, делится ли это второе число на первое без остатка. Например, 7 — множитель 21, поскольку 21 ÷ 7 равно 3. Кроме того, это значит, что 3 — также множитель 21. Однако 8 — не множитель 21, поскольку 21 ÷ 8 = 2,625. Дробная часть означает наличие остатка, так что нацело они не делятся.

Оператор деления по модулю `%` демонстрирует, есть ли остаток: `21 % 7` оказывается равно `0`, так что остатка нет, и 7 — множитель 21, в то время как `21 % 8` равно `1`, ненулевое значение, означающее что 8 — не один из множителей 21. Эта методика используется в нашей программе разложения на множители в строке 35 для определения того, является ли число множителем.

Функция `math.sqrt()` возвращает квадратный корень передаваемого в нее числа. Например, `math.sqrt(25)` возвращает `5.0`, поскольку 5 в квадрате равно 25, а значит, 5 является квадратным корнем из 25.

```
1. """Разложение на множители, (c) Эл Свейгарт al@inventwithpython.com
2. Находит все множители заданного числа
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: крошечная, для начинающих, математическая"""
5.
6. import math, sys
```

```
 7.
 8. print('''Factor Finder, by Al Sweigart al@inventwithpython.com
 9.
10. A number's factors are two numbers that, when multiplied with each
11. other, produce the number. For example, 2 x 13 = 26, so 2 and 13 are
12. factors of 26. 1 x 26 = 26, so 1 and 26 are also factors of 26. We
13. say that 26 has four factors: 1, 2, 13, and 26.
14.
15. If a number only has two factors (1 and itself), we call that a prime
16. number. Otherwise, we call it a composite number.
17.
18. Can you discover some prime numbers?
19. ''')
20.
21. while True:  # Основной цикл программы.
22.     print('Enter a positive whole number to factor (or QUIT):')
23.     response = input('> ')
24.     if response.upper() == 'QUIT':
25.         sys.exit()
26.
27.     if not (response.isdecimal() and int(response) > 0):
28.         continue
29.     number = int(response)
30.
31.     factors = []
32.
33.     # Поиск множителей числа:
34.     for i in range(1, int(math.sqrt(number)) + 1):
35.         if number % i == 0:  # Если остатка нет, значит — множитель.
36.             factors.append(i)
37.             factors.append(number // i)
38.
39.     # Преобразуем во множество, чтобы избавиться от повторов:
40.     factors = list(set(factors))
41.     factors.sort()
42.
43.     # Выводим результаты:
44.     for i, factor in enumerate(factors):
45.         factors[i] = str(factor)
46.     print(', '.join(factors))
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `factors.append(i)` в строке 36?

2. Что будет, если удалить или закомментировать `factors = list(set(factors))` в строке 40? (Подсказка: введите на входе квадрат какого-нибудь числа, например 25, 36 или 49.)
3. Что будет, если удалить или закомментировать `factors.sort()` в строке 41?
4. Какое сообщение об ошибке вы получите, если `factors = []` в строке 31 замените на `factors = ''`?
5. Что будет, если `factors = []` в строке 31 заменить на `factors = [-42]`?
6. Какое сообщение об ошибке вы получите, если `factors = []` в строке 31 замените на `factors = ['hello']`?

25

Быстрый стрелок

Эта программа проверяет быстроту вашей реакции: вы должны нажать Enter сразу же, как только увидите слово DRAW. Однако осторожнее: нажмете Enter до появления DRAW — и проиграете! Самый ли вы быстрый стрелок на Диком Западе?

Программа в действии

Результат выполнения `fastdraw.py` выглядит следующим образом:

```
Fast Draw, by Al Sweigart al@inventwithpython.com

Time to test your reflexes and see if you are the fastest
draw in the west!
When you see "DRAW", you have 0.3 seconds to press Enter.
But you lose if you press Enter before "DRAW" appears.

Press Enter to begin...

It is high noon...
DRAW!

You took 0.3485 seconds to draw. Too slow!
Enter QUIT to stop, or press Enter to play again.
> quit
Thanks for playing!
```

Описание работы

Функция `input()` приостанавливает программу в ожидании ввода пользователем строкового значения. Столь простое поведение означает, что создавать игры в режиме реального времени с помощью одной только этой функции не получится. Однако программы *буферизуют* вводимые с клавиатуры данные, так что, если нажать клавиши C, A и T до вызова `input()`, они будут сохранены и появятся сразу же после выполнения `input()`.

Фиксация времени непосредственно перед вызовом `input()` в строке 22 и сразу же после этого вызова в строке 24 позволяет определить, сколько времени потребовалось игроку, чтобы нажать Enter. Однако если данная клавиша была нажата до вызова `input()`, то буферизованное нажатие Enter приведет к мгновенному возврату из `input()` (обычно примерно за 3 миллисекунды). Поэтому в строке 26 мы проверяем, не было ли время меньше 0,01 секунды (то есть 10 миллисекунд), чтобы выяснить, не нажал ли игрок Enter слишком рано.

```
 1. """Быстрый стрелок, (c) Эл Свейгарт al@inventwithpython.com
 2. Проверьте свои рефлексы и узнайте, самый ли вы быстрый стрелок на Диком Западе.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, игра"""
 5.
 6. import random, sys, time
 7.
 8. print('Fast Draw, by Al Sweigart al@inventwithpython.com')
 9. print()
10. print('Time to test your reflexes and see if you are the fastest')
11. print('draw in the west!')
12. print('When you see "DRAW", you have 0.3 seconds to press Enter.')
13. print('But you lose if you press Enter before "DRAW" appears.')
14. print()
15. input('Press Enter to begin...')
16.
17. while True:
18.     print()
19.     print('It is high noon...')
20.     time.sleep(random.randint(20, 50) / 10.0)
21.     print('DRAW!')
22.     drawTime = time.time()
23.     input()  # Возврат из этой функции не происходит до нажатия Enter.
24.     timeElapsed = time.time() - drawTime
25.
26.     if timeElapsed < 0.01:
27.         # Если игрок нажал Enter до появления DRAW!, возврат из вызова
28.         # input() происходит практически мгновенно.
29.         print('You drew before "DRAW" appeared! You lose.')
30.     elif timeElapsed > 0.3:
31.         timeElapsed = round(timeElapsed, 4)
```

```
32.         print('You took', timeElapsed, 'seconds to draw. Too slow!')
33.     else:
34.         timeElapsed = round(timeElapsed, 4)
35.         print('You took', timeElapsed, 'seconds to draw.')
36.         print('You are the fastest draw in the west! You win!')
37.
38.     print('Enter QUIT to stop, or press Enter to play again.')
39.     response = input('> ').upper()
40.     if response == 'QUIT':
41.         print('Thanks for playing!')
42.         sys.exit()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `drawTime = time.time()` в строке 22 заменить на `drawTime = 0`?
2. Что будет, если `timeElapsed > 0.3` в строке 30 заменить на `timeElapsed < 0.3`?
3. Что будет, если `time.time() - drawTime` в строке 24 заменить на `time.time() + drawTime`?
4. Что будет, если удалить или закомментировать `input('Press Enter to begin...')` в строке 15?

26

Фибоначчи

Последовательность Фибоначчи — знаменитая математическая закономерность, открытие которой приписывается итальянскому математику Фибоначчи, жившему в XIII веке (хотя на самом деле она была открыта еще раньше). Последовательность начинается с 0 и 1, а каждое следующее число равно сумме двух предыдущих. Последовательность бесконечна:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987 ...

Последовательность Фибоначчи применяется при написании музыки, прогнозах курсов акций, определении закономерностей соцветий в головках подсолнечника и многих других сферах. Наша программа позволяет рассчитать столько чисел Фибоначчи, сколько вам нужно. Больше информации об этой последовательности можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Числа_Фибоначчи.

Программа в действии

Результат выполнения `fibonacci.py` выглядит следующим образом:

```
Fibonacci Sequence, by Al Sweigart al@inventwithpython.com
--сокращено--
Enter the Nth Fibonacci number you wish to
calculate (such as 5, 50, 1000, 9999), or QUIT to quit:
> 50
```

```
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181, 6765, 10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229,
832040, 1346269, 2178309, 3524578, 5702887, 9227465, 14930352, 24157817,
39088169, 63245986, 102334155, 165580141, 267914296, 433494437, 701408733,
1134903170, 1836311903, 2971215073, 4807526976, 7778742049
```

Описание работы

Поскольку числа Фибоначчи быстро вырастают до очень больших значений, в строках с 46-й по 50-ю мы проверяем, не ввел ли пользователь число, превышающее 10 000, и выводим в этом случае предупреждение, что отображение результатов на экране может занять некоторое время. И хотя программа может выполнять миллионы вычислений практически мгновенно, вывод текста на экран происходит относительно медленно и может занимать несколько секунд. Предупреждение сообщает пользователю, что он всегда может прервать выполнение программы, нажав сочетание клавиш Ctrl+C.

```
 1. """Последовательность Фибоначчи, (c) Эл Свейгарт al@inventwithpython.com
 2. Вычисляет числа из последовательности Фибоначчи: 0, 1, 1, 2, 3, 5, 8, 13...
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, математическая"""
 5.
 6. import sys
 7.
 8. print('''Fibonacci Sequence, by Al Sweigart al@inventwithpython.com
 9.
10. The Fibonacci sequence begins with 0 and 1, and the next number is the
11. sum of the previous two numbers. The sequence continues forever:
12.
13. 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987...
14. ''')
15.
16. while True:  # Основной цикл программы.
17.     while True:  # Спрашиваем, пока пользователь не введет допустимое число.
18.         print('Enter the Nth Fibonacci number you wish to')
19.         print('calculate (such as 5, 50, 1000, 9999), or QUIT to quit:')
20.         response = input('> ').upper()
21.
22.         if response == 'QUIT':
23.             print('Thanks for playing!')
24.             sys.exit()
25.
26.         if response.isdecimal() and int(response) != 0:
27.             nth = int(response)
28.             break  # Когда пользователь ввел допустимое число — выходим из цикла
29.
30.         print('Please enter a number greater than 0, or QUIT.')
```

```
31.     print()
32.
33.     # Обработка частных случаев, если пользователь ввел 1 или 2:
34.     if nth == 1:
35.         print('0')
36.         print()
37.         print('The #1 Fibonacci number is 0.')
38.         continue
39.     elif nth == 2:
40.         print('0, 1')
41.         print()
42.         print('The #2 Fibonacci number is 1.')
43.         continue
44.
45.     # Отображаем предупреждение, если пользователь ввел большое число:
46.     if nth >= 10000:
47.         print('WARNING: This will take a while to display on the')
48.         print('screen. If you want to quit this program before it is')
49.         print('done, press Ctrl-C.')
50.         input('Press Enter to begin...')
51.
52.     # Вычисляем N-е число Фибоначчи:
53.     secondToLastNumber = 0
54.     lastNumber = 1
55.     fibNumbersCalculated = 2
56.     print('0, 1, ', end='')  # Выводим первые два числа Фибоначчи.
57.
58.     # Выводим все остальные числа Фибоначчи:
59.     while True:
60.         nextNumber = secondToLastNumber + lastNumber
61.         fibNumbersCalculated += 1
62.
63.         # Выводим следующее число последовательности:
64.         print(nextNumber, end='')
65.
66.         # Проверяем, нашли ли мы требуемое пользователем N-е число:
67.         if fibNumbersCalculated == nth:
68.             print()
69.             print()
70.             print('The #', fibNumbersCalculated, ' Fibonacci ',
71.                   'number is ', nextNumber, sep='')
72.             break
73.
74.         # Выводим запятую между членами последовательности:
75.         print(', ', end='')
76.
77.         # Заменяем последние два числа:
78.         secondToLastNumber = lastNumber
79.         lastNumber = nextNumber
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- использовать отличные от 0 и 1 начальные числа;
- вычислять следующее число как сумму трех, а не двух предыдущих.

Исследование программы

Это очень простая программа, так что вариантов ее модификации не так уж много. Вместо этого задумайтесь: для чего ее можно использовать? И какие еще полезные последовательности можно вычислить программным образом?

27

Аквариум

Наблюдайте за вашими собственными виртуальными рыбками в виртуальном аквариуме, заполненном пузырьками воздуха и крупными водорослями! При каждом запуске программа генерирует случайным образом рыбок различных видов и цветов. Сделайте паузу и насладитесь успокаивающей тишью этого программного аквариума или попробуйте запрограммировать виртуальных акул, чтобы вселить ужас в его обитателей! Данную программу нельзя запустить из IDE или редактора, поскольку она использует модуль `bext` и для правильного отображения требует запуска из командной строки или терминала. Больше информации о модуле `bext` можно найти по адресу https://pypi.org/project/bext/.

Программа в действии

На рис. 27.1 показано, как выглядит результат выполнения `fishtank.py`.

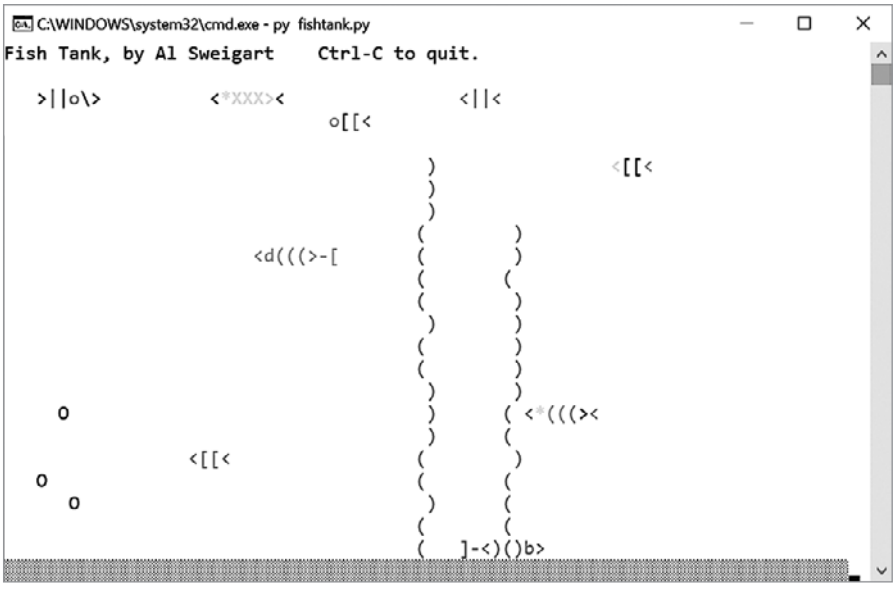


Рис. 27.1. Результат работы программы-аквариума с несколькими рыбками, крупными водорослями и пузырьками воздуха

Описание работы

Современные графические программы для генерации динамических изображений часто полностью очищают все окно и перерисовывают его 30 или 60 раз в секунду. Получается *частота кадров* (frame rate, FPS) 30 или 60 кадров в секунду. Чем выше FPS, тем более плавным выглядит движение.

Вывод в окно терминала происходит намного медленнее. Если очищать все окно терминала для перерисовки его содержимого с помощью модуля `bext`, то получится только около 3 или 4 FPS, а значит, мерцание изображения в окне будет довольно заметным.

Можно ускорить этот процесс, рисуя символы лишь в тех частях экрана терминала, которые поменялись. Большую часть вывода программы-аквариума составляет пустое пространство, так что функции `clearAquarium()` для имитации движения элементов достаточно только выводить символы пробела `' '` там, где сейчас находятся рыбки, водоросли и пузырьки. В результате частота кадров растет,

мерцание изображения уменьшается и смотреть на изображение аквариума гораздо приятнее.

```
"""Аквариум, (c) Эл Свейгарт al@inventwithpython.com
Безмятежное динамическое изображение аквариума. Нажмите Ctrl+C для останова.
Аналогична ASCIIQuarium и @EmojiAquarium, но моя программа основана на
более старой программе ASCII-аквариума под DOS.
https://robobunny.com/projects/asciiquarium/html/
https://twitter.com/EmojiAquarium
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: очень большая, графика, bext"""

import random, sys, time

try:
    import bext
except ImportError:
    print('This program requires the bext module, which you')
    print('can install by following the instructions at')
    print('https://pypi.org/project/Bext/')
    sys.exit()

# Задаем константы:
WIDTH, HEIGHT = bext.size()
# В Windows нельзя вывести что-либо в последнем столбце без добавления
# автоматически символа новой строки, поэтому уменьшаем ширину на 1:
WIDTH -= 1

NUM_KELP = 2  # (!) Попробуйте заменить это значение на 10.
NUM_FISH = 10  # (!) Попробуйте заменить это значение на 2 или 100.
NUM_BUBBLERS = 1  # (!) Попробуйте заменить это значение на 0 или 10.
FRAMES_PER_SECOND = 4  # (!) Попробуйте заменить это число на 1 или 60.
# (!) Попробуйте изменить константы, чтобы получился аквариум с одними
# только водорослями или пузырьками.
# Примечание: все строковые значения в ассоциативном массиве рыбок должны быть
# одинаковой длины.
FISH_TYPES = [
  {'right': ['><>'],          'left': ['<><']},
  {'right': ['>||>'],         'left': ['<||<']},
  {'right': ['>))>'],         'left': ['<[[<']},
  {'right': ['>||o', '>||.'], 'left': ['o||<', '.||<']},
  {'right': ['>))o', '>)).'], 'left': ['o[[<', '.[[<']},
  {'right': ['>-==>'],        'left': ['<==-<']},
  {'right': [r'>\\>'],        'left': ['<//<']},
  {'right': ['><)))*>'],      'left': ['<*(((><']},
  {'right': ['}-[[[*>'],      'left': ['<*]]]-{']},
  {'right': [']-<)))b>'],     'left': ['<d(((>-[']},
  {'right': ['><XXX*>'],      'left': ['<*XXX><']},
  {'right': ['_.-._.-^=>', '.-._.-.^=>',
             '-._.-._^=>', '._.-._.^=>'],
   'left':  ['<=^-._.-._', '<=^.-._.-.',
             '<=^_.-._.-', '<=^._.-._.']},
```

```
 50.    ]  # (!) Попробуйте добавить в FISH_TYPES свои типы рыб.
 51. LONGEST_FISH_LENGTH = 10  # Максимальная длина строкового значения в FISH_TYPES.
 52.
 53. # Позиции по x и y, при которых рыбка наталкивается на край экрана:
 54. LEFT_EDGE = 0
 55. RIGHT_EDGE = WIDTH - 1 - LONGEST_FISH_LENGTH
 56. TOP_EDGE = 0
 57. BOTTOM_EDGE = HEIGHT - 2
 58.
 59.
 60. def main():
 61.     global FISHES, BUBBLERS, BUBBLES, KELPS, STEP
 62.     bext.bg('black')
 63.     bext.clear()
 64.
 65.     # Генерируем глобальные переменные:
 66.     FISHES = []
 67.     for i in range(NUM_FISH):
 68.         FISHES.append(generateFish())
 69.
 70.     # Примечание: мы нарисовали пузырьки, но не сами воздуховоды.
 71.     BUBBLERS = []
 72.     for i in range(NUM_BUBBLERS):
 73.         # Все воздуховоды начинаются в разных местах.
 74.         BUBBLERS.append(random.randint(LEFT_EDGE, RIGHT_EDGE))
 75.     BUBBLES = []
 76.
 77.     KELPS = []
 78.     for i in range(NUM_KELP):
 79.         kelpx = random.randint(LEFT_EDGE, RIGHT_EDGE)
 80.         kelp = {'x': kelpx, 'segments': []}
 81.         # Генерируем сегменты водорослей:
 82.         for i in range(random.randint(6, HEIGHT - 1)):
 83.             kelp['segments'].append(random.choice(['(', ')']))
 84.         KELPS.append(kelp)
 85.
 86.     # Запускаем моделирование:
 87.     STEP = 1
 88.     while True:
 89.         simulateAquarium()
 90.         drawAquarium()
 91.         time.sleep(1 / FRAMES_PER_SECOND)
 92.         clearAquarium()
 93.         STEP += 1
 94.
 95.
 96. def getRandomColor():
 97.     """Возвращает строковое значение со случайным цветом."""
 98.     return random.choice(('black', 'red', 'green', 'yellow', 'blue',
 99.                           'purple', 'cyan', 'white'))
100.
```

```
101.
102. def generateFish():
103.     """Возвращает соответствующий рыбке ассоциативный массив."""
104.     fishType = random.choice(FISH_TYPES)
105.
106.     # Задает цвета для каждого из символов рыбки:
107.     colorPattern = random.choice(('random', 'head-tail', 'single'))
108.     fishLength = len(fishType['right'][0])
109.     if colorPattern == 'random':  # Все части окрашиваются случайным образом.
110.         colors = []
111.         for i in range(fishLength):
112.             colors.append(getRandomColor())
113.     if colorPattern == 'single' or colorPattern == 'head-tail':
114.         colors = [getRandomColor()] * fishLength  # Все одного цвета.
115.     if colorPattern == 'head-tail':  # Голова/хвост — отличного от тела цвета.
116.         headTailColor = getRandomColor()
117.         colors[0] = headTailColor  # Задаем цвет головы.
118.         colors[-1] = headTailColor  # Задаем цвет хвоста.
119.
120.     # Задаем остальные части структуры данных для рыбки:
121.     fish = {'right':            fishType['right'],
122.             'left':             fishType['left'],
123.             'colors':           colors,
124.             'hSpeed':           random.randint(1, 6),
125.             'vSpeed':           random.randint(5, 15),
126.             'timeToHDirChange': random.randint(10, 60),
127.             'timeToVDirChange': random.randint(2, 20),
128.             'goingRight':       random.choice([True, False]),
129.             'goingDown':        random.choice([True, False])}
130.
131.     # 'x' - всегда крайняя слева сторона тела рыбки:
132.     fish['x'] = random.randint(0, WIDTH - 1 - LONGEST_FISH_LENGTH)
133.     fish['y'] = random.randint(0, HEIGHT - 2)
134.     return fish
135.
136.
137. def simulateAquarium():
138.     """Моделирует один шаг движений в аквариуме."""
139.     global FISHES, BUBBLERS, BUBBLES, KELP, STEP
140.
141.     # Моделирует один шаг движения рыбки:
142.     for fish in FISHES:
143.         # Горизонтальное перемещение рыбки:
144.         if STEP % fish['hSpeed'] == 0:
145.             if fish['goingRight']:
146.                 if fish['x'] != RIGHT_EDGE:
147.                     fish['x'] += 1  # Перемещение рыбки вправо.
148.                 else:
149.                     fish['goingRight'] = False  # Поворот рыбки.
150.                     fish['colors'].reverse()  # Инверсия цветов.
151.             else:
```

```
152.                 if fish['x'] != LEFT_EDGE:
153.                     fish['x'] -= 1  # Перемещение рыбки влево.
154.                 else:
155.                     fish['goingRight'] = True  # Поворот рыбки.
156.                     fish['colors'].reverse()  # Инверсия цветов.
157.         # Рыбки могут случайным образом менять направление горизонтального
158.         # движения:
159.         fish['timeToHDirChange'] -= 1
160.         if fish['timeToHDirChange'] == 0:
161.             fish['timeToHDirChange'] = random.randint(10, 60)
162.             # Поворот рыбки:
163.             fish['goingRight'] = not fish['goingRight']
164.
165.         # Вертикальное перемещение рыбки:
166.         if STEP % fish['vSpeed'] == 0:
167.             if fish['goingDown']:
168.                 if fish['y'] != BOTTOM_EDGE:
169.                     fish['y'] += 1  # Перемещение рыбки вниз.
170.                 else:
171.                     fish['goingDown'] = False  # Поворот рыбки.
172.             else:
173.                 if fish['y'] != TOP_EDGE:
174.                     fish['y'] -= 1  # Перемещение рыбки вверх.
175.                 else:
176.                     fish['goingDown'] = True  # Поворот рыбки.
177.         # Рыбки могут случайным образом менять направление вертикального
178.         # движения:
179.         fish['timeToVDirChange'] -= 1
180.         if fish['timeToVDirChange'] == 0:
181.             fish['timeToVDirChange'] = random.randint(2, 20)
182.             # Поворот рыбки:
183.             fish['goingDown'] = not fish['goingDown']
184.
185.     # Генерируем пузырьки из воздуховодов:
186.     for bubbler in BUBBLERS:
187.         # Вероятность создания пузырька: 1 из 5:
188.         if random.randint(1, 5) == 1:
189.             BUBBLES.append({'x': bubbler, 'y': HEIGHT - 2})
190.
191.     # Перемещаем пузырьки:
192.     for bubble in BUBBLES:
193.         diceRoll = random.randint(1, 6)
194.         if (diceRoll == 1) and (bubble['x'] != LEFT_EDGE):
195.             bubble['x'] -= 1  # Пузырек воздуха перемещается влево.
196.         elif (diceRoll == 2) and (bubble['x'] != RIGHT_EDGE):
197.             bubble['x'] += 1  # Пузырек воздуха перемещается вправо.
198.
199.         bubble['y'] -= 1  # Пузырек воздуха всегда поднимается вверх.
200.
201.     # Проходим в цикле по BUBBLES в обратном порядке для удаления
202.     # из BUBBLES в цикле.
```

```
    for i in range(len(BUBBLES) - 1, -1, -1):
        if BUBBLES[i]['y'] == TOP_EDGE:  # Удаляем достигшие верха пузырьки.
            del BUBBLES[i]

    # Моделирует один шаг колебания водоросли:
    for kelp in KELPS:
        for i, kelpSegment in enumerate(kelp['segments']):
            # Вероятность смены колебаний — 1 из 20:
            if random.randint(1, 20) == 1:
                if kelpSegment == '(':
                    kelp['segments'][i] = ')'
                elif kelpSegment == ')':
                    kelp['segments'][i] = '('


def drawAquarium():
    """Отрисовываем аквариум на экране."""
    global FISHES, BUBBLERS, BUBBLES, KELP, STEP

    # Сообщение о возможности прервать отрисовку.
    bext.fg('white')
    bext.goto(0, 0)
    print('Fish Tank, by Al Sweigart    Ctrl-C to quit.', end='')

    # Отрисовываем пузырьки воздуха:
    bext.fg('white')
    for bubble in BUBBLES:
        bext.goto(bubble['x'], bubble['y'])
        print(random.choice(('o', 'O')), end='')

    # Отрисовываем рыбок:
    for fish in FISHES:
        bext.goto(fish['x'], fish['y'])

        # Получаем правильный текст для рыбки, смотрящей влево или вправо.
        if fish['goingRight']:
            fishText = fish['right'][STEP % len(fish['right'])]
        else:
            fishText = fish['left'][STEP % len(fish['left'])]

        # Отрисовываем все символы рыбки правильными цветами.
        for i, fishPart in enumerate(fishText):
            bext.fg(fish['colors'][i])
            print(fishPart, end='')

    # Отрисовываем водоросли:
    bext.fg('green')
    for kelp in KELPS:
        for i, kelpSegment in enumerate(kelp['segments']):
            if kelpSegment == '(':
                bext.goto(kelp['x'], BOTTOM_EDGE - i)
```

```
254.             elif kelpSegment == ')':
255.                 bext.goto(kelp['x'] + 1, BOTTOM_EDGE - i)
256.             print(kelpSegment, end='')
257.
258.     # Отрисовываем песок на дне:
259.     bext.fg('yellow')
260.     bext.goto(0, HEIGHT - 1)
261.     print(chr(9617) * (WIDTH - 1), end='')  # Draws sand.
262.
263.     sys.stdout.flush()  # (Необходимо для использующих модуль bext программ.)
264.
265.
266. def clearAquarium():
267.     """Зарисовываем весь экран пробелами."""
268.     global FISHES, BUBBLERS, BUBBLES, KELP
269.
270.     # Зарисовываем пузырьки воздуха:
271.     for bubble in BUBBLES:
272.         bext.goto(bubble['x'], bubble['y'])
273.         print(' ', end='')
274.
275.     # Зарисовываем рыбок:
276.     for fish in FISHES:
277.         bext.goto(fish['x'], fish['y'])
278.
279.         # Отрисовываем все символы текста рыбки правильными цветами.
280.         print(' ' * len(fish['left'][0]), end='')
281.
282.     # Зарисовываем водоросли:
283.     for kelp in KELPS:
284.         for i, kelpSegment in enumerate(kelp['segments']):
285.             bext.goto(kelp['x'], HEIGHT - 2 - i)
286.             print('  ', end='')
287.
288.     sys.stdout.flush()  # (Необходимо для использующих модуль bext программ.)
289.
290.
291. # Если программа не импортируется, а запускается, производим запуск:
292. if __name__ == '__main__':
293.     try:
294.         main()
295.     except KeyboardInterrupt:
296.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- добавить с помощью ASCII-графики за́мок, появляющийся в случайные моменты времени на песчаном дне;

- добавить крабов, движущихся по песчаному дну;
- сделать так, чтобы рыбки случайным образом ненадолго ускорялись.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `LONGEST_FISH_LENGTH = 10` в строке 51 заменить на `LONGEST_FISH_LENGTH = 50`?
2. Что будет, если `'right': fishType['right']` в строке 121 заменить на `'right': fishType['left']`?
3. Что будет, если `bext.fg('green')` в строке 249 заменить на `bext.fg('red')`?
4. Что будет, если удалить или закомментировать `clearAquarium()` в строке 92?
5. Что будет, если `bext.fg(fish['colors'][i])` в строке 245 заменить на `bext.fg('random')`?
6. Что будет, если `random.randint(10, 60)` в строке 161 заменить на `1`?

28

Заливка

«Заливка» — многоцветная игра, в которой игроку нужно заполнить игральную доску одним цветом, изменяя цвет элемента в верхнем левом углу, распространяющийся на всех соседние клетки того же исходного цвета. Эта игра аналогична игре для смартфонов Flood It!. У нашей программы есть и версия для страдающих дальтонизмом, в которой вместо цветных элементов используются различные фигуры. В основе ее работы лежит рекурсивный алгоритм заливки доски, аналогичный инструментам «банка с краской» и «заливка» во многих приложениях для рисования.

Программа в действии

На рис. 28.1 показано, как выглядит результат выполнения `flooder.py`.

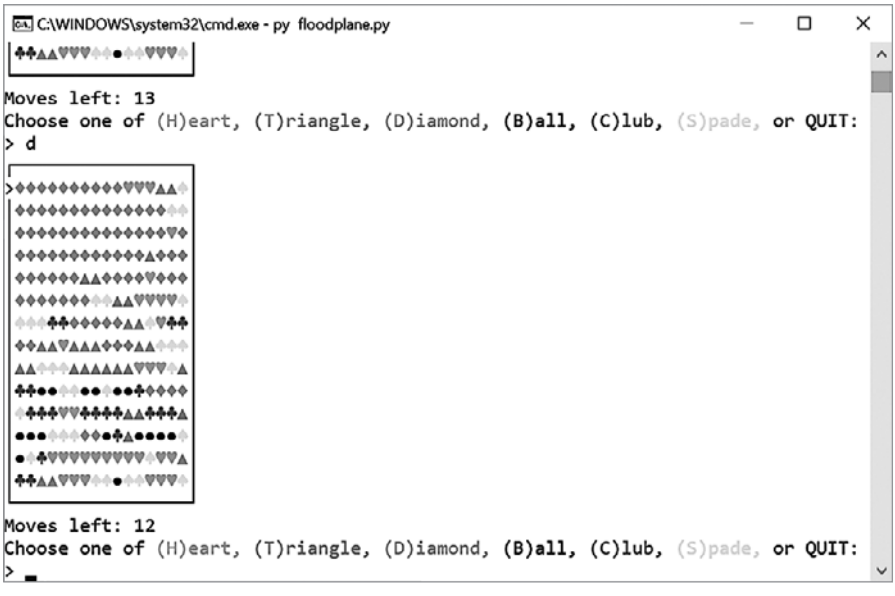


Рис. 28.1. Результаты работы игры «Заливка» в режиме для страдающих дальтонизмом, в котором вместо прямоугольников демонстрируются фигуры разной формы

Описание работы

Доступность для людей с ограниченными возможностями — серьезная проблема в компьютерных играх, решаемая различными путями. Например, при дальтонизме, в частности самой распространенной его форме — красно-зеленом дальтонизме, оттенки красного и зеленого кажутся человеку одинаковыми, вследствие чего ему сложно различать их на экране. Мы можем повысить доступность «Заливки» с помощью режима, в котором вместо различных цветов будут задействованы фигуры различной формы. Обратите внимание: даже в этом режиме используются цвета, так что можете вообще отказаться от «стандартного» режима, если хотите, и даже хорошо различающие цвета игроки смогут играть в режиме для дальтоников. Лучшие версии для людей с ограниченными возможностями — те, которые учитывают соображения доступности изначально, а не добавляют их позднее в качестве отдельного режима. Такой подход снижает требуемый объем кода и упрощает исправление ошибок в будущем.

В числе других проблем доступности — обеспечение размера шрифта, достаточного для людей с плохим зрением, визуальные подсказки для звуковых эффектов и субтитры к устной речи для слабослышащих, а также возможность переопределения элементов управления на другие клавиши, чтобы люди могли играть в игры с помощью только одной руки. На канале Game Maker's Toolkit на Youtube есть серия видеороликов Designing for Disability («Проектирование с учетом особых потребностей»), охватывающих многие аспекты проектирования игр с учетом доступности.

```
"""Заливка, (c) Эл Свейгарт al@inventwithpython.com
Многоцветная игра, в которой нужно заполнить игральную доску одним цветом.
Включает специальный режим для игроков с дальтонизмом.
По мотивам игры Flood It!
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, bext, игра"""

import random, sys

try:
    import bext
except ImportError:
    print('This program requires the bext module, which you')
    print('can install by following the instructions at')
    print('https://pypi.org/project/Bext/')
    sys.exit()

# Задаем константы:
BOARD_WIDTH = 16  # (!) Попробуйте заменить это значение на 4 или 40.
BOARD_HEIGHT = 14  # (!) Попробуйте заменить это значение на 4 или 20.
MOVES_PER_GAME = 20  # (!) Попробуйте заменить это значение на 3 или 300.

# Константы для различных фигур, используемых в режиме для дальтоников:
HEART     = chr(9829)  # Символ 9829 — '♥'.
DIAMOND   = chr(9830)  # Символ 9830 — '♦'.
SPADE     = chr(9824)  # Символ 9824 — '♠'.
CLUB      = chr(9827)  # Символ 9827 — '♣'.
BALL      = chr(9679)  # Символ 9679 — '●'.
TRIANGLE  = chr(9650)  # Символ 9650 — '▲'.

BLOCK     = chr(9608)  # Символ 9608 — '█'
LEFTRIGHT = chr(9472)  # Символ 9472 — '─'
UPDOWN    = chr(9474)  # Символ 9474 — '│'
DOWNRIGHT = chr(9484)  # Символ 9484 — '┌'
DOWNLEFT  = chr(9488)  # Символ 9488 — '┐'
UPRIGHT   = chr(9492)  # Символ 9492 — '└'
UPLEFT    = chr(9496)  # Символ 9496 — '┘'
# Список кодов chr() можно найти на https://inventwithpython.com/chr

# Все цвета/формы элементов, используемые на доске:
TILE_TYPES = (0, 1, 2, 3, 4, 5)
COLORS_MAP = {0: 'red', 1: 'green', 2:'blue',
```

```
43.               3:'yellow', 4:'cyan', 5:'purple'}
44. COLOR_MODE = 'color mode'
45. SHAPES_MAP = {0: HEART, 1: TRIANGLE, 2: DIAMOND,
46.               3: BALL, 4: CLUB, 5: SPADE}
47. SHAPE_MODE = 'shape mode'
48.
49.
50. def main():
51.     bext.bg('black')
52.     bext.fg('white')
53.     bext.clear()
54.     print('''Flooder, by Al Sweigart al@inventwithpython.com
55.
56. Set the upper left color/shape, which fills in all the
57. adjacent squares of that color/shape. Try to make the
58. entire board the same color/shape.''')
59.
60.     print('Do you want to play in colorblind mode? Y/N')
61.     response = input('> ')
62.     if response.upper().startswith('Y'):
63.         displayMode = SHAPE_MODE
64.     else:
65.         displayMode = COLOR_MODE
66.
67.     gameBoard = getNewBoard()
68.     movesLeft = MOVES_PER_GAME
69.
70.     while True:  # Основной цикл игры.
71.         displayBoard(gameBoard, displayMode)
72.
73.         print('Moves left:', movesLeft)
74.         playerMove = askForPlayerMove(displayMode)
75.         changeTile(playerMove, gameBoard, 0, 0)
76.         movesLeft -= 1
77.
78.         if hasWon(gameBoard):
79.             displayBoard(gameBoard, displayMode)
80.             print('You have won!')
81.             break
82.         elif movesLeft == 0:
83.             displayBoard(gameBoard, displayMode)
84.             print('You have run out of moves!')
85.             break
86.
87. def getNewBoard():
88.     """Возвращает ассоциативный массив с новой доской "Заливки"."""
89.
90.     # Роль ключей играют кортежи (x, y), а значений — клетки
91.     # на соответствующих позициях.
92.     board = {}
93.
94.     # Генерируем случайные цвета для игральной доски.
```

```
    for x in range(BOARD_WIDTH):
        for y in range(BOARD_HEIGHT):
            board[(x, y)] = random.choice(TILE_TYPES)

    # Делаем несколько элементов такими же, как соседний.
    # В результате получаются группы элементов одинакового цвета/формы.
    for i in range(BOARD_WIDTH * BOARD_HEIGHT):
        x = random.randint(0, BOARD_WIDTH - 2)
        y = random.randint(0, BOARD_HEIGHT - 1)
        board[(x + 1, y)] = board[(x, y)]
    return board


def displayBoard(board, displayMode):
    """Выводит игральную доску на экран."""
    bext.fg('white')
    # Отображает верхний край доски:
    print(DOWNRIGHT + (LEFTRIGHT * BOARD_WIDTH) + DOWNLEFT)

    # Выводим отдельные строки:
    for y in range(BOARD_HEIGHT):
        bext.fg('white')
        if y == 0:  # Первая строка начинается с '>'.
            print('>', end='')
        else:  # Последующие строки начинаются с белой вертикальной линии.
            print(UPDOWN, end='')

        # Выводим все клетки данной строки:
        for x in range(BOARD_WIDTH):
            bext.fg(COLORS_MAP[board[(x, y)]])
            if displayMode == COLOR_MODE:
                print(BLOCK, end='')
            elif displayMode == SHAPE_MODE:
                print(SHAPES_MAP[board[(x, y)]], end='')

        bext.fg('white')
        print(UPDOWN)  # Строки заканчиваются белой вертикальной линией.
    # Выводим нижний край игральной доски:
    print(UPRIGHT + (LEFTRIGHT * BOARD_WIDTH) + UPLEFT)


def askForPlayerMove(displayMode):
    """Даем возможность игроку выбрать цвет верхнего левого элемента."""
    while True:
        bext.fg('white')
        print('Choose one of ', end='')

        if displayMode == COLOR_MODE:
            bext.fg('red')
            print('(R)ed ', end='')
            bext.fg('green')
```

```
146.             print('(G)reen ', end='')
147.             bext.fg('blue')
148.             print('(B)lue ', end='')
149.             bext.fg('yellow')
150.             print('(Y)ellow ', end='')
151.             bext.fg('cyan')
152.             print('(C)yan ', end='')
153.             bext.fg('purple')
154.             print('(P)urple ', end='')
155.         elif displayMode == SHAPE_MODE:
156.             bext.fg('red')
157.             print('(H)eart, ', end='')
158.             bext.fg('green')
159.             print('(T)riangle, ', end='')
160.             bext.fg('blue')
161.             print('(D)iamond, ', end='')
162.             bext.fg('yellow')
163.             print('(B)all, ', end='')
164.             bext.fg('cyan')
165.             print('(C)lub, ', end='')
166.             bext.fg('purple')
167.             print('(S)pade, ', end='')
168.         bext.fg('white')
169.         print('or QUIT:')
170.         response = input('> ').upper()
171.         if response == 'QUIT':
172.             print('Thanks for playing!')
173.             sys.exit()
174.         if displayMode == COLOR_MODE and response in tuple('RGBYCP'):
175.             # Возвращаем номер типа элемента в зависимости от response:
176.             return {'R': 0, 'G': 1, 'B': 2,
177.                 'Y': 3, 'C': 4, 'P': 5}[response]
178.         if displayMode == SHAPE_MODE and response in tuple('HTDBCS'):
179.             # Возвращаем номер типа элемента в зависимости от response:
180.             return {'H': 0, 'T': 1, 'D':2,
181.                 'B': 3, 'C': 4, 'S': 5}[response]
182.
183.
184. def changeTile(tileType, board, x, y, charToChange=None):
185.     """Меняем цвет/форму клетки с помощью алгоритма рекурсивной
186.     заливки."""
187.     if x == 0 and y == 0:
188.         charToChange = board[(x, y)]
189.         if tileType == charToChange:
190.             return  # Простейший случай: тот же самый элемент.
191.
192.     board[(x, y)] = tileType
193.
194.     if x > 0 and board[(x - 1, y)] == charToChange:
195.         # Рекурсивно: меняем левый соседний элемент:
196.         changeTile(tileType, board, x - 1, y, charToChange)
```

```
197.     if y > 0 and board[(x, y - 1)] == charToChange:
198.         # Рекурсивно: меняем верхний соседний элемент:
199.         changeTile(tileType, board, x, y - 1, charToChange)
200.     if x < BOARD_WIDTH - 1 and board[(x + 1, y)] == charToChange:
201.         # Рекурсивно: меняем правый соседний элемент:
202.         changeTile(tileType, board, x + 1, y, charToChange)
203.     if y < BOARD_HEIGHT - 1 and board[(x, y + 1)] == charToChange:
204.         # Рекурсивно: меняем нижний соседний элемент:
205.         changeTile(tileType, board, x, y + 1, charToChange)
206.
207.
208. def hasWon(board):
209.     """Возвращает True, если вся доска — одного цвета/формы."""
210.     tile = board[(0, 0)]
211.
212.     for x in range(BOARD_WIDTH):
213.         for y in range(BOARD_HEIGHT):
214.             if board[(x, y)] != tile:
215.                 return False
216.     return True
217.
218.
219. # Если программа не импортируется, а запускается, выполняем запуск:
220. if __name__ == '__main__':
221.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (`!`). Можете также сами попробовать придумать, как сделать следующее:

- добавить дополнительные формы и цвета;
- создать другие формы игральной доски, помимо прямоугольной.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если `board = {}` в строке 92 замените на `board = []`?
2. Какое сообщение об ошибке вы получите, если `return board` в строке 105 замените на `return None`?
3. Что будет, если `movesLeft -= 1` в строке 76 заменить на `movesLeft -= 0`?

29

Моделирование лесного пожара

В данной программе мы моделируем лес, деревья в котором непрерывно растут, а затем сгорают. На каждом этапе моделирования существует равная 1 % вероятность того, что на пустом месте вырастет дерево, а также 1 % вероятности того, что в дерево попадет молния и оно сгорит. Пожары распространяются на непосредственно прилегающие к нему деревья, поэтому более густой лес скорее пострадает от сильного пожара, чем разреженный. Данная программа была создана под впечатлением от программы Emoji Sim Ники Кейс (Nicky Case), размещенной по адресу http://nca-se.me/simulating/model/.

Программа в действии

На рис. 29.1 показано, как выглядит результат выполнения `forestfiresim.py`.

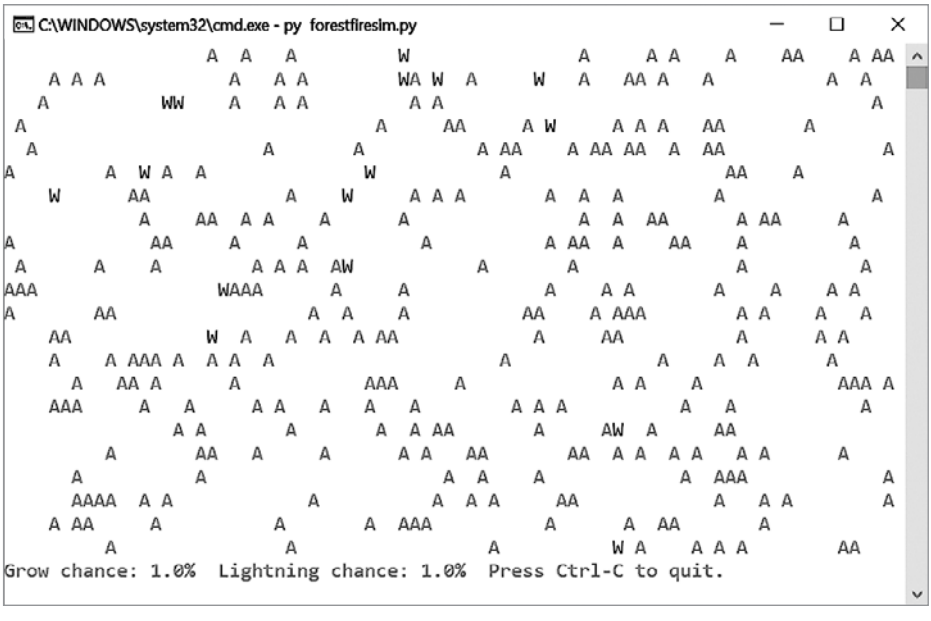


Рис. 29.1. Моделирование лесных пожаров, зеленые A обозначают деревья, а красные W — пожары

Описание работы

Эта программа — пример *эмерджентного поведения*, при котором взаимодействие между простыми составляющими системы приводит к сложным закономерностям. На пустых местах вырастают деревья, молнии их поджигают, а огонь превращает деревья обратно в пустое пространство, распространяясь при этом на соседние деревья. Подбирая различную скорость роста деревьев и частоту ударов молний, вы можете добиться различных явлений. Например, низкая вероятность удара молнии и высокая скорость роста приводят к обширным постоянным лесным пожарам, поскольку деревья располагаются кучно и быстро вырастают снова. Низкая же скорость роста деревьев при высокой вероятности ударов молний приводит к возникновению нескольких маленьких пожаров, быстро потухающих из-за отсутствия деревьев поблизости. Мы не программировали специально никаких подобных видов поведения, они возникают естественным образом в созданной нами системе.

```
 1. """Моделирование лесных пожаров, (c) Эл Свейгарт al@inventwithpython.com
 2. Имитационное моделирование распространения лесных пожаров. Нажмите Ctrl+C
 3. для останова.
 4. По мотивам программы Emoji Sim Ники Кейс http://ncase.me/simulating/model/
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, bext, имитационное моделирование"""
 7. import random, sys, time
 8.
 9. try:
10.     import bext
11. except ImportError:
12.     print('This program requires the bext module, which you')
13.     print('can install by following the instructions at')
14.     print('https://pypi.org/project/Bext/')
15.     sys.exit()
16.
17. # Задаем константы:
18. WIDTH = 79
19. HEIGHT = 22
20.
21. TREE = 'A'
22. FIRE = 'W'
23. EMPTY = ' '
24.
25. # (!) Попробуйте заменить эти параметры на что-либо в диапазоне от 0.0 до 1.0:
26. INITIAL_TREE_DENSITY = 0.20  # Начальное заполнение леса деревьями.
27. GROW_CHANCE = 0.01  # Вероятность превращения пустого места в дерево.
28. FIRE_CHANCE = 0.01  # Вероятность попадания в дерево молнии и его сгорания.
29.
30. # (!) Попробуйте задать длительность паузы равной 1.0 или 0.0:
31. PAUSE_LENGTH = 0.5
32.
33.
34. def main():
35.     forest = createNewForest()
36.     bext.clear()
37.
38.     while True:  # Основной цикл программы.
39.         displayForest(forest)
40.
41.         # Отдельный шаг моделирования:
42.         nextForest = {'width': forest['width'],
43.                       'height': forest['height']}
44.
45.         for x in range(forest['width']):
46.             for y in range(forest['height']):
47.                 if (x, y) in nextForest:
48.                     # Если значение nextForest[(x, y)] уже было задано
49.                     # на предыдущей итерации, ничего не делаем:
50.                     continue
51.
```

```
52.                 if ((forest[(x, y)] == EMPTY)
53.                     and (random.random() <= GROW_CHANCE)):
54.                     # Выращиваем дерево на данном пустом участке.
55.                     nextForest[(x, y)] = TREE
56.                 elif ((forest[(x, y)] == TREE)
57.                     and (random.random() <= FIRE_CHANCE)):
58.                     # Молния поджигает это дерево.
59.                     nextForest[(x, y)] = FIRE
60.                 elif forest[(x, y)] == FIRE:
61.                     # Это дерево горит.
62.                     # Проходим в цикле по всем соседним участкам:
63.                     for ix in range(-1, 2):
64.                         for iy in range(-1, 2):
65.                             # Огонь перекидывается на соседние деревья:
66.                             if forest.get((x + ix, y + iy)) == TREE:
67.                                 nextForest[(x + ix, y + iy)] = FIRE
68.                     # Дерево сгорело, стираем его:
69.                     nextForest[(x, y)] = EMPTY
70.                 else:
71.                     # Просто копируем существующий объект:
72.                     nextForest[(x, y)] = forest[(x, y)]
73.         forest = nextForest
74.
75.         time.sleep(PAUSE_LENGTH)
76.
77. def createNewForest():
78.     """Возвращает ассоциативный массив в качестве новой структуры данных
79.     для леса."""
80.     forest = {'width': WIDTH, 'height': HEIGHT}
81.     for x in range(WIDTH):
82.         for y in range(HEIGHT):
83.             if (random.random() * 100) <= INITIAL_TREE_DENSITY:
84.                 forest[(x, y)] = TREE  # Начинается с дерева.
85.             else:
86.                 forest[(x, y)] = EMPTY  # Начинается с пустого места.
87.     return forest
88.
89.
90. def displayForest(forest):
91.     """Отображает структуру данных для леса на экране."""
92.     bext.goto(0, 0)
93.     for y in range(forest['height']):
94.         for x in range(forest['width']):
95.             if forest[(x, y)] == TREE:
96.                 bext.fg('green')
97.                 print(TREE, end='')
98.             elif forest[(x, y)] == FIRE:
99.                 bext.fg('red')
100.                print(FIRE, end='')
101.            elif forest[(x, y)] == EMPTY:
102.                print(EMPTY, end='')
```

```
103.         print()
104.     bext.fg('reset')  # Используем цвет шрифта по умолчанию.
105.     print('Grow chance: {}%  '.format(GROW_CHANCE * 100), end='')
106.     print('Lightning chance: {}%  '.format(FIRE_CHANCE * 100), end='')
107.     print('Press Ctrl-C to quit.')
108.
109. # Если программа не импортируется, а запускается, выполняем запуск:
110. if __name__ == '__main__':
111.     try:
112.         main()
113.     except KeyboardInterrupt:
114.         sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем
115.                     # программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- добавлять случайным образом озера и реки, играющие роль противопожарных просек, которые огонь не может пересечь;
- добавить вероятность того, что огонь перекинется с дерева на соседнее;
- добавить разные типы деревьев с различными вероятностями возгорания;
- добавить различные состояния горения дерева, чтобы полное сгорание дерева занимало несколько шагов моделирования.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `bext.fg('green')` в строке 96 заменить на `bext.fg('random')`?
2. Что будет, если `EMPTY = ' '` в строке 23 заменить на `EMPTY = '.'`?
3. Что будет, если `forest.get((x + ix, y + iy)) == TREE` в строке 66 заменить на `forest.get((x + ix, y + iy)) == EMPTY`?
4. Что будет, если `nextForest[(x, y)] = EMPTY` в строке 69 заменить на `nextForest[(x, y)] = FIRE`?
5. Что будет, если `forest[(x, y)] = EMPTY` в строке 86 заменить на `forest[(x, y)] = TREE`?

30

Четыре в ряд

В этой классической настольной игре с отбрасыванием игровых элементов для двух игроков необходимо выстроить четыре элемента в ряд горизонтально, вертикально или по диагонали, не дав сопернику сделать то же самое. Данная игра аналогична игре Connect Four («Соедини четыре») или «крестики-нолики на бесконечном поле».

Программа в действии

Результат выполнения `fourinarow.py` выглядит следующим образом:

```
Four in a Row, by Al Sweigart al@inventwithpython.com
--сокращено--
     1234567
    +-------+
    |.......|
    |.......|
    |.......|
    |.......|
    |.......|
    |.......|
    +-------+
Player X, enter a column or QUIT:
> 3
```

```
     1234567
    +-------+
    |.......|
    |.......|
    |.......|
    |.......|
    |.......|
    |..X....|
    +-------+
Player O, enter a column or QUIT:
> 5
--сокращено--
Player O, enter a column or QUIT:
> 4

     1234567
    +-------+
    |.......|
    |.......|
    |XXX.XO.|
    |OOOOXO.|
    |OOOXOX.|
    |OXXXOXX|
    +-------+
Player O has won!
```

Описание работы

Структура программ в проектах настольных игр в этой книге похожа и обычно включает ассоциативный массив или список для хранения состояния доски, функцию `getNewBoard()`, возвращающую структуру данных для доски, функцию `displayBoard()` для визуализации структуры данных для доски на экране и т. д. Можете посмотреть на прочие проекты в данной книге с тегом *настольная игра* и сравнить их друг с другом, особенно если планируете сами писать свои программы для настольных игр.

```
  1. """Четыре в ряд, (c) Эл Свейгарт al@inventwithpython.com
  2. Игра с отбрасыванием игровых элементов, требующая выстроить
  3. четыре в ряд, аналогично игре Connect Four
  4. Код размещен на https://nostarch.com/big-book-small-python-projects
  5. Теги: большая, игра, настольная игра, для двух игроков"""
  6. import sys
  7.
  8. # Константы, необходимые для отображения доски:
  9. EMPTY_SPACE = '.'  # Точки считать проще, чем пробелы.
 10. PLAYER_X = 'X'
 11. PLAYER_O = 'O'
 12.
```

```
13. # Не забудьте поменять displayBoard() & COLUMN_LABELS, если меняете BOARD_WIDTH.
14. BOARD_WIDTH = 7
15. BOARD_HEIGHT = 6
16. COLUMN_LABELS = ('1', '2', '3', '4', '5', '6', '7')
17. assert len(COLUMN_LABELS) == BOARD_WIDTH
18.
19.
20. def main():
21.     print("""Four in a Row, by Al Sweigart al@inventwithpython.com
22.
23. Two players take turns dropping tiles into one of seven columns, trying
24. to make four in a row horizontally, vertically, or diagonally.
25. """)
26.
27.     # Начинаем новую игру:
28.     gameBoard = getNewBoard()
29.     playerTurn = PLAYER_X
30.
31.     while True:  # Очередь играть одного из игроков.
32.         # Отображаем доску и даем игроку возможность сделать ход:
33.         displayBoard(gameBoard)
34.         playerMove = askForPlayerMove(playerTurn, gameBoard)
35.         gameBoard[playerMove] = playerTurn
36.
37.         # Проверяем, выиграл ли игрок или не достигнута ли ничья:
38.         if isWinner(playerTurn, gameBoard):
39.             displayBoard(gameBoard)  # Отображаем доску один последний раз.
40.             print('Player ' + playerTurn + ' has won!')
41.             sys.exit()
42.         elif isFull(gameBoard):
43.             displayBoard(gameBoard)  # Отображаем доску один последний раз.
44.             print('There is a tie!')
45.             sys.exit()
46.
47.         # Переход хода к другому игроку:
48.         if playerTurn == PLAYER_X:
49.             playerTurn = PLAYER_O
50.         elif playerTurn == PLAYER_O:
51.             playerTurn = PLAYER_X
52.
53. def getNewBoard():
54.     """Возвращает ассоциативный массив, соответствующий игральной доске
55.     для Connect Four.
56.
57.     Ключами служат кортежи (columnIndex, rowIndex) из двух целых чисел,
58.     а значениями — одно из строковых значений 'X', 'O' или '.' (пустое
59.     место)."""
60.     board = {}
61.     for columnIndex in range(BOARD_WIDTH):
62.         for rowIndex in range(BOARD_HEIGHT):
63.             board[(columnIndex, rowIndex)] = EMPTY_SPACE
```

```
    return board

def displayBoard(board):
    """Отображает доску и все клетки на экране."""

    '''Подготовка списка для передачи в строковый метод format()
    для шаблона доски. В этом списке хранятся все клетки доски
    (и пустые участки) слева направо, сверху вниз:'''
    tileChars = []
    for rowIndex in range(BOARD_HEIGHT):
        for columnIndex in range(BOARD_WIDTH):
            tileChars.append(board[(columnIndex, rowIndex)])

    # Отображаем доску:
    print("""
     1234567
    +-------+
    |{}{}{}{}{}{}{}|
    |{}{}{}{}{}{}{}|
    |{}{}{}{}{}{}{}|
    |{}{}{}{}{}{}{}|
    |{}{}{}{}{}{}{}|
    |{}{}{}{}{}{}{}|
    +-------+""".format(*tileChars))


def askForPlayerMove(playerTile, board):
    """Даем возможность игроку выбрать столбец на доске
    для размещения элемента.
    Возвращает кортеж (столбец, строка), в который попадет элемент."""
    while True:  # Запрашиваем игрока, пока не будет сделан допустимый ход.
        print('Player {}, enter a column or QUIT:'.format(playerTile))
        response = input('> ').upper().strip()

        if response == 'QUIT':
            print('Thanks for playing!')
            sys.exit()

        if response not in COLUMN_LABELS:
            print('Enter a number from 1 to {}.'.format(BOARD_WIDTH))
            continue  # Снова спрашиваем у игрока ход.

        columnIndex = int(response) - 1  # -1 for 0-based the index.

        # Если столбец заполнен, снова запрашиваем ход:
        if board[(columnIndex, 0)] != EMPTY_SPACE:
            print('That column is full, select another one.')
            continue  # Снова спрашиваем у игрока ход.

        # Начиная снизу, ищем первый пустой участок.
        for rowIndex in range(BOARD_HEIGHT - 1, -1, -1):
```

```
            if board[(columnIndex, rowIndex)] == EMPTY_SPACE:
                return (columnIndex, rowIndex)


def isFull(board):
    """Возвращает True, если в `board` нет пустых участков,
    в противном случае возвращает False."""
    for rowIndex in range(BOARD_HEIGHT):
        for columnIndex in range(BOARD_WIDTH):
            if board[(columnIndex, rowIndex)] == EMPTY_SPACE:
                return False  # Нашли пустой участок и возвращаем False.
    return True  # Все участки заполнены.


def isWinner(playerTile, board):
    """Возвращает True, если `playerTile` содержит в одной строке
    на `board` четыре элемента в ряд, иначе возвращает False."""

    # Проходим по всей доске в поисках четырех в ряд:
    for columnIndex in range(BOARD_WIDTH - 3):
        for rowIndex in range(BOARD_HEIGHT):
            # Ищем горизонтальные четыре в ряд, двигаясь вправо:
            tile1 = board[(columnIndex, rowIndex)]
            tile2 = board[(columnIndex + 1, rowIndex)]
            tile3 = board[(columnIndex + 2, rowIndex)]
            tile4 = board[(columnIndex + 3, rowIndex)]
            if tile1 == tile2 == tile3 == tile4 == playerTile:
                return True

    for columnIndex in range(BOARD_WIDTH):
        for rowIndex in range(BOARD_HEIGHT - 3):
            # Ищем вертикальные четыре в ряд, двигаясь вниз:
            tile1 = board[(columnIndex, rowIndex)]
            tile2 = board[(columnIndex, rowIndex + 1)]
            tile3 = board[(columnIndex, rowIndex + 2)]
            tile4 = board[(columnIndex, rowIndex + 3)]
            if tile1 == tile2 == tile3 == tile4 == playerTile:
                return True

    for columnIndex in range(BOARD_WIDTH - 3):
        for rowIndex in range(BOARD_HEIGHT - 3):
            # Ищем четыре в ряд, двигаясь вправо вниз по диагонали:
            tile1 = board[(columnIndex, rowIndex)]
            tile2 = board[(columnIndex + 1, rowIndex + 1)]
            tile3 = board[(columnIndex + 2, rowIndex + 2)]
            tile4 = board[(columnIndex + 3, rowIndex + 3)]
            if tile1 == tile2 == tile3 == tile4 == playerTile:
                return True

            # Ищем четыре в ряд, двигаясь влево вниз по диагонали:
            tile1 = board[(columnIndex + 3, rowIndex)]
```

```
166.             tile2 = board[(columnIndex + 2, rowIndex + 1)]
167.             tile3 = board[(columnIndex + 1, rowIndex + 2)]
168.             tile4 = board[(columnIndex, rowIndex + 3)]
169.             if tile1 == tile2 == tile3 == tile4 == playerTile:
170.                 return True
171.     return False
172.
173.
174. # Если программа не импортируется, а запускается, производим запуск:
175. if __name__ == '__main__':
176.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- создать варианты «три в ряд» и «пять в ряд»;
- создать вариант данной игры для трех игроков;
- добавить «джокерный» элемент, случайным образом пропадающий после ходов всех игроков, который может использовать любой из игроков;
- добавить «заблокированные» элементы, не доступные для использования никому из игроков.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `PLAYER_O = 'O'` в строке 11 заменить на `PLAYER_O = 'X'`?
2. Что будет, если `return (columnIndex, rowIndex)` в строке 116 заменить на `return (columnIndex, 0)`?
3. Что будет, если `response == 'QUIT'` в строке 98 заменить на `response != 'QUIT'`?
4. Какое сообщение об ошибке вы получите, если `tileChars = []` в строке 72 замените на `tileChars = {}`?

31

Угадай число

«Угадай число» — классическая игра, на которой начинающие могут попрактиковаться в программировании. В этой игре компьютер загадывает случайное число от 1 до 100. У игрока есть 10 попыток угадать его. После каждой из попыток компьютер говорит игроку, больше или меньше загаданного это число.

Программа в действии

Результат выполнения `guess.py` выглядит следующим образом:

```
Guess the Number, by Al Sweigart al@inventwithpython.com

I am thinking of a number between 1 and 100.
You have 10 guesses left. Take a guess.
> 50
Your guess is too high.
You have 9 guesses left. Take a guess.
> 25
Your guess is too low.
--snip--
You have 5 guesses left. Take a guess.
> 42
Yay! You guessed my number!
```

Описание работы

В программе «Угадай число» используется несколько базовых идей программирования: циклы, операторы if-else, функции, вызовы методов и случайные числа. Модуль `random` языка Python позволяет генерировать псевдослучайные числа — числа, выглядящие случайными, но на самом деле таковыми не являющиеся. Генерировать псевдослучайные числа компьютеру проще, чем действительно случайные, и они считаются «достаточно случайными» для таких приложений, как компьютерные игры и некоторые научные модели.

Модуль `random` языка Python генерирует псевдослучайные числа, исходя из заданного начального значения, и все последовательности псевдослучайных чисел, сгенерированных на основе одного начального значения, будут одинаковы. Например, введите следующие команды в интерактивную командную оболочку:

```
>>> import random
>>> random.seed(42)
>>> random.randint(1, 10); random.randint(1, 10); random.randint(1, 10)
2
1
5
```

Если перезапустить командную оболочку и выполнить этот код снова, то будут сгенерированы те же псевдослучайные числа: `2`, `1`, `5`. Компьютерная игра Minecraft генерирует свои псевдослучайные виртуальные миры на основе начальных значений, поэтому различные игроки могут воссоздать один и тот же мир, воспользовавшись одним начальным значением.

```
 1. """Угадай число, (c) Эл Свейгарт al@inventwithpython.com
 2. Угадайте загаданное число по подсказкам.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, игра"""
 5.
 6. import random
 7.
 8.
 9. def askForGuess():
10.     while True:
11.         guess = input('> ')  # Введите свое предположение.
12.
13.         if guess.isdecimal():
14.             return int(guess)  # Преобразуем строковое представление
15.                                # предположения в число.
16.         print('Please enter a number between 1 and 100.')
17.
18. print('Guess the Number, by Al Sweigart al@inventwithpython.com')
19. print()
20. secretNumber = random.randint(1, 100)  # Выбираем случайное число.
```

```
21. print('I am thinking of a number between 1 and 100.')
22.
23. for i in range(10):  # У игрока есть 10 попыток.
24.     print('You have {} guesses left. Take a guess.'.format(10 - i))
25.
26.     guess = askForGuess()
27.     if guess == secretNumber:
28.         break  # Если число угадано — выходим из цикла.
29.
30.     # Даем подсказку:
31.     if guess < secretNumber:
32.         print('Your guess is too low.')
33.     if guess > secretNumber:
34.         print('Your guess is too high.')
35.
36. # Раскрываем результаты:
37. if guess == secretNumber:
38.     print('Yay! You guessed my number!')
39. else:
40.     print('Game over. The number I was thinking of was', secretNumber)
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создать игру «Угадай букву», подсказки в которой основаны на алфавитном порядке букв;
- сделать так, чтобы в подсказках после каждой попытки выводилось «теплее» или «холоднее», в зависимости от предыдущей попытки игрока.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `input('> ')` в строке 11 заменить на `input(secretNumber)`?
2. Какое сообщение об ошибке вы получите, если `return int(guess)` в строке 14 замените на `return guess`?
3. Что будет, если `random.randint(1, 100)` в строке 20 заменить на `random.randint(1, 1)`?
4. Что будет, если `format(10 - i)` в строке 24 заменить на `format(i)`?
5. Какое сообщение об ошибке вы получите, если `guess == secretNumber` в строке 37 замените на `guess = secretNumber`?

32

Простак

В этой короткой и простой программе вы изучите тайное тонкое искусство заинтриговать простака на многие часы. Я не стану раскрывать здесь соль шутки. Просто скопируйте код и запустите его. Данный проект замечательно подходит для начинающих.

Программа в действии

Результат выполнения `gullible.py` выглядит следующим образом:

```
Gullible, by Al Sweigart al@inventwithpython.com
Do you want to know how to keep a gullible person busy for hours? Y/N
> y
Do you want to know how to keep a gullible person busy for hours? Y/N
> y
Do you want to know how to keep a gullible person busy for hours? Y/N
> yes
Do you want to know how to keep a gullible person busy for hours? Y/N
> YES
Do you want to know how to keep a gullible person busy for hours? Y/N
> TELL ME HOW TO KEEP A GULLIBLE PERSON BUSY FOR HOURS
"TELL ME HOW TO KEEP A GULLIBLE PERSON BUSY FOR HOURS" is not a valid yes/no
response.
Do you want to know how to keep a gullible person busy for hours? Y/N
> y
```

```
Do you want to know how to keep a gullible person busy for hours? Y/N
> y
Do you want to know how to keep a gullible person busy for hours? Y/N
> n
Thank you. Have a nice day!
```

Описание работы

Для удобства использования ваши программы должны стремиться интерпретировать целый спектр данных, которые может ввести пользователь. Например, данная программа задает пользователю вопрос, предполагающий ответ типа «да/нет», но игроку будет проще ввести `y` или `n` вместо полных слов. Программа также понимает, что хочет пользователь, если нажата клавиша Caps Lock, поскольку вызывает строковый метод `lower()` для введенного игроком строкового значения. Таким образом, `'y'`, `'yes'`, `'Y'`, `'Yes'` и `'YES'` интерпретируются программой одинаково. То же самое относится и к отрицательному ответу пользователя.

```
 1. """Простак, (c) Эл Свейгарт al@inventwithpython.com
 2. Как заинтриговать простака на многие часы (программа-шутка).
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, юмор"""
 5.
 6. print('Gullible, by Al Sweigart al@inventwithpython.com')
 7.
 8. while True:  # Основной цикл программы.
 9.     print('Do you want to know how to keep a gullible person busy for
        hours? Y/N')
10.     response = input('> ')  # Получаем ответ пользователя.
11.     if response.lower() == 'no' or response.lower() == 'n':
12.         break  # В случае "no" выходим из цикла.
13.     if response.lower() == 'yes' or response.lower() == 'y':
14.         continue  # В случае "yes" продолжаем с начала цикла.
15.     print('"{}" is not a valid yes/no response.'.format(response))
16.
17. print('Thank you. Have a nice day!')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `response.lower() == 'no'` в строке 11 заменить на `response.lower() != 'no'`?
2. Что будет, если `while True:` в строке 8 заменить на `while False:`?

33

Мини-игра со взломом

В этой игре игрок должен взломать компьютер, угадав слово из семи букв, используемое в качестве секретного пароля. Блоки памяти компьютера выдают возможные слова, а игрок получает подсказки относительно того, насколько близки его догадки к правильному паролю. Например, если секретный пароль — MONITOR, а игрок ввел CONTAIN, подсказка будет содержать информацию о совпадении двух из семи букв, поскольку и MONITOR, и CONTAIN содержат буквы O и N — вторую и третью соответственно. Эта игра напоминает проект 1 и игру с взломом в серии компьютерных игр Fallout.

Программа в действии

Результат выполнения `hacking.py` выглядит следующим образом:

```
Hacking Minigame, by Al Sweigart al@inventwithpython.com
Find the password in the computer's memory:
0x1150  $],>@|~~RESOLVE^    0x1250  {>+)<!?CHICKEN,%
0x1160  }@%_-:;/$^(|<|!(    0x1260  .][})?#@#ADDRESS
0x1170  _;)][#?<&~$~+&}}    0x1270  ,#=)>{-;/DESPITE
0x1180  %[!]{REFUGEE@?~,    0x1280  }/.}!-DISPLAY%%/
0x1190  _[^%[@}^<_+{_@$~    0x1290  =>>,:*%?_?@+{%#.
0x11a0  )?~/)+PENAltY?-=    0x12a0  >[,?*#IMPROVE@$/
--сокращено--
```

```
Enter password: (4 tries remaining)
> resolve
Access Denied (2/7 correct)
Enter password: (3 tries remaining)
> improve
A C C E S S   G R A N T E D
```

Описание работы

В названии этой игры упоминается взлом, но никакого настоящего взлома не происходит. Если бы мы просто перечислили на экране возможные слова, то игровой процесс остался бы таким же. Однако косметические дополнения, имитирующие банки памяти компьютера, придают восхитительное ощущение взлома компьютера. Внимание к нюансам и впечатлению пользователя от работы с программой превращает простую и скучную игру в по-настоящему увлекательную.

```
"""Мини-игра со взломом, (c) Эл Свейгарт al@inventwithpython.com
Мини-игра со взломом из Fallout 3. Найдите семибуквенное слово-пароль
с помощью подсказок, возвращаемых при каждой попытке угадать его.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, графика, игра, головоломка"""

# Примечание: для этой программы необходим файл sevenletterwords.txt.
# Скачать его можно на https://inventwithpython.com/sevenletterwords.txt

import random, sys

# Задаем константы:
# Заполнитель мусорными символами для "памяти компьютера".
GARBAGE_CHARS = '~!@#$%^&*()_+-={}[]|;:,.<>?/'

# Загружаем список WORDS из текстового файла с семибуквенными словами.
with open('sevenletterwords.txt') as wordListFile:
    WORDS = wordListFile.readlines()
for i in range(len(WORDS)):
    # Преобразуем каждое слово в верхний регистр и удаляем символ новой строки
    # в конце:

    WORDS[i] = WORDS[i].strip().upper()
def main():
    """Запуск одной игры со взломом."""
    print('''Hacking Minigame, by Al Sweigart al@inventwithpython.com
Find the password in the computer's memory. You are given clues after
each guess. For example, if the secret password is MONITOR but the
player guessed CONTAIN, they are given the hint that 2 out of 7 letters
were correct, because both MONITOR and CONTAIN have the letter O and N
as their 2nd and 3rd letter. You get four guesses.\n''')
    input('Press Enter to begin...')

```

```
34.     gameWords = getWords()
35.     # "Память компьютера" — только для вида, но выглядит круто:
36.     computerMemory = getComputerMemoryString(gameWords)
37.     secretPassword = random.choice(gameWords)
38.
39.     print(computerMemory)
40.     # Начинаем с четырех оставшихся попыток и постепенно уменьшаем
41      # их количество:
42.     for triesRemaining in range(4, 0, -1):
43.         playerMove = askForPlayerGuess(gameWords, triesRemaining)
44.         if playerMove == secretPassword:
45.             print('A C C E S S   G R A N T E D')
46.             return
47.         else:
48.             numMatches = numMatchingLetters(secretPassword, playerMove)
49.             print('Access Denied ({}/7 correct)'.format(numMatches))
50.     print('Out of tries. Secret password was {}.'.format(secretPassword))
51.
52. def getWords():
53.     """Возвращает список из 12 слов — возможных паролей.
54.
55.     Секретный пароль будет первым словом в списке.
56.     Ради честной игры мы попытаемся гарантировать наличие слов
57.     с различным количеством совпадающих с паролем букв."""
58.     secretPassword = random.choice(WORDS)
59.     words = [secretPassword]
60.
61.     # Находим еще два слова; количество совпадающих  букв — 0.
62.     # "< 3" потому, что секретный пароль уже входит в список слов.
63.     while len(words) < 3:
64.         randomWord = getOneWordExcept(words)
65.         if numMatchingLetters(secretPassword, randomWord) == 0:
66.             words.append(randomWord)
67.
68.     # Находим два слова, у которых совпадает три буквы
69.     # (но прекращаем поиск после 500 попыток, если не удалось найти).
70.     for i in range(500):
71.         if len(words) == 5:
72.             break  # Нашли пять слов, так что выходим из цикла.
73.
74.         randomWord = getOneWordExcept(words)
75.         if numMatchingLetters(secretPassword, randomWord) == 3:
76.             words.append(randomWord)
77.
78.     # Находим по крайней мере семь слов, у которых совпадает хотя бы одна
79.     # буква (но прекращаем поиск после 500 попыток, если не удалось найти).
80.     for i in range(500):
81.         if len(words) == 12:
82.             break  # Нашли 7 или более слов, так что выходим из цикла.
83.
84.         randomWord = getOneWordExcept(words)
```

```
        if numMatchingLetters(secretPassword, randomWord) != 0:
            words.append(randomWord)

    # Добавляем любые случайные слова, чтобы всего их было 12.
    while len(words) < 12:
        randomWord = getOneWordExcept(words)
        words.append(randomWord)

    assert len(words) == 12
    return words


def getOneWordExcept(blocklist=None):
    """Возвращает случайное слово из списка WORDS, не входящее в blocklist."""
    if blocklist == None:
        blocklist = []

    while True:
        randomWord = random.choice(WORDS)
        if randomWord not in blocklist:
            return randomWord


def numMatchingLetters(word1, word2):
    """Возвращает число совпадающих букв в указанных двух словах."""
    matches = 0
    for i in range(len(word1)):
        if word1[i] == word2[i]:
            matches += 1
    return matches


def getComputerMemoryString(words):
    """Возвращает строковое значение, соответствующее "памяти компьютера"."""

    # Выбираем по одной содержащей слово строке. Всего строк 16,
    # но они разбиты на две половины.
    linesWithWords = random.sample(range(16 * 2), len(words))
    # Начальный адрес памяти (также только для вида).
    memoryAddress = 16 * random.randint(0, 4000)
    # Создаем строковое значение для "памяти компьютера".
    computerMemory = []  # Будет включать 16 строковых значений, по одному
                         # на строку.
    nextWord = 0  # Индекс в WORDS слова, помещаемого в строку.
    for lineNum in range(16):  # "Память компьютера" содержит 16 строк.
        # Создаем половину строки мусорных символов:
        leftHalf = ''
        rightHalf = ''
        for j in range(16):  # Каждая половина содержит 16 символов.
            leftHalf += random.choice(GARBAGE_CHARS)
            rightHalf += random.choice(GARBAGE_CHARS)
```

```
136.
137.         # Заполняем пароль из WORDS:
138.         if lineNum in linesWithWords:
139.             # Находим случайное место для вставки слова в половине строки:
140.             insertionIndex = random.randint(0, 9)
141.             # Вставляем слово:
142.             leftHalf = (leftHalf[:insertionIndex] + words[nextWord]
143.                 + leftHalf[insertionIndex + 7:])
144.             nextWord += 1  # Обновляем слово для вставки в половину строки.
145.         if lineNum + 16 in linesWithWords:
146.             # Находим случайное место в половине строки для вставки слова:
147.             insertionIndex = random.randint(0, 9)
148.             # Вставляем слово:
149.             rightHalf = (rightHalf[:insertionIndex] + words[nextWord]
150.                 + rightHalf[insertionIndex + 7:])
151.             nextWord += 1  # Обновляем слово для вставки в половину строки.
152.
153.         computerMemory.append('0x' + hex(memoryAddress)[2:].zfill(4)
154.                     + '  ' + leftHalf + '    '
155.                     + '0x' + hex(memoryAddress + (16*16))[2:].zfill(4)
156.                     + '  ' + rightHalf)
157.
158.         memoryAddress += 16  # Перескакиваем, скажем, с 0xe680 на 0xe690.
159.
160.     # Все строковые значения списка computerMemory для возвращения
161.     # объединяются в одно большое строковое значение:
162.     return '\n'.join(computerMemory)
163.
164.
165. def askForPlayerGuess(words, tries):
166.     """Ввод пользователем догадки."""
167.     while True:
168.         print('Enter password: ({} tries remaining)'.format(tries))
169.         guess = input('> ').upper()
170.         if guess in words:
171.             return guess
172.         print('That is not one of the possible passwords listed above.')
173.         print('Try entering "{}" or "{}".'.format(words[0], words[1]))
174.
175.
175. # Если программа не импортируется, а запускается, производим запуск:
177. if __name__ == '__main__':
178.     try:
179.         main()
180.     except KeyboardInterrupt:
181.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- найдите список слов в интернете и создайте собственный файл `sevenletterwords.txt`, возможно, с шести- или восьмибуквенными словами;
- создайте другую визуализацию «памяти компьютера».

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `for j in range(16):` в строке 133 заменить на `for j in range(0):`?
2. Что будет, если `GARBAGE_CHARS = '~!@#$%^&*()_+-={}[]|;:,.<>?/'` в строке 14 заменить на `GARBAGE_CHARS = '.'`?
3. Что будет, если `gameWords = getWords()` в строке 34 заменить на `gameWords = ['MALKOVICH'] * 20`?
4. Какое сообщение об ошибке вы получите, если `return words` в строке 94 замените на `return`?
5. Что будет, если `randomWord = random.choice(WORDS)` в строке 103 заменить на `secretPassword = 'PASSWORD'`?

34

«Виселица» и «Гильотина»

В этой классической игре со словами необходимо угадывать буквы в загаданном слове. За каждую неправильную букву рисуется дополнительная часть виселицы. Попробуйте угадать слово целиком, прежде чем будет нарисована вся виселица. В роли загаданных слов в этой версии — названия животных, например RABBIT и PIGEON, но вы можете заменить их собственным набором слов.

В переменной `HANGMAN_PICS` содержатся строковые значения для ASCII-графики, отражающей каждый шаг рисования виселицы:

```
+--+     +--+     +--+     +--+     +--+     +--+     +--+
 |  |     |  |     |  |     |  |     |  |     |  |     |  |
    |     O  |     O  |     O  |     O  |     O  |     O  |
    |        |     |  |    /|  |    /|\ |    /|\ |    /|\ |
    |        |        |     |  |     |  |    /   |    / \ |
    |        |        |     |  |     |  |        |        |
=====    =====    =====    =====    =====    =====    =====
```

Чтобы придать игре французский колорит, можете заменить строковые значения в переменной `HANGMAN_PICS` на следующие рисунки гильотины:

```
|        |   |    |===|    |===|    |===|    |===|    |===|
|        |   |    |   |    |   |    |   |    || /|    || /|
|        |   |    |   |    |   |    |   |    ||/ |    ||/ |
|        |   |    |   |    |   |    |   |    |    |    |    |
|        |   |    |   |    |   |    |   |    |    |    |    |
|        |   |    |   |    |   |    |/-\|    |/-\|    |/-\|
|        |   |    |   |    |\ /|    |\ /|    |\ /|    |\O/|
|===     |===|    |===|    |===|    |===|    |===|    |===|
```

Программа в действии

Результат выполнения `hangman.py` выглядит следующим образом:

```
Hangman, by Al Sweigart al@inventwithpython.com

 +--+
 |  |
    |
    |
    |
    |
=====
The category is: Animals

Missed letters: No missed letters yet.


_ _ _ _ _
Guess a letter.
> e
--сокращено--
 +--+
 |  |
 O  |
/|  |
    |
    |
=====
The category is: Animals

Missed letters: A I S
O T T E _
Guess a letter.
> r
Yes! The secret word is: OTTER
You have won!
```

Описание работы

Принцип игры у «Виселицы» и «Гильотины» один, различны только внешние представления, что позволяет с легкостью заменить ASCII-графику виселицы на ASCII-графику гильотины без изменения основной логики программы. Подобное разделение представления и логики в программе упрощает добавление новых возможностей и перепроектирование. В коммерческой разработке программного обеспечения эта стратегия служит примером одного из *архитектурных паттернов проектирования*, основная задача которого — структурирование программ в целях большей понятности и упрощения модификации. В основном эти принципы используются в крупных программных продуктах, но применимы и к более мелким.

```
 1. """Виселица, (с) Эл Свейгарт al@inventwithpython.com
 2. Угадайте буквы загаданного слова, пока не будет нарисована виселица.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: большая, игра, слова, головоломка"""
 5.
 6. # Одна из версий этой игры приведена в книге Invent Your Own
 7. # Computer Games with Python на https://nostarch.com/inventwithpython
 8.
 9. import random, sys
10.
11. # Задаем константы:
12. # (!) Попробуйте добавить в HANGMAN_PICS новые строковые значения
13. # или изменить существующие, чтобы рисовать гильотину вместо виселицы.
14. HANGMAN_PICS = [r"""
15.  +--+
16.  |  |
17.     |
18.     |
19.     |
20.     |
21. =====""",
22. r"""
23.  +--+
24.  |  |
25.  O  |
26.     |
27.     |
28.     |
29. =====""",
30. r"""
31.  +--+
32.  |  |
33.  O  |
34.  |  |
35.     |
36.     |
37. =====""",
38. r"""
39.  +--+
40.  |  |
41.  O  |
42. /|  |
43.     |
44.     |
45. =====""",
46. r"""
47.  +--+
48.  |  |
49.  O  |
50. /|\ |
51.     |
```

```
    |
=====""",
r"""
 +--+
 |  |
 O  |
/|\ |
/   |
    |
=====""",
r"""
 +--+
 |  |
 O  |
/|\ |
/ \ |
    |
=====""]

# (!) Попробуйте заменить константы CATEGORY и WORDS на другие строковые
# значения.
CATEGORY = 'Animals'
WORDS = 'ANT BABOON BADGER BAT BEAR BEAVER CAMEL CAT CLAM COBRA COUGAR
➥ COYOTE CROW DEER DOG DONKEY DUCK EAGLE FERRET FOX FROG GOAT GOOSE HAWK
➥ LION LIZARD LLAMA MOLE MONKEY MOOSE MOUSE MULE NEWT OTTER OWL PANDA PARROT
➥ PIGEON PYTHON RABBIT RAM RAT RAVEN RHINO SALMON SEAL SHARK SHEEP SKUNK
➥ SLOTH SNAKE SPIDER STORK SWAN TIGER TOAD TROUT TURKEY TURTLE WEASEL WHALE
➥ WOLF WOMBAT ZEBRA'.split()

def main():
    print('Hangman, by Al Sweigart al@inventwithpython.com')

    # Переменные для новой игры:
    missedLetters = []  # Список неправильных попыток угадать буквы.
    correctLetters = []  # Список правильных попыток угадать буквы.
    secretWord = random.choice(WORDS)  # Загаданное слово.

    while True:  # Основной цикл игры.
        drawHangman(missedLetters, correctLetters, secretWord)

        # Пусть пользователь введет свою букву:
        guess = getPlayerGuess(missedLetters + correctLetters)

        if guess in secretWord:
            # Добавляем правильную догадку в correctLetters:
            correctLetters.append(guess)
            # Проверяем, не выиграл ли игрок:
            foundAllLetters = True  # Начинаем с предположения,
                                    # что он выиграл.
            for secretWordLetter in secretWord:
                if secretWordLetter not in correctLetters:
```

```
 98.                     # В загаданном слове есть буква, пока еще отсутствующая
 99.                     # в correctLetters, так что игрок пока что не выиграл:
100.                     foundAllLetters = False
101.                     break
102.             if foundAllLetters:
103.                 print('Yes! The secret word is:', secretWord)
104.                 print('You have won!')
105.                 break  # Выходим из основного цикла игры.
106.         else:
107.             # Игрок не угадал:
108.             missedLetters.append(guess)
109.
110.             # Проверяем, не превысил ли игрок допустимое количество попыток
111.             # и проиграл. ("- 1", поскольку пустая виселица в
112.             # HANGMAN_PICS не считается)
113.             if len(missedLetters) == len(HANGMAN_PICS) - 1:
114.                 drawHangman(missedLetters, correctLetters, secretWord)
115.                 print('You have run out of guesses!')
116.                 print('The word was "{}"'.format(secretWord))
117.                 break
118.
119.
120. def drawHangman(missedLetters, correctLetters, secretWord):
121.     """Рисуем текущее состояние виселицы вместе с неугаданными
122.     и правильно угаданными буквами загаданного слова."""
123.     print(HANGMAN_PICS[len(missedLetters)])
124.     print('The category is:', CATEGORY)
125.     print()
126.
127.     # Отображаем неправильные попытки угадать букву:
128.     print('Missed letters: ', end='')
129.     for letter in missedLetters:
130.         print(letter, end=' ')
131.     if len(missedLetters) == 0:
132.         print('No missed letters yet.')
133.     print()
134.
135.     # Отображаем пропуски вместо загаданного слова (по одному на букву):
136.     blanks = ['_'] * len(secretWord)
137.
138.     # Заменяем пропуски на угаданные буквы:
139.     for i in range(len(secretWord)):
140.         if secretWord[i] in correctLetters:
141.             blanks[i] = secretWord[i]
142.
143.     # Отображаем загаданное слово с пропусками между буквами:
144.     print(' '.join(blanks))
145.
146.
147. def getPlayerGuess(alreadyGuessed):
148.     """Возвращает введенную пользователем букву. Убеждается,
149.     что пользователь ввел одну букву, которую не вводил ранее."""
```

```
150.     while True:  # Запрашиваем, пока пользователь не введет допустимую букву.
151.         print('Guess a letter.')
152.         guess = input('> ').upper()
153.         if len(guess) != 1:
154.             print('Please enter a single letter.')
155.         elif guess in alreadyGuessed:
156.             print('You have already guessed that letter. Choose again.')
157.         elif not guess.isalpha():
158.             print('Please enter a LETTER.')
159.         else:
160.             return guess
161.
162.
163. # Если программа не импортируется, а запускается, производим запуск:
164. if __name__ == '__main__':
165.     try:
166.         main()
167.     except KeyboardInterrupt:
168.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (`!`). Можете также сами попробовать придумать, как сделать следующее:

- добавить возможность выбора категории и позволить пользователю выбирать, слова из какой категории он хочет угадывать;
- добавить пользователю возможность выбора версии программы с виселицей или гильотиной.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `missedLetters.append(guess)` в строке 108?
2. Что будет, если `drawHangman(missedLetters, correctLetters, secretWord)` в строке 85 заменить на `drawHangman(correctLetters, missedLetters, secretWord)`?
3. Что будет, если `['_']` в строке 136 заменить на `['*']`?
4. Что будет, если выражение `print(' '.join(blanks))` в строке 144 заменить на `print(secretWord)`?

35

Гексагональная сетка

Эта короткая программа генерирует мозаичное изображение гексагональной сетки, напоминающей мелкоячеистую проволочную сетку, и демонстрирует, что для создания интересных изображений не нужно много кода. Несколько более сложный вариант данной программы приведен в проекте 65.

Обратите внимание: в этой программе используются неформатированные строки, в которых открывающая кавычка предваряется буквой `r` в нижнем регистре, так что обратные косые черты не интерпретируются как символы экранирования.

Программа в действии

На рис. 35.1 показано, как выглядит результат выполнения `shiningcarpet.py`.

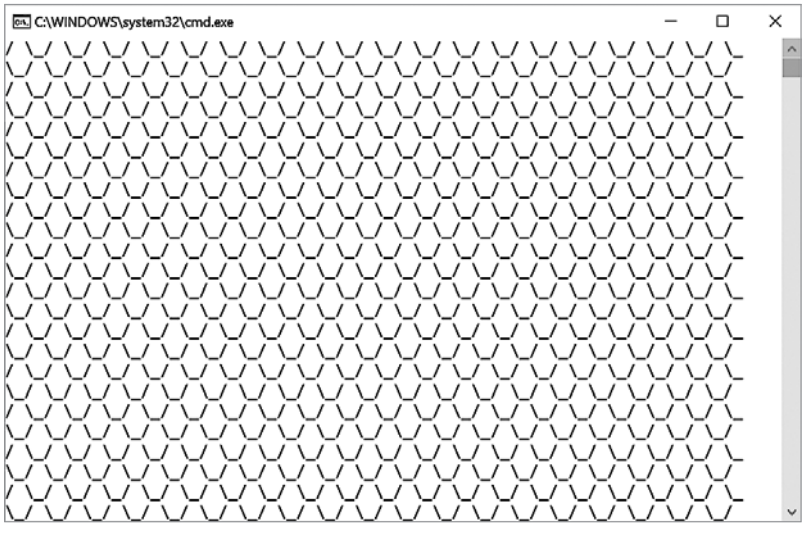


Рис. 35.1. Выведенное мозаичное изображение гексагональной сетки

Описание работы

Эффективность программирования заключается в возможности быстро и безошибочно выполнять повторяющиеся команды. Благодаря этому с помощью десятка строк кода можно создавать на экране сотни, тысячи или даже миллионы шестиугольников.

В командной строке или в окне терминала можно перенаправить выводимые программой данные с экрана в текстовый файл. В Windows для создания текстового файла с шестиугольниками выполните команду `py hexgrid.py > hextiles.txt`. В Linux и macOS выполните команду `python3 hexgrid.py > hextiles.txt`. А поскольку накладываемое размером экрана ограничение снято, можете увеличить значения констант `X_REPEAT` и `Y_REPEAT` и сохранить полученное в файл, который затем можно будет легко распечатать на бумаге, отправить по электронной почте или разместить в соцсети. То же самое относится к любому сгенерированному с помощью компьютера графическому изображению.

```
1. """Гексагональная сетка, (c) Эл Свейгарт al@inventwithpython.com
2. Выводит на экран мозаику в виде гексагональной сетки
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: крошечная, для начинающих, графика"""
```

```
 5.
 6. # Задаем константы:
 7. # (!) Попробуйте изменить эти значения:
 8. X_REPEAT = 19  # Количество ячеек по горизонтали.
 9. Y_REPEAT = 12  # Количество ячеек по вертикали.
10.
11. for y in range(Y_REPEAT):
12.     # Выводим верхнюю половину шестиугольника:
13.     for x in range(X_REPEAT):
14.         print(r'/ \_', end='')
15.     print()
16.
17.     # Выводим нижнюю половину шестиугольника:
18.     for x in range(X_REPEAT):
19.         print(r'\_/ ', end='')
20.     print()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- увеличить размер шестиугольников;
- выполнять замощение прямоугольниками вместо шестиугольников.

Попробуйте поэкспериментировать и переделать эту программу так, чтобы создавалась более крупная шестиугольная сетка наподобие следующего узора:

```
 /   \     /   \     /   \     /   \     /   \     /   \     /   \     /   \
/     \___/     \___/     \___/     \___/     \___/     \___/     \___/     \
\     /   \     /   \     /   \     /   \     /   \     /   \     /   \     /
 \___/     \___/     \___/     \___/     \___/     \___/     \___/     \___/
 /   \     /   \     /   \     /   \     /   \     /   \     /   \     /   \
/     \___/     \___/     \___/     \___/     \___/     \___/     \___/     \
  /     \         /     \         /     \         /     \         /     \
 /       \       /       \       /       \       /       \       /       \
/         \_____/         \_____/         \_____/         \_____/         \_____
\         /     \         /     \         /     \         /     \         /
 \       /       \       /       \       /       \       /       \       /
  \_____/         \_____/         \_____/         \_____/         \_____/
  /     \         /     \         /     \         /     \         /     \
 /       \       /       \       /       \       /       \       /       \
/         \_____/         \_____/         \_____/         \_____/         \_____
```

Исследование программы

Программа простая, так что вариантов ее модификации не так уж много. Попробуйте вместо этого подумать, как можно запрограммировать генерацию узоров других форм.

36

Песочные часы

Эта программа для визуализации включает приближенный физический движок для моделирования падения песка через небольшое отверстие песочных часов. Песок скапливается горкой в нижней части песочных часов; а в конце песочные часы переворачиваются и процесс повторяется.

Программа в действии

На рис. 36.1 показано, как выглядит результат выполнения `hourglass.py`.

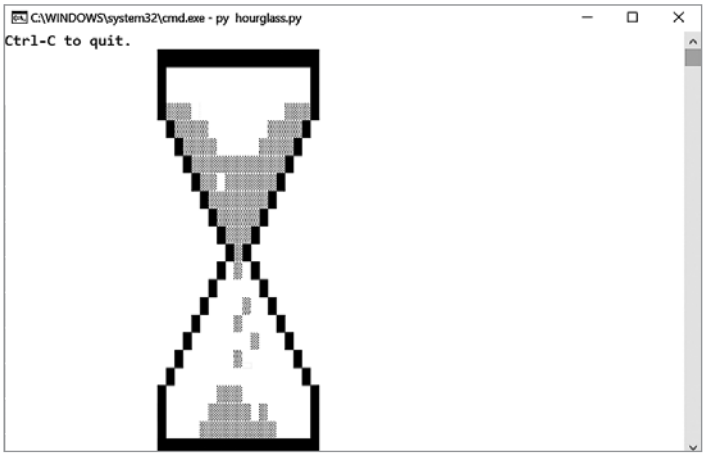


Рис. 36.1. Результаты работы программы «Песочные часы»

Описание работы

Программа «Песочные часы» реализует примитивный *физический движок* (physics engine). Это программное обеспечение для имитационного моделирования падения физических объектов под влиянием силы тяжести, их столкновения друг с другом и вообще движения в соответствии с законами физики. Физические движки применяются в компьютерных играх, при создании динамических изображений и имитационном моделировании, имеющем научные цели. В строках с 91-й по 102-ю каждая «песчинка» проверяет, есть ли внизу незанятое место, и движется вниз, если есть. В противном случае она проверяет, есть ли пространство для движения вниз и влево (строки с 104-й по 112-ю) или вниз и вправо (строки с 114-й по 122-ю). Конечно, *кинематика* — раздел классической физики, занимающийся движением макроскопических объектов, — намного обширнее. Однако простое моделирование движения песка в песочных часах, на которое было бы приятно смотреть, не требует научной степени по физике.

```
 1. """Песочные часы, (c) Эл Свейгарт al@inventwithpython.com
 2. Динамическое изображение песочных часов. Нажмите Ctrl+C для останова.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: большая, графика, bext, имитационное моделирование"""
 5.
 6. import random, sys, time
 7.
 8. try:
 9.     import bext
10. except ImportError:
11.     print('This program requires the bext module, which you')
12.     print('can install by following the instructions at')
13.     print('https://pypi.org/project/Bext/')
14.     sys.exit()
15.
16. # Задаем константы:
17. PAUSE_LENGTH = 0.2  # (!) Попробуйте заменить это значение на 0.0 или 1.0.
18. # (!) Попробуйте заменить это значение на любое число от 0 до 100:
19. WIDE_FALL_CHANCE = 50
20.
21. SCREEN_WIDTH = 79
22. SCREEN_HEIGHT = 25
23. X = 0  # Индекс значений X в кортеже (x, y) – 0.
24. Y = 1  # Индекс значений Y в кортеже (x, y) – 1.
25. SAND = chr(9617)
26. WALL = chr(9608)
27. # Описываем стенки песочных часов:
28. HOURGLASS = set()  # Кортежи (x, y) для стенок песочных часов.
29. # (!) Попробуйте закомментировать часть строк HOURGLASS.add(), чтобы стереть
30. # стенки песочных часов:
31. for i in range(18, 37):
32.     HOURGLASS.add((i, 1))   # Верхняя крышка песочных часов.
```

```
    HOURGLASS.add((i, 23))  # Нижняя крышка.
for i in range(1, 5):
    HOURGLASS.add((18, i))  # Верхняя левая прямая стенка.
    HOURGLASS.add((36, i))  # Верхняя правая прямая стенка.
    HOURGLASS.add((18, i + 19))  # Стенка внизу слева.
    HOURGLASS.add((36, i + 19))  # Стенка внизу справа.
for i in range(8):
    HOURGLASS.add((19 + i, 5 + i))   # Верхняя левая наклонная стенка.
    HOURGLASS.add((35 - i, 5 + i))   # Верхняя правая наклонная стенка.
    HOURGLASS.add((25 - i, 13 + i))  # Нижняя левая наклонная стенка.
    HOURGLASS.add((29 + i, 13 + i))  # Нижняя правая наклонная стенка.

# Начальная горка песка в верхней половине песочных часов:
INITIAL_SAND = set()
for y in range(8):
    for x in range(19 + y, 36 - y):
        INITIAL_SAND.add((x, y + 4))


def main():
    bext.fg('yellow')
    bext.clear()

    # Выводим сообщение о возможности выхода:
    bext.goto(0, 0)
    print('Ctrl-C to quit.', end='')

    # Отображаем стенки песочных часов:
    for wall in HOURGLASS:
        bext.goto(wall[X], wall[Y])
        print(WALL, end='')

    while True:  # Основной цикл программы.
        allSand = list(INITIAL_SAND)

        # Рисуем начальную горку песка:
        for sand in allSand:
            bext.goto(sand[X], sand[Y])
            print(SAND, end='')

        runHourglassSimulation(allSand)


def runHourglassSimulation(allSand):
    """Моделируем падение песка, пока он не прекратит двигаться.
    """
    while True:  # Проходим в цикле, пока песок не закончится.
        random.shuffle(allSand)  # Случайный порядок песчинок.

        sandMovedOnThisStep = False
        for i, sand in enumerate(allSand):
```

```
            if sand[Y] == SCREEN_HEIGHT - 1:
                # Песок на дне, а значит, больше не будет двигаться:
                continue

            # Если под песчинкой пусто, перемещаем ее вниз:
            noSandBelow = (sand[X], sand[Y] + 1) not in allSand
            noWallBelow = (sand[X], sand[Y] + 1) not in HOURGLASS
            canFallDown = noSandBelow and noWallBelow

            if canFallDown:
                # Отрисовываем песчинку ниже на одну строку:
                bext.goto(sand[X], sand[Y])
                print(' ', end='')  # Очищаем старое место ее расположения.
                bext.goto(sand[X], sand[Y] + 1)
                print(SAND, end='')

                # Песчинка ниже на одну строку:
                allSand[i] = (sand[X], sand[Y] + 1)
                sandMovedOnThisStep = True
            else:
                # Проверяем, может ли песчинка упасть влево:
                belowLeft = (sand[X] - 1, sand[Y] + 1)
                noSandBelowLeft = belowLeft not in allSand
                noWallBelowLeft = belowLeft not in HOURGLASS
                left = (sand[X] - 1, sand[Y])
                noWallLeft = left not in HOURGLASS
                notOnLeftEdge = sand[X] > 0
                canFallLeft = (noSandBelowLeft and noWallBelowLeft
                    and noWallLeft and notOnLeftEdge)

                # Проверяем, может ли песчинка упасть вправо:
                belowRight = (sand[X] + 1, sand[Y] + 1)
                noSandBelowRight = belowRight not in allSand
                noWallBelowRight = belowRight not in HOURGLASS
                right = (sand[X] + 1, sand[Y])
                noWallRight = right not in HOURGLASS
                notOnRightEdge = sand[X] < SCREEN_WIDTH - 1
                canFallRight = (noSandBelowRight and noWallBelowRight
                    and noWallRight and notOnRightEdge)

                # Задаем направление падения:
                fallingDirection = None
                if canFallLeft and not canFallRight:
                    fallingDirection = -1  # Задаем падение песка влево.
                elif not canFallLeft and canFallRight:
                    fallingDirection = 1  # Задаем падение песка вправо.
                elif canFallLeft and canFallRight:
                    # Возможны оба направления, так что выбираем случайным
                    # образом:
                    fallingDirection = random.choice((-1, 1))
                # Проверяем, может ли песчинка упасть "далеко",
```

```
135.                # на две позиции влево или вправо, вместо одной:
136.                if random.random() * 100 <= WIDE_FALL_CHANCE:
137.                    belowTwoLeft = (sand[X] - 2, sand[Y] + 1)
138.                    noSandBelowTwoLeft = belowTwoLeft not in allSand
139.                    noWallBelowTwoLeft = belowTwoLeft not in HOURGLASS
140.                    notOnSecondToLeftEdge = sand[X] > 1
141.                    canFallTwoLeft = (canFallLeft and noSandBelowTwoLeft
142.                        and noWallBelowTwoLeft and notOnSecondToLeftEdge)
143.
144.                    belowTwoRight = (sand[X] + 2, sand[Y] + 1)
145.                    noSandBelowTwoRight = belowTwoRight not in allSand
146.                    noWallBelowTwoRight = belowTwoRight not in HOURGLASS
147.                    notOnSecondToRightEdge = sand[X] < SCREEN_WIDTH - 2
148.                    canFallTwoRight = (canFallRight
149.                        and noSandBelowTwoRight and noWallBelowTwoRight
150.                        and notOnSecondToRightEdge)
151.
152.                    if canFallTwoLeft and not canFallTwoRight:
153.                        fallingDirection = -2
154.                    elif not canFallTwoLeft and canFallTwoRight:
155.                        fallingDirection = 2
156.                    elif canFallTwoLeft and canFallTwoRight:
157.                        fallingDirection = random.choice((-2, 2))
158.
159.                if fallingDirection == None:
160.                    # Эта песчинка никуда упасть не может,
161.                    # переходим к следующей.
162.                    continue
163.                # Отрисовываем песчинку на новом месте:
164.                bext.goto(sand[X], sand[Y])
165.                print(' ', end='')  # Затираем старую песчинку.
166.                bext.goto(sand[X] + fallingDirection, sand[Y] + 1)
167.                print(SAND, end='')  # Отрисовываем новую песчинку.
168.
169.                # Перемещаем песчинку на новое место:
170.                allSand[i] = (sand[X] + fallingDirection, sand[Y] + 1)
171.                sandMovedOnThisStep = True
172.
173.        sys.stdout.flush()  # (Необходимо для использующих bext программ.)
174.        time.sleep(PAUSE_LENGTH)  # Пауза
175.
176.        # Если на этом шаге песок вообще не двигался, обнуляем песочные часы:
177.        if not sandMovedOnThisStep:
178.            time.sleep(2)
179.            # Вытираем весь песок:
180.            for sand in allSand:
181.                bext.goto(sand[X], sand[Y])
182.                print(' ', end='')
183.            break  # Выходим из основного цикла моделирования.
184.
185.
```

```
186. # Если программа не импортируется, а запускается, производим запуск:
187. if __name__ == '__main__':
188.     try:
189.         main()
190.     except KeyboardInterrupt:
191.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- реализовать другие формы стенок, помимо песочных часов;
- создать места на экране, откуда непрерывно высыпаются новые песчинки.

Исследование программы

Попробуйте ответить на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `range(18, 37)` в строке 31 заменить на `range(18, 30)`?
2. Что будет, если `range(8)` в строке 39 заменить на `range(0)`?
3. Что будет, если `sandMovedOnThisStep = False` в строке 82 заменить на `sandMovedOnThisStep = True`?
4. Что будет, если выражение `fallingDirection = None` в строке 125 заменить на `fallingDirection = 1`?
5. Что будет, если `random.random() * 100 <= WIDE_FALL_CHANCE` в строке 136 заменить на `random.random() * 0 <= WIDE_FALL_CHANCE`?

37

Голодные роботы

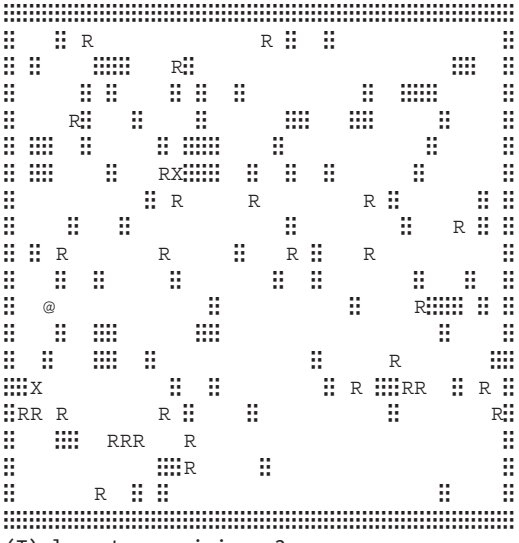
Вы заперты в лабиринте с голодными роботами! Почему роботам нужно есть — неизвестно, и вы не горите желанием это выяснить. Роботы плохо запрограммированы и движутся напрямик к вам, даже если на их пути стоит стена. Вам нужно обманом заставить роботов сталкиваться друг с другом (или мертвыми роботами) и не попасться им.

У вас есть специальное устройство, позволяющее телепортироваться в случайно выбираемое место, но заряда батареи хватит только на две телепортации. Кроме того, как вы, так и роботы могут срезать углы!

Программа в действии

Результат выполнения `hungryrobots.py` выглядит следующим образом:

```
Hungry Robots, by Al Sweigart al@inventwithpython.com
--сокращено--
```



```
(T)eleports remaining: 2
                    (Q) (W) ( )
                    (A) (S) (D)
Enter move or QUIT: (Z) (X) ( )
--сокращено--
```

Описание работы

Декартовы координаты x и y, соответствующие позициям в игре, позволяют нам математически вычислить направление, в котором должны двигаться роботы. В программировании x-координата растет при движении вправо, а y-координата — при движении вниз. Это значит, что если x-координата робота больше x-координаты игрока, то робот должен двигаться влево (то есть код должен производить вычитание из текущей x-координаты), чтобы приблизиться к игроку. Если же x-координата робота меньше, то он должен двигаться вправо (то есть код должен прибавлять к текущей x-координате). То же самое относится к движению вверх и вниз на основе соответствующих y-координат.

```
1. """Голодные роботы, (c) Эл Свейгарт al@inventwithpython.com
2. Сбегите от голодных роботов, сталкивая их друг с другом.
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: большая, игра"""
```

```
 5.
 6. import random, sys
 7.
 8. # Задаем константы:
 9. WIDTH = 40           # (!) Попробуйте заменить это значение на 70 или 10.
10. HEIGHT = 20          # (!) Попробуйте заменить это значение на 10.
11. NUM_ROBOTS = 10      # (!) Попробуйте заменить это значение на 1 или 30.
12. NUM_TELEPORTS = 2    # (!) Попробуйте заменить это значение на 0 или 9999.
13. NUM_DEAD_ROBOTS = 2  # (!) Попробуйте заменить это значение на 0 или 20.
14. NUM_WALLS = 100      # (!) Попробуйте заменить это значение на 0 или 1000.
15.
16. EMPTY_SPACE = ' '    # (!) Попробуйте заменить это значение на '.'.
17. PLAYER = '@'         # (!) Попробуйте заменить это значение на 'R'.
18. ROBOT = 'R'          # (!) Попробуйте заменить это значение на '@'.
19. DEAD_ROBOT = 'X'     # (!) Попробуйте заменить это значение на 'R'.
20.
21. # (!) Попробуйте заменить это значение на '#', или 'O', или ' ':
22. WALL = chr(9617)  # Символ 9617 соответствует '⣿'
23.
24.
25. def main():
26.     print('''Hungry Robots, by Al Sweigart al@inventwithpython.com
27.
28. You are trapped in a maze with hungry robots! You don't know why robots
29. need to eat, but you don't want to find out. The robots are badly
30. programmed and will move directly toward you, even if blocked by walls.
31. You must trick the robots into crashing into each other (or dead robots)
32. without being caught. You have a personal teleporter device, but it only
33. has enough battery for {} trips. Keep in mind, you and robots can slip
34. through the corners of two diagonal walls!
35. '''.format(NUM_TELEPORTS))
36.
37.     input('Press Enter to begin...')
38.
39.     # Подготавливаем новую игру:
40.     board = getNewBoard()
41.     robots = addRobots(board)
42.     playerPosition = getRandomEmptySpace(board, robots)
43.     while True:  # Основной цикл игры.
44.         displayBoard(board, robots, playerPosition)
45.
46.         if len(robots) == 0:  # Проверяем, не выиграл ли игрок.
47.             print('All the robots have crashed into each other and you')
48.             print('lived to tell the tale! Good job!')
49.             sys.exit()
50.
51.         # Передвигаем игрока и роботов:
52.         playerPosition = askForPlayerMove(board, robots, playerPosition)
53.         robots = moveRobots(board, robots, playerPosition)
54.
55.         for x, y in robots:  # Проверяем, не проиграл ли игрок.
```

```
            if (x, y) == playerPosition:
                displayBoard(board, robots, playerPosition)
                print('You have been caught by a robot!')
                sys.exit()


def getNewBoard():
    """Возвращает соответствующий доске ассоциативный массив. Роль ключей
    играют кортежи (x, y) целочисленных индексов позиций на доске,
    а значений — WALL, EMPTY_SPACE и DEAD_ROBOT. В ассоциативном массиве есть
    также ключ 'teleports' для числа оставшихся у игрока попыток телепортации.
    Живые роботы хранятся отдельно от ассоциативного массива board"""
    board = {'teleports': NUM_TELEPORTS}

    # Создаем пустую доску:
    for x in range(WIDTH):
        for y in range(HEIGHT):
            board[(x, y)] = EMPTY_SPACE

    # Добавляем стенки на краях доски:
    for x in range(WIDTH):
        board[(x, 0)] = WALL  # Верхняя стенка.
        board[(x, HEIGHT - 1)] = WALL  # Нижняя стенка.
    for y in range(HEIGHT):
        board[(0, y)] = WALL  # Левая стенка.
        board[(WIDTH - 1, y)] = WALL  # Правая стенка.

    # Добавляем еще стенки случайным образом:
    for i in range(NUM_WALLS):
        x, y = getRandomEmptySpace(board, [])
        board[(x, y)] = WALL

    # Добавляем изначально мертвых роботов:
    for i in range(NUM_DEAD_ROBOTS):
        x, y = getRandomEmptySpace(board, [])
        board[(x, y)] = DEAD_ROBOT
    return board

def getRandomEmptySpace(board, robots):
    """Возвращает целочисленный кортеж (x, y) для пустого пространства
    на доске."""
    while True:
        randomX = random.randint(1, WIDTH - 2)
        randomY = random.randint(1, HEIGHT - 2)
        if isEmpty(randomX, randomY, board, robots):
            break
    return (randomX, randomY)


def isEmpty(x, y, board, robots):
```

```
106.     """Возвращает True, если клетка (x, y) на доске пуста
107.     и не содержит робота."""
108.     return board[(x, y)] == EMPTY_SPACE and (x, y) not in robots
109.
110.
111. def addRobots(board):
112.     """Добавляет NUM_ROBOTS роботов в пустые места на доске, а также возвращает
113.     список (x, y) этих пустых мест, в которых теперь находятся роботы."""
114.     robots = []
115.     for i in range(NUM_ROBOTS):
116.         x, y = getRandomEmptySpace(board, robots)
117.         robots.append((x, y))
118.     return robots
119.
120.
121. def displayBoard(board, robots, playerPosition):
122.     """Отображает доску, роботов и игрока на экране."""
123.     # Проходим в цикле по всем позициям на доске:
124.     for y in range(HEIGHT):
125.         for x in range(WIDTH):
126.             # Рисуем соответствующий символ:
127.             if board[(x, y)] == WALL:
128.                 print(WALL, end='')
129.             elif board[(x, y)] == DEAD_ROBOT:
130.                 print(DEAD_ROBOT, end='')
131.             elif (x, y) == playerPosition:
132.                 print(PLAYER, end='')
133.             elif (x, y) in robots:
134.                 print(ROBOT, end='')
135.             else:
136.                 print(EMPTY_SPACE, end='')
137.         print()  # Выводим символ новой строки.
138.
139.
140. def askForPlayerMove(board, robots, playerPosition):
141.     """Возвращает целочисленный кортеж (x, y) места, куда далее идет
142.     игрок, с учетом текущего местоположения и краев доски."""
143.     playerX, playerY = playerPosition
144.
145.     # Ищем, движение в каких направлениях не преграждает стенка:
146.     q = 'Q' if isEmpty(playerX - 1, playerY - 1, board, robots) else ' '
147.     w = 'W' if isEmpty(playerX + 0, playerY - 1, board, robots) else ' '
148.     e = 'E' if isEmpty(playerX + 1, playerY - 1, board, robots) else ' '
149.     d = 'D' if isEmpty(playerX + 1, playerY + 0, board, robots) else ' '
150.     c = 'C' if isEmpty(playerX + 1, playerY + 1, board, robots) else ' '
151.     x = 'X' if isEmpty(playerX + 0, playerY + 1, board, robots) else ' '
152.     z = 'Z' if isEmpty(playerX - 1, playerY + 1, board, robots) else ' '
153.     a = 'A' if isEmpty(playerX - 1, playerY + 0, board, robots) else ' '
154.     allMoves = (q + w + e + d + c + x + a + z + 'S')
155.
156.     while True:
```

```
157.         # Определяем ход игрока:
158.         print('(T)eleports remaining: {}'.format(board["teleports"]))
159.         print('                       ({}) ({}) ({})'.format(q, w, e))
160.         print('                       ({}) (S) ({})'.format(a, d))
161.         print('Enter move or QUIT: ({}) ({}) ({})'.format(z, x, c))
162.
163.         move = input('> ').upper()
164.         if move == 'QUIT':
165.             print('Thanks for playing!')
166.             sys.exit()
167.         elif move == 'T' and board['teleports'] > 0:
168.             # Телепортируем игрока на случайную новую позицию:
169.             board['teleports'] -= 1
170.             return getRandomEmptySpace(board, robots)
171.         elif move != '' and move in allMoves:
172.             # Возвращаем новую позицию игрока в соответствии со сделанным ходом:
173.             return {'Q': (playerX - 1, playerY - 1),
174.                     'W': (playerX + 0, playerY - 1),
175.                     'E': (playerX + 1, playerY - 1),
176.                     'D': (playerX + 1, playerY + 0),
177.                     'C': (playerX + 1, playerY + 1),
178.                     'X': (playerX + 0, playerY + 1),
179.                     'Z': (playerX - 1, playerY + 1),
180.                     'A': (playerX - 1, playerY + 0),
181.                     'S': (playerX, playerY)}[move]
182.
183.
184. def moveRobots(board, robotPositions, playerPosition):
185.     """Возвращаем список кортежей (x, y) новых позиций роботов,
186.     после их попыток переместиться в направлении игрока."""
187.     playerx, playery = playerPosition
188.     nextRobotPositions = []
189.
190.     while len(robotPositions) > 0:
191.         robotx, roboty = robotPositions[0]
192.
193.         # Определяем направление движения робота.
194.         if robotx < playerx:
195.             movex = 1  # Перемещаем вправо.
196.         elif robotx > playerx:
197.             movex = -1  # Перемещаем влево.
198.         elif robotx == playerx:
199.             movex = 0  # Не перемещаем по горизонтали.
200.
201.         if roboty < playery:
202.             movey = 1  # Перемещаем вверх.
203.         elif roboty > playery:
204.             movey = -1  # Перемещаем вниз.
205.         elif roboty == playery:
206.             movey = 0  # Не перемещаем по вертикали.
207.         # Проверяем, не натыкается ли робот на стену, и корректируем
```

```
208.         # направление его движения:
209.         if board[(robotx + movex, roboty + movey)] == WALL:
210.             # Робот натолкнется на стену, так что выбираем другой ход:
211.             if board[(robotx + movex, roboty)] == EMPTY_SPACE:
212.                 movey = 0  # Робот не может переместиться горизонтально.
213.             elif board[(robotx, roboty + movey)] == EMPTY_SPACE:
214.                 movex = 0  # Робот не может переместиться вертикально.
215.             else:
216.                 # Робот не может переместиться.
217.                 movex = 0
218.                 movey = 0
219.         newRobotx = robotx + movex
220.         newRoboty = roboty + movey
221.
222.         if (board[(robotx, roboty)] == DEAD_ROBOT
223.             or board[(newRobotx, newRoboty)] == DEAD_ROBOT):
224.             # Робот попал на место столкновения, удаляем его.
225.             del robotPositions[0]
226.             continue
227.         # Проверяем, не наткнулся ли он на другого робота, и уничтожаем обоих
228.         # в этом случае:
229.         if (newRobotx, newRoboty) in nextRobotPositions:
230.             board[(newRobotx, newRoboty)] = DEAD_ROBOT
231.             nextRobotPositions.remove((newRobotx, newRoboty))
232.         else:
233.             nextRobotPositions.append((newRobotx, newRoboty))
234.
235.         # Удаляем роботов из robotPositions по мере их перемещения.
236.         del robotPositions[0]
237.     return nextRobotPositions
238.
239.
240. # Если программа не импортируется, а запускается, производим запуск:
241. if __name__ == '__main__':
242.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- создайте роботов двух различных видов: способных перемещаться только по диагонали и только в основных направлениях;
- предоставьте игроку возможность оставлять после себя ограниченное количество ловушек, останавливающих любого ступившего на них робота;
- предоставьте игроку возможность создать для своей защиты ограниченное количество «мгновенных стен».

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `WALL = chr(9617)` в строке 22 заменить на `WALL = 'R'`?
2. Что будет, если `return nextRobotPositions` в строке 237 заменить на `return robotPositions`?
3. Что будет, если удалить или закомментировать `displayBoard(board, robots, playerPosition)` в строке 44?
4. Что будет, если удалить или закомментировать `robots = moveRobots(board, robots, playerPosition)` в строке 53?

38

«Я обвиняю!»

Вы — известный детектив Матильда Камю. Пропала кошка Зофи, и вам нужно изучить все зацепки. Подозреваемые всегда врут или всегда говорят правду. Найдете ли вы Зофи вовремя и обвините ли того, кого нужно?

В данной игре вы едете на такси в различные места го́рода. В каждом из этих мест есть подозреваемый и предмет. Вы можете спрашивать подозреваемых о других подозреваемых и предметах, сравнивать их ответы со своими записями и определять, говорят они правду или лгут. Кто-нибудь из них может знать, кто похитил Зофи (или где она находится, или какой предмет можно найти в месте пребывания похитителя), но вы должны решить, верите ли вы им. На поиск злоумышленника у вас пять минут, но если вы три раза обвините кого-то ошибочно, то проиграете. Эта игра создана под впечатлением от игры Where's an Egg? («Найди яйцо») с сайта Homestar Runner.

Программа в действии

Результат выполнения `jaccuse.py` выглядит следующим образом:

```
J'ACCUSE! (a mystery game)
--сокращено--
Time left: 5 min, 0 sec
  You are in your TAXI. Where do you want to go?
(A)LBINO ALLIGATOR PIT
```

```
(B)OWLING ALLEY
(C)ITY HALL
(D)UCK POND
(H)IPSTER CAFE
(O)LD BARN
(U)NIVERSITY LIBRARY
(V)IDEO GAME MUSEUM
(Z)OO
> a

Time left: 4 min, 48 sec
  You are at the ALBINO ALLIGATOR PIT.
  ESPRESSA TOFFEEPOT with the ONE COWBOY BOOT is here.

(J) "J'ACCUSE!" (3 accusations left)
(Z) Ask if they know where ZOPHIE THE CAT is.
(T) Go back to the TAXI.
(1) Ask about ESPRESSA TOFFEEPOT
(2) Ask about ONE COWBOY BOOT
> z
  They give you this clue: "DUKE HAUTDOG"
Press Enter to continue...
--сокращено--
```

Описание работы

Чтобы разобраться с данной игрой, необходимо внимательно изучить ассоциативный массив `clues`, описанный в строках с 51-й по 109-ю. Можете раскомментировать строки с 151-й по 154-ю, чтобы вывести его на экран. Роль ключей в этом ассоциативном массиве играют строковые значения из списка `SUSPECTS`, а роль значений — «ассоциативные массивы зацепок». Каждый из этих ассоциативных массивов зацепок содержит строковые значения из списков `SUSPECTS` и `ITEMS`, служащие ответами одних подозреваемых относительно других подозреваемых и предметов. Например, если `clues['DUKE HAUTDOG']['CANDLESTICK']` равно `'DUCK POND'`, то, когда мы спросим Дюка Отдога о подсвечнике (Candlestick), Дюк ответит, что тот находится в Утином пруду (Duck pond). Подозреваемые, предметы, места и преступник перетасовываются при каждой новой игре.

Эта структура данных — центральная для нашей программы, следовательно, чтобы разобраться в программе в целом, необходимо разобраться с этой структурой.

```
1. """Я обвиняю!, (c) Эл Свейгарт al@inventwithpython.com
2. Детективная игра с обманом и пропавшей кошкой
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: очень большая, игра, юмор, головоломка"""
5.
6. # Сыграть в исходную Flash-игру вы можете по адресу:
7. # https://homestarrunner.com/videlectrix/wheresanegg.html
```

```
 8. # Больше информации — в статье http://www.hrwiki.org/wiki/Where's_an_Egg%3F
 9.
10. import time, random, sys
11.
12. # Задаем константы:
13. SUSPECTS = ['DUKE HAUTDOG', 'MAXIMUM POWERS', 'BILL MONOPOLIS', 'SENATOR
    SCHMEAR', 'MRS. FEATHERTOSS', 'DR. JEAN SPLICER', 'RAFFLES THE CLOWN',
    'ESPRESSA TOFFEEPOT', 'CECIL EDGAR VANDERTON']
14. ITEMS = ['FLASHLIGHT', 'CANDLESTICK', 'RAINBOW FLAG', 'HAMSTER WHEEL', 'ANIME
    VHS TAPE', 'JAR OF PICKLES', 'ONE COWBOY BOOT', 'CLEAN UNDERPANTS', '5 DOLLAR
    GIFT CARD']
15. PLACES = ['ZOO', 'OLD BARN', 'DUCK POND', 'CITY HALL', 'HIPSTER CAFE',
    'BOWLING ALLEY', 'VIDEO GAME MUSEUM', 'UNIVERSITY LIBRARY', 'ALBINO
    ALLIGATOR PIT']
16. TIME_TO_SOLVE = 300  # Длительность игры — 300 секунд (5 минут).
17. # Первые буквы и максимальная длина мест действия необходимы для отображения
18. # меню:
19. PLACE_FIRST_LETTERS = {}
20. LONGEST_PLACE_NAME_LENGTH = 0
21. for place in PLACES:
22.     PLACE_FIRST_LETTERS[place[0]] = place
23.     if len(place) > LONGEST_PLACE_NAME_LENGTH:
24.         LONGEST_PLACE_NAME_LENGTH = len(place)
25.
26. # Основные проверки корректности констант:
27. assert len(SUSPECTS) == 9
28. assert len(ITEMS) == 9
29. assert len(PLACES) == 9
30. # Первые буквы не должны повторяться:
31. assert len(PLACE_FIRST_LETTERS.keys()) == len(PLACES)
32.
33.
34. knownSuspectsAndItems = []
35. # visitedPlaces: ключи — места действия, значения —
    ➥ строковые значения для находящихся там подозреваемых и предметов.
36. visitedPlaces = {}
37. currentLocation = 'TAXI'  # Начинаем игру в такси.
38. accusedSuspects = []  # Обвиненные подозреваемые никаких зацепок не дают.
39. liars = random.sample(SUSPECTS, random.randint(3, 4))
40. accusationsLeft = 3  # Вы можете обвинить не более трех человек.
41. culprit = random.choice(SUSPECTS)
42. # Ссылки на общие индексы; например, SUSPECTS[0] и ITEMS[0] находятся
43. # в PLACES[0].
44. random.shuffle(SUSPECTS)
45. random.shuffle(ITEMS)
46. random.shuffle(PLACES)
47.
48. # Создаем структуры данных для зацепок, полученных от говорящих правду
49. # о каждом из предметов и подозреваемых.
50. # clues: ключи — подозреваемые, у которых попросили зацепку,
    ➥ значение — "ассоциативный массив зацепок".
```

```
clues = {}
for i, interviewee in enumerate(SUSPECTS):
    if interviewee in liars:
        continue  # Пока пропускаем лжецов.

    # Ключи в этом "ассоциативном массиве зацепок" — предметы
    # и подозреваемые, значения — полученные зацепки.
    clues[interviewee] = {}
    clues[interviewee]['debug_liar'] = False  # Удобно для отладки.
    for item in ITEMS:  # Выбираем зацепки относительно всех предметов.
        if random.randint(0, 1) == 0:  # Говорит, где находится предмет:
            clues[interviewee][item] = PLACES[ITEMS.index(item)]
        else:  # Говорит, у кого предмет:
            clues[interviewee][item] = SUSPECTS[ITEMS.index(item)]
    for suspect in SUSPECTS:  # Выбираем зацепки относительно всех
                              # подозреваемых.
        if random.randint(0, 1) == 0:  # Говорит, где находится подозреваемый:
            clues[interviewee][suspect] = PLACES[SUSPECTS.index(suspect)]
        else:  # Говорит, какой предмет есть у подозреваемого:
            clues[interviewee][suspect] = ITEMS[SUSPECTS.index(suspect)]
# Создаем структуры данных для получаемых от лжецов зацепок
# относительно всех предметов и подозреваемых:
for i, interviewee in enumerate(SUSPECTS):
    if interviewee not in liars:
        continue  # Мы уже обработали тех, кто говорит правду.

    # Ключи в этом "ассоциативном массиве зацепок" — предметы
    # и подозреваемые, значения — полученные зацепки.
    clues[interviewee] = {}
    clues[interviewee]['debug_liar'] = True  # Удобно для отладки.

    # Этот опрашиваемый подозреваемый – лжец, и его зацепки ложны:
    for item in ITEMS:
        if random.randint(0, 1) == 0:
            while True:  # Выбираем случайное (неправильное) место.
                # Лжет относительно местонахождения предмета.
                clues[interviewee][item] = random.choice(PLACES)
                if clues[interviewee][item] != PLACES[ITEMS.index(item)]:
                    # Выходим из цикла после выбора ложной зацепки.
                    break
        else:
            while True:  # Выбираем случайного (неправильного) подозреваемого.
                clues[interviewee][item] = random.choice(SUSPECTS)
                if clues[interviewee][item] != SUSPECTS[ITEMS.index(item)]:
                    # Выходим из цикла после выбора ложной зацепки.
                    break
    for suspect in SUSPECTS:
        if random.randint(0, 1) == 0:
            while True:  # Выбираем случайное (неправильное) место.
                clues[interviewee][suspect] = random.choice(PLACES)
                if clues[interviewee][suspect] != PLACES[ITEMS.index(item)]:
```

```
                    # Выходим из цикла после выбора ложной зацепки.
                    break
        else:
            while True:  # Выбираем случайный (неправильный) предмет.
                clues[interviewee][suspect] = random.choice(ITEMS)
                if clues[interviewee][suspect] !=
                ➥ ITEMS[SUSPECTS.index(suspect)]:
                    # Выходим из цикла после выбора ложной зацепки.
                    break

# Создаем структуры данных для ответов на вопросы о Зофи:
zophieClues = {}
for interviewee in random.sample(SUSPECTS, random.randint(3, 4)):
    kindOfClue = random.randint(1, 3)
    if kindOfClue == 1:
        if interviewee not in liars:
            # (Правдиво) отвечают, у кого Зофи.
            zophieClues[interviewee] = culprit
        elif interviewee in liars:
            while True:
                # Выбираем (неправильного) подозреваемого.
                zophieClues[interviewee] = random.choice(SUSPECTS)
                if zophieClues[interviewee] != culprit:
                    # Выходим из цикла после выбора ложной зацепки.
                    break

    elif kindOfClue == 2:
        if interviewee not in liars:
            # (Правдиво) отвечают, где Зофи.
            zophieClues[interviewee] = PLACES[SUSPECTS.index(culprit)]
        elif interviewee in liars:
            while True:
                # Выбираем случайное (неправильное) место.
                zophieClues[interviewee] = random.choice(PLACES)
                if zophieClues[interviewee] != PLACES[SUSPECTS.index(culprit)]:
                    # Выходим из цикла после выбора ложной зацепки.
                    break
    elif kindOfClue == 3:
        if interviewee not in liars:
            # (Правдиво) отвечают, близ какого предмета находится Зофи.
            zophieClues[interviewee] = ITEMS[SUSPECTS.index(culprit)]
        elif interviewee in liars:
            while True:
                # Выбираем случайный (неправильный) предмет.
                zophieClues[interviewee] = random.choice(ITEMS)
                if zophieClues[interviewee] != ITEMS[SUSPECTS.index(culprit)]:
                    # Выходим из цикла после выбора ложной зацепки.
                    break

# Эксперимент: раскомментируйте этот код, чтобы посмотреть на содержимое
# структур данных с зацепками:
```

```
152. #import pprint
153. #pprint.pprint(clues)
154. #pprint.pprint(zophieClues)
155. #print('culprit =', culprit)
156. # НАЧАЛО ИГРЫ
157. print("""J'ACCUSE! (a mystery game)")
158. By Al Sweigart al@inventwithpython.com
159. Inspired by Homestar Runner\'s "Where\'s an Egg?" game
160.
161. You are the world-famous detective Mathilde Camus.
162. ZOPHIE THE CAT has gone missing, and you must sift through the clues.
163. Suspects either always tell lies, or always tell the truth. Ask them
164. about other people, places, and items to see if the details they give are
165. truthful and consistent with your observations. Then you will know if
166. their clue about ZOPHIE THE CAT is true or not. Will you find ZOPHIE THE
167. CAT in time and accuse the guilty party?
168. """)
169. input('Press Enter to begin...')
170.
171.
172. startTime = time.time()
173. endTime = startTime + TIME_TO_SOLVE
174.
175. while True:  # Основной цикл игры.
176.     if time.time() > endTime or accusationsLeft == 0:
177.         # Обрабатываем условие "игра окончена":
178.         if time.time() > endTime:
179.             print('You have run out of time!')
180.         elif accusationsLeft == 0:
181.             print('You have accused too many innocent people!')
182.         culpritIndex = SUSPECTS.index(culprit)
183.         print('It was {} at the {} with the {} who catnapped her!'.
             format(culprit, PLACES[culpritIndex], ITEMS[culpritIndex]))
184.         print('Better luck next time, Detective.')
185.         sys.exit()
186.
187.     print()
188.     minutesLeft = int(endTime - time.time()) // 60
189.     secondsLeft = int(endTime - time.time()) % 60
190.     print('Time left: {} min, {} sec'.format(minutesLeft, secondsLeft))
191.
192.     if currentLocation == 'TAXI':
193.         print('  You are in your TAXI. Where do you want to go?')
194.         for place in sorted(PLACES):
195.             placeInfo = ''
196.             if place in visitedPlaces:
197.                 placeInfo = visitedPlaces[place]
198.             nameLabel = '(' + place[0] + ')' + place[1:]
199.             spacing = " " * (LONGEST_PLACE_NAME_LENGTH - len(place))
200.             print('{} {}{}'.format(nameLabel, spacing, placeInfo))
201.         print('(Q)UIT GAME')
```

```
        while True:  # Продолжаем спрашивать, пока не получим допустимый ответ.
            response = input('> ').upper()
            if response == '':
                continue  # Спрашиваем снова.
            if response == 'Q':
                print('Thanks for playing!')
                sys.exit()
            if response in PLACE_FIRST_LETTERS.keys():
                break
        currentLocation = PLACE_FIRST_LETTERS[response]
        continue  # Возвращаемся к началу основного цикла игры.

    # Находимся в одном из мест; игрок может запрашивать зацепки.
    print('  You are at the {}.'.format(currentLocation))
    currentLocationIndex = PLACES.index(currentLocation)
    thePersonHere = SUSPECTS[currentLocationIndex]
    theItemHere = ITEMS[currentLocationIndex]
    print('  {} with the {} is here.'.format(thePersonHere, theItemHere))

    # Добавляем находящихся в этом месте подозреваемого и предмет
    # в наш список известных подозреваемых и предметов:
    if thePersonHere not in knownSuspectsAndItems:
        knownSuspectsAndItems.append(thePersonHere)
    if ITEMS[currentLocationIndex] not in knownSuspectsAndItems:
        knownSuspectsAndItems.append(ITEMS[currentLocationIndex])
    if currentLocation not in visitedPlaces.keys():
        visitedPlaces[currentLocation] = '({}, {})'.format(
        ➥ thePersonHere.lower(), theItemHere.lower())

    # Обвиненные ранее ошибочно подозреваемые не дают
    # игроку зацепок:
    if thePersonHere in accusedSuspects:
        print('They are offended that you accused them,')
        print('and will not help with your investigation.')
        print('You go back to your TAXI.')
        print()
        input('Press Enter to continue...')
        currentLocation = 'TAXI'
        continue  # Возвращаемся к началу основного цикла игры.
    # Отображаем меню с известными подозреваемыми и предметами, о которых
    # можно спросить:
    print()
    print('(J) "J\'ACCUSE!" ({} accusations left)'.format(accusationsLeft))
    print('(Z) Ask if they know where ZOPHIE THE CAT is.')
    print('(T) Go back to the TAXI.')
    for i, suspectOrItem in enumerate(knownSuspectsAndItems):
        print('({}) Ask about {}'.format(i + 1, suspectOrItem))

    while True:  # Продолжаем спрашивать, пока не получим допустимый ответ.
        response = input('> ').upper()
        if response in 'JZT' or (response.isdecimal() and 0 < int(response) <=
        len(knownSuspectsAndItems)):
```

```
252.             break
253.
254.     if response == 'J':  # Игрок обвиняет этого подозреваемого.
255.         accusationsLeft -= 1  # Учитываем использованное обвинение.
256.         if thePersonHere == culprit:
257.             # Вы обвинили того, кого нужно.
258.             print('You\'ve cracked the case, Detective!')
259.             print('It was {} who had catnapped ZOPHIE THE CAT.'.format(culprit))
260.             minutesTaken = int(time.time() - startTime) // 60
261.             secondsTaken = int(time.time() - startTime) % 60
262.             print('Good job! You solved it in {} min, {}
                 sec.'.format(minutesTaken, secondsTaken))
263.             sys.exit()
264.         else:
265.             # Вы обвинили не того, кого нужно.
266.             accusedSuspects.append(thePersonHere)
267.             print('You have accused the wrong person, Detective!')
268.             print('They will not help you with anymore clues.')
269.             print('You go back to your TAXI.')
270.             currentLocation = 'TAXI'
271.
272.     elif response == 'Z':  # Игрок спрашивает о Зофи.
273.         if thePersonHere not in zophieClues:
274.             print('"I don\'t know anything about ZOPHIE THE CAT."')
275.         elif thePersonHere in zophieClues:
276.             print('  They give you this clue:
                 ➥ "{}"'.format(zophieClues[thePersonHere]))
277.             # Добавляем не относящиеся к местам зацепки в список известного:
278.             if zophieClues[thePersonHere] not in knownSuspectsAndItems and
                 zophieClues[thePersonHere] not in PLACES:
279.                 knownSuspectsAndItems.append(zophieClues[thePersonHere])
280.
281.     elif response == 'T':  # Игрок возвращается в такси.
282.         currentLocation = 'TAXI'
283.         continue  # Возвращаемся к началу основного цикла игры.
284.
285.     else:  # Игрок спрашивает о подозреваемом или предмете.
286.         thingBeingAskedAbout = knownSuspectsAndItems[int(response) - 1]
287.         if thingBeingAskedAbout in (thePersonHere, theItemHere):
288.             print('  They give you this clue: "No comment."')
289.         else:
290.             print('  They give you this clue:
                 "{}"'.format(clues[thePersonHere][thingBeingAskedAbout]))
291.             # Добавляем не относящиеся к местам зацепки в список известного:
292.             if clues[thePersonHere][thingBeingAskedAbout] not in
                 ➥ knownSuspectsAndItems
                 ➥ and clues[thePersonHere][thingBeingAskedAbout] not in PLACES:
293.                 knownSuspectsAndItems.append(clues[thePersonHere]
                     ➥ [thingBeingAskedAbout])
294.
295.     input('Press Enter to continue...')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `TIME_TO_SOLVE = 300` в строке 16 заменить на `TIME_TO_SOLVE = 0`?
2. Что будет, если `time.time() > endTime or accusationsLeft == 0` в строке 176 заменить на `time.time() > endTime and accusationsLeft == 0`?
3. Что будет, если `place[1:]` в строке 198 заменить на `place`?
4. Что будет, если `startTime + TIME_TO_SOLVE` в строке 173 заменить на `startTime * TIME_TO_SOLVE`?

39

Муравей Лэнгтона

«Муравей Лэнгтона» — клеточный автомат для имитационного моделирования на двумерной сетке, схожий с проектом 13. При моделировании муравей начинает с клетки одного из двух цветов. Если клетка окрашена в первый цвет, то муравей меняет его на второй, поворачивается на 90 градусов вправо и перемещается на одну клетку. Если эта клетка второго цвета, то муравей меняет ее на первый, поворачивается на 90 градусов влево и перемещается вперед на одну клетку.

Несмотря на очень простой набор правил, при этом моделировании демонстрируется сложное эмерджентное поведение. Можно моделировать несколько муравьев в одном пространстве, что вызывает интересные взаимодействия при пересечении их путей движения. Муравья Лэнгтона придумал специалист в области компьютерных наук Крис Лэнгтон (Chris Langton) в 1986 году. Дополнительную информацию о «Муравье Лэнгтона» можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Муравей_Лэнгтона.

Программа в действии

На рис. 39.1 показан результат выполнения `langtonsant.py`.

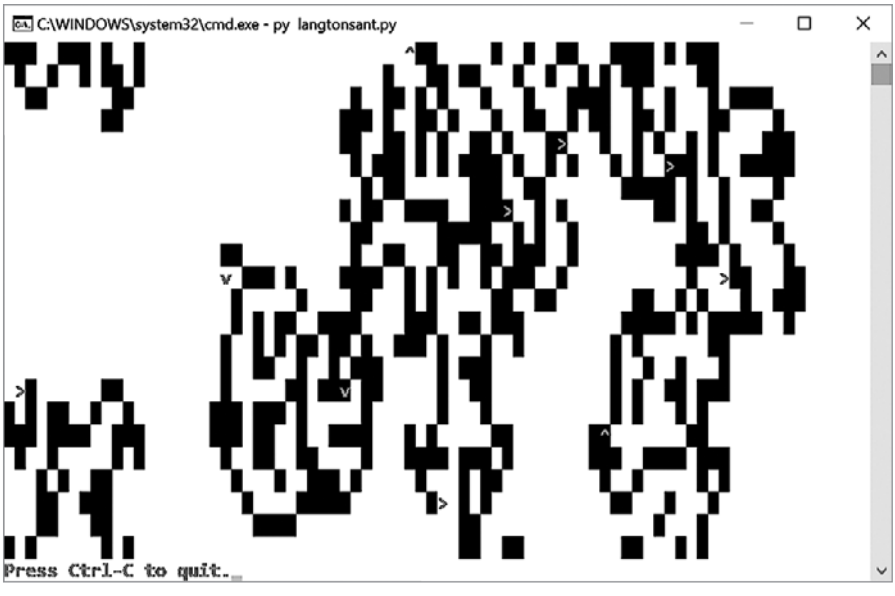


Рис. 39.1. Завораживающие результаты работы клеточного автомата «Муравей Лэнгтона»

Описание работы

В этой программе есть два вида «направлений движения». Во-первых, в ассоциативных массивах, соответствующих муравьям, хранятся *основные направления движения*: север, юг, запад, восток, а также *направления вращения*: повороты налево или направо (точнее, против часовой стрелки и по ней, поскольку мы смотрим на муравьев сверху). Муравьи поворачивают налево или направо в зависимости от клетки, на которой стоят, так что строки с 78-й по 100-ю задают новое основное направление на базе текущего основного направления движения и направления вращения.

```
1. """Муравей Лэнгтона, (с) Эл Свейгарт al@inventwithpython.com
2. Динамическое изображение клеточного автомата. Нажмите Ctrl+C для останова.
3. Дополнительная информация: https://ru.wikipedia.org/wiki/Муравей_Лэнгтона
4. Код размещен на https://nostarch.com/big-book-small-python-projects
5. Теги: большая, графика, bext, имитационное моделирование"""
6.
```

```
 7. import copy, random, sys, time
 8.
 9. try:
10.     import bext
11. except ImportError:
12.     print('This program requires the bext module, which you')
13.     print('can install by following the instructions at')
14.     print('https://pypi.org/project/Bext/')
15.     sys.exit()
16.
17. # Задаем константы:
18. WIDTH, HEIGHT = bext.size()
19. # В Windows нельзя вывести что-либо в последнем столбце без добавления
20. # автоматически символа новой строки, поэтому уменьшаем ширину на 1:
21. WIDTH -= 1
22. HEIGHT -= 1  # Учет сообщения о возможности выхода в самом низу.
23.
24. NUMBER_OF_ANTS = 10  # (!) Попробуйте заменить это значение на 1 или 50.
25. PAUSE_AMOUNT = 0.1  # (!) Попробуйте заменить это значение на 1.0 или 0.0.
26.
27. # (!) Попробуйте поменять их, чтобы изменить внешний вид муравьев:
28. ANT_UP = '^'
29. ANT_DOWN = 'v'
30. ANT_LEFT = '<'
31. ANT_RIGHT = '>'
32.
33. # (!) Попробуйте заменить эти цвета на что-то из 'black', 'red',
34. # 'green', 'yellow', 'blue', 'purple', 'cyan', or 'white'.
35. # (Все поддерживаемые модулем bext цвета.)
36. ANT_COLOR = 'red'
37. BLACK_TILE = 'black'
38. WHITE_TILE = 'white'
39.
40. NORTH = 'north'
41. SOUTH = 'south'
42. EAST = 'east'
43. WEST = 'west'
44.
45.
46. def main():
47.     bext.fg(ANT_COLOR)  # Цвет муравьев совпадает с цветом переднего плана.
48.     bext.bg(WHITE_TILE)  # Задаем для начала белый цвет фона.
49.     bext.clear()
50.
51.     # Создаем новую структуру данных для доски:
52.     board = {'width': WIDTH, 'height': HEIGHT}
53.
54.     # Создаем структуры данных для муравьев:
55.     ants = []
56.     for i in range(NUMBER_OF_ANTS):
57.         ant = {
```

```
            'x': random.randint(0, WIDTH - 1),
            'y': random.randint(0, HEIGHT - 1),
            'direction': random.choice([NORTH, SOUTH, EAST, WEST]),
        }
        ants.append(ant)

    # Отслеживаем поменявшие цвет клетки, которые нужно перерисовать
    # на экране:
    changedTiles = []

    while True:  # Основной цикл программы.
        displayBoard(board, ants, changedTiles)
        changedTiles = []

        # nextBoard — то, как доска будет выглядеть на следующем шаге
        # моделирования. Начинаем с копии доски для текущего шага:
        nextBoard = copy.copy(board)

        # Выполняем один шаг моделирования для каждого муравья:
        for ant in ants:
            if board.get((ant['x'], ant['y']), False) == True:
                nextBoard[(ant['x'], ant['y'])] = False
                # Поворот по часовой стрелке:
                if ant['direction'] == NORTH:
                    ant['direction'] = EAST
                elif ant['direction'] == EAST:
                    ant['direction'] = SOUTH
                elif ant['direction'] == SOUTH:
                    ant['direction'] = WEST
                elif ant['direction'] == WEST:
                    ant['direction'] = NORTH
            else:
                nextBoard[(ant['x'], ant['y'])] = True
                # Поворот против часовой стрелки:
                if ant['direction'] == NORTH:
                    ant['direction'] = WEST
                elif ant['direction'] == WEST:
                    ant['direction'] = SOUTH
                elif ant['direction'] == SOUTH:
                    ant['direction'] = EAST
                elif ant['direction'] == EAST:
                    ant['direction'] = NORTH
            changedTiles.append((ant['x'], ant['y']))

            # Передвигаем муравья вперед в направлении, в котором он "смотрит":
            if ant['direction'] == NORTH:
                ant['y'] -= 1
            if ant['direction'] == SOUTH:
                ant['y'] += 1
            if ant['direction'] == WEST:
                ant['x'] -= 1
```

```
109.             if ant['direction'] == EAST:
110.                 ant['x'] += 1
111.
112.             # Если муравей пересекает край экрана, необходимо
113.             # перенести его на другую сторону.
114.             ant['x'] = ant['x'] % WIDTH
115.             ant['y'] = ant['y'] % HEIGHT
116.
117.             changedTiles.append((ant['x'], ant['y']))
118.
119.         board = nextBoard
120.
121.
122. def displayBoard(board, ants, changedTiles):
123.     """Отображает доску и муравьев на экране. Аргумент changedTiles
124.     представляет собой список кортежей (x, y) для поменявшихся клеток
125.     на экране, которые необходимо перерисовать."""
126.
127.     # Выводим на экран структуру данных для доски:
128.     for x, y in changedTiles:
129.         bext.goto(x, y)
130.         if board.get((x, y), False):
131.             bext.bg(BLACK_TILE)
132.         else:
133.             bext.bg(WHITE_TILE)
134.
135.         antIsHere = False
136.         for ant in ants:
137.             if (x, y) == (ant['x'], ant['y']):
138.                 antIsHere = True
139.                 if ant['direction'] == NORTH:
140.                     print(ANT_UP, end='')
141.                 elif ant['direction'] == SOUTH:
142.                     print(ANT_DOWN, end='')
143.                 elif ant['direction'] == EAST:
144.                     print(ANT_LEFT, end='')
145.                 elif ant['direction'] == WEST:
146.                     print(ANT_RIGHT, end='')
147.                 break
148.         if not antIsHere:
149.             print(' ', end='')
150.
151.     # Отображаем внизу экрана сообщение о возможности выхода:
152.     bext.goto(0, HEIGHT)
153.     bext.bg(WHITE_TILE)
154.     print('Press Ctrl-C to quit.', end='')
155.
156.     sys.stdout.flush()  # (Необходимо для использующих модуль bext программ.)
157.     time.sleep(PAUSE_AMOUNT)
158.
159.
```

```
160. # Если программа не импортируется, а запускается, выполняем запуск:
161. if __name__ == '__main__':
162.     try:
163.         main()
164.     except KeyboardInterrupt:
165.         print("Langton's Ant, by Al Sweigart al@inventwithpython.com")
166.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- реализовать возможность загрузки/сохранения состояния клеток доски из текстового файла и в него;
- создать дополнительные состояния клеток доски с новыми правилами движения и посмотреть, какие виды поведения проявят себя;
- реализовать какие-нибудь из идей, предлагаемых в статье «Википедии» «Муравей Лэнгтона».

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `print(' ', end='')` в строке 149 заменить на `print('.', end='')`?
2. Что будет, если `ant['y'] += 1` в строке 106 заменить на `ant['y'] -= 1`?
3. Что будет, если `nextBoard[(ant['x'], ant['y'])] = False` в строке 79 заменить на `nextBoard[(ant['x'], ant['y'])] = True`?
4. Что будет, если `WIDTH -= 1` в строке 21 заменить на `WIDTH -= 40`?
5. Что будет, если `board = nextBoard` в строке 119 заменить на `board = board`?

40

П0г0в0рим (leetspeak)

Можно ли лучше продемонстрировать свои невероятные хакерские навыки, чем с помощью замены букв текста цифрами: m4d h4x0r 5k1llz![1] Эта программа для работы с текстом автоматически преобразует обычный текст на английском языке в leetspeak, самый крутой способ общаться в интернете. Ну или по крайней мере так было в 1993 году.

Привыкнуть к нему непросто, но, немного попрактиковавшись, вы сможете свободно читать на leetspeak. Например, `1t +@]<3s 4 w|-|1le +o g37 |_|s3|) 70, b|_|+ y0u (an 3\/3nt|_|/-\lly r3a|) l33t$peak ph1|_|3n+ly`[2]. На первых порах читать leetspeak непросто, но сама программа проста и подходит для начинающих. Больше информации о leetspeak можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Leet.

Программа в действии

Результат выполнения `leetspeak.py` выглядит следующим образом:

```
L3375P34]< (leetspeek)
By Al Sweigart al@inventwithpython.com

Enter your leet message:
> I am a leet hacker. Fear my mad skills. The 90s were over two decades ago.
```

---

[1] Mad hacker skills («невероятные хакерские навыки»). — *Примеч. пер.*

[2] Предыдущее предложение на leetspeak: It takes a while to get used to, but with some practice, you'll eventually be able to read leetspeak fluently. — *Примеч. пер.*

```
! @m a l33t h@(]<er. ph3@r my m4|) $k|ll$. +h3 90s w3r3 0ver tw0 d3(ad3$ 4g0.
(Copied leetspeak to clipboard.)
```

Описание работы

Ассоциативный массив в переменной `charMapping` в строке 36 задает соответствия обычных букв английского языка символам leetspeak. Однако, поскольку одной букве может соответствовать несколько символов leetspeak (например, букве `'t'` может соответствовать как `'7'`, так и `'+'`), каждое значение в ассоциативном массиве `charMapping` представляет собой список строковых значений. При создании нового строкового значения leetspeak программа с вероятностью 30 % использует изначальный символ из сообщения на английском языке и с вероятностью 70 % — один из нескольких символов leetspeak. Это значит, что одно и то же сообщение на английском языке может быть переведено на leetspeak несколькими способами.

```
 1. """Leet, (c) Эл Свейгарт al@inventwithpython.com
 2. Переводит сообщения на английском языке в l33t.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, слова"""
 5.
 6. import random
 7.
 8. try:
 9.     import pyperclip  # pyperclip копирует текст в буфер обмена.
10. except ImportError:
11.     pass  # Если pyperclip не установлена, ничего не делаем. Не проблема.
12.
13.
14. def main():
15.     print('''L3375P34]< (leetspeek)
16. By Al Sweigart al@inventwithpython.com
17.
18. Enter your leet message:''')
19.     english = input('> ')
20.     print()
21.     leetspeak = englishToLeetspeak(english)
22.     print(leetspeak)
23.
24.     try:
25.         # Попытка использования pyperclip, если библиотека
26.         # не импортирована, приведет к генерации исключения NameError:
27.         pyperclip.copy(leetspeak)
28.         print('(Copied leetspeak to clipboard.)')
29.     except NameError:
30.         pass  # Если pyperclip не установлена, ничего не делаем.
31.
32. def englishToLeetspeak(message):
33.     """Преобразует строковое значение на английском языке из message
```

```
34.     в leetspeak."""
35.     # Проверяем, что все ключи в `charMapping` — в нижнем регистре.
36.     charMapping = {
37.     'a': ['4', '@', '/-\\'], 'c': ['('], 'd': ['|)'], 'e': ['3'],
38.     'f': ['ph'], 'h': [']-[', '|-|'], 'i': ['1', '!', '|'], 'k': [']<'],
39.     'o': ['0'], 's': ['$', '5'], 't': ['7', '+'], 'u': ['|_|'],
40.     'v': ['\\/']}
41.     leetspeak = ''
42.     for char in message:  # Проверяем каждый символ:
43.         # Меняем символ на leetspeak с вероятностью 70 %.
44.         if char.lower() in charMapping and random.random() <= 0.70:
45.             possibleLeetReplacements = charMapping[char.lower()]
46.             leetReplacement = random.choice(possibleLeetReplacements)
47.             leetspeak = leetspeak + leetReplacement
48.         else:
49.             # Не преобразуем этот символ:
50.             leetspeak = leetspeak + char
51.     return leetspeak
52.
53.
54. # Если программа не импортируется, а запускается, производим запуск:
55. if __name__ == '__main__':
56.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- модифицировать ассоциативный массив `charMapping` для поддержки новых символов leet;
- добавить возможность перевода leet обратно на английский язык.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `return leetspeak` в строке 51 заменить на `return message`?
2. Что будет, если `char.lower()` в строке 44 заменить на `char`?
3. Что будет, если `char.lower()` в строке 44 заменить на `char.upper()`?
4. Что будет, если `leetspeak = leetspeak + leetReplacement` в строке 47 заменить на `leetspeak = leetReplacement`?

41

Под счастливой звездой

В данной игре на везение игрок бросает кости и собирает «звезды». Чем больше бросков он делает, тем больше «звезд» может собрать, но стоит собрать три «черепа» — и игрок все теряет! Эта короткая многопользовательская игра подходит для любого количества игроков, а значит, идеальна для вечеринок.

Когда ход доходит до вас, вы берете три случайные игральные кости из чашки и бросаете их. Вы можете выбросить «звезду» (star), «череп» (skull) или «вопросительный знак» (question mark). Если вы завершаете свой ход, то получаете по одному очку за выброшенную «звезду». Если же решаете бросать кости снова, то сохраняете набранные «вопросительные знаки» и берете новые кости на замену «звездам» и «черепам». Если вы собрали три «черепа», то теряете все набранные «звезды» и ход переходит к следующему игроку.

Когда один из игроков набирает 13 очков, все остальные получают по дополнительному ходу, прежде чем игра завершается. Побеждает набравший наибольшее количество очков.

В чашке лежат шесть золотых костей, четыре серебряные и три бронзовые. На золотых костях больше «звезд», на бронзовых — «черепов», а на серебряных их поровну.

Программа в действии

Результат выполнения `luckystars.py` выглядит следующим образом:

```
Lucky Stars, by Al Sweigart al@inventwithpython.com
--сокращено--
SCORES: Alice=0, Bob=0
It is Alice's turn.

+-----------+ +-----------+ +-----------+
|           | |     .     | |           |
|           | |    ,O,    | |           |
|     ?     | | 'ooOOOoo' | |     ?     |
|           | |   `OOO`   | |           |
|           | |   O' 'O   | |           |
+-----------+ +-----------+ +-----------+
     GOLD          GOLD         BRONZE
Stars collected: 1   Skulls collected: 0
Do you want to roll again? Y/N
> y

+-----------+ +-----------+ +-----------+
|     .     | |    ___    | |           |
|    ,O,    | |   /   \   | |           |
| 'ooOOOoo' | |  |() ()|  | |     ?     |
|   `OOO`   | |   \ ^ /   | |           |
|   O' 'O   | |    VVV    | |           |
+-----------+ +-----------+ +-----------+
     GOLD         BRONZE        BRONZE
Stars collected: 2   Skulls collected: 1
Do you want to roll again? Y/N
--сокращено--
```

Описание работы

Текстовая графика в этой программе хранится в виде строковых значений в списках в переменных `STAR_FACE`, `SKULL_FACE` и `QUESTION_FACE`. Подобный формат упрощает запись ее в редакторе кода, а код в строках с 154-й по 157-ю выводит ее на экран. Обратите внимание, что, поскольку все три игральные кости показываются вместе, код должен выводить на экран текст на лицевых сторонах костей по горизонтальным строкам. Если просто выполнить команду наподобие `print(STAR_FACE)`, то три кости будут выведены друг над другом, а не рядом.

```
1. """Под счастливой звездой, (c) Эл Свейгарт al@inventwithpython.com
2. Игра на везение, цель которой — набрать как можно больше "звезд" путем
3. выбрасывания костей. Можете бросать кости столько раз, сколько хотите,
4. но, если выбросите три "черепа", потеряете все набранные "звезды".
5.
```

```
Создано под впечатлением от игры Zombie Dice от Steve Jackson Games.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, игра, многопользовательская"""

import random

# Задаем константы:
GOLD = 'GOLD'
SILVER = 'SILVER'
BRONZE = 'BRONZE'

STAR_FACE = ["+-----------+",
             "|     .     |",
             "|    ,O,    |",
             "| 'ooOOOoo' |",
             "|   `OOO`   |",
             "|   O' 'O   |",
             "+-----------+"]
SKULL_FACE = ['+-----------+',
              '|    ___    |',
              '|   /   \\   |',
              '|  |() ()|  |',
              '|   \\ ^ /   |',
              '|    VVV    |',
              '+-----------+']
QUESTION_FACE = ['+-----------+',
                 '|           |',
                 '|           |',
                 '|     ?     |',
                 '|           |',
                 '|           |',
                 '+-----------+']
FACE_WIDTH = 13
FACE_HEIGHT = 7

print("""Lucky Stars, by Al Sweigart al@inventwithpython.com

A "press your luck" game where you roll dice with Stars, Skulls, and
Question Marks.

On your turn, you pull three random dice from the dice cup and roll
them. You can roll Stars, Skulls, and Question Marks. You can end your
turn and get one point per Star. If you choose to roll again, you keep
the Question Marks and pull new dice to replace the Stars and Skulls.
If you collect three Skulls, you lose all your Stars and end your turn.

When a player gets 13 points, everyone else gets one more turn before
the game ends. Whoever has the most points wins.

There are 6 Gold dice, 4 Silver dice, and 3 Bronze dice in the cup.
Gold dice have more Stars, Bronze dice have more Skulls, and Silver is
```

```
57. even.
58. """)
59.
60. print('How many players are there?')
61. while True:  # Выполняем цикл, пока пользователь не введет число.
62.     response = input('> ')
63.     if response.isdecimal() and int(response) > 1:
64.         numPlayers = int(response)
65.         break
66.     print('Please enter a number larger than 1.')
67. playerNames = []  # Список строковых значений с именами игроков.
68. playerScores = {}  # Ключами служат имена игроков, значениями — счет в виде
69.                     # целых чисел.
70. for i in range(numPlayers):
71.     while True:  # Выполняем цикл, пока пользователь не введет имя.
72.         print('What is player #' + str(i + 1) + '\'s name?')
73.         response = input('> ')
74.         if response != '' and response not in playerNames:
75.             playerNames.append(response)
76.             playerScores[response] = 0
77.             break
78.         print('Please enter a name.')
79. print()
80.
81. turn = 0  # Первый ход — игрока из playerNames[0].
82. # (!) Раскомментируйте, чтобы игрок 'Al' начал игру с тремя очками:
83. # playerScores['Al'] = 3
84. endGameWith = None
85. while True:  # Основной цикл игры.
86.     # Отображаем счет всех игроков:
87.     print()
88.     print('SCORES: ', end='')
89.     for i, name in enumerate(playerNames):
90.         print(name + ' = ' + str(playerScores[name]), end='')
91.         if i != len(playerNames) - 1:
92.             # Имена всех игроков, кроме последнего, отделены запятыми.
93.             print(', ', end='')
94.     print('\n')
95.
96.     # Изначально количество собранных "звезд" и "черепов" равно 0.
97.     stars = 0
98.     skulls = 0
99.     # В чашке — шесть золотых костей, четыре серебряные и три бронзовые:
100.    cup = ([GOLD] * 6) + ([SILVER] * 4) + ([BRONZE] * 3)
101.    hand = []  # Сначала у вас нет никаких костей.
102.    print('It is ' + playerNames[turn] + '\'s turn.')
103.    while True:  # Каждая итерация цикла соответствует броску костей.
104.        print()
105.
106.        # Убеждаемся, что в чашке осталось достаточно костей:
107.        if (3 - len(hand)) > len(cup):
```

```
            # Переход хода, поскольку костей в чашке недостаточно:
            print('There aren\'t enough dice left in the cup to '
                + 'continue ' + playerNames[turn] + '\'s turn.')
            break

        # Берем кости из чашки, пока в руках у игрока не будет три:
        random.shuffle(cup)  # Перемешиваем кости в чашке.
        while len(hand) < 3:
            hand.append(cup.pop())

        # Бросаем кости:
        rollResults = []
        for dice in hand:
            roll = random.randint(1, 6)
            if dice == GOLD:
                # Бросаем золотую кость (три "звезды", два "знака вопроса",
                ➥ один "череп"):
                if 1 <= roll <= 3:
                    rollResults.append(STAR_FACE)
                    stars += 1
                elif 4 <= roll <= 5:
                    rollResults.append(QUESTION_FACE)
                else:
                    rollResults.append(SKULL_FACE)
                    skulls += 1
            if dice == SILVER:
                # Бросаем серебряную кость (две "звезды", два "знака вопроса",
                ➥ два "черепа"):
                if 1 <= roll <= 2:
                    rollResults.append(STAR_FACE)
                    stars += 1
                elif 3 <= roll <= 4:
                    rollResults.append(QUESTION_FACE)
                else:
                    rollResults.append(SKULL_FACE)
                    skulls += 1
            if dice == BRONZE:
                # Бросаем бронзовую кость (одна "звезда", два "знака вопроса",
                ➥ три "черепа"):
                if roll == 1:
                    rollResults.append(STAR_FACE)
                    stars += 1
                elif 2 <= roll <= 4:
                    rollResults.append(QUESTION_FACE)
                else:
                    rollResults.append(SKULL_FACE)
                    skulls += 1

        # Отображаем результаты броска:
        for lineNum in range(FACE_HEIGHT):
            for diceNum in range(3):
```

```
156.                 print(rollResults[diceNum][lineNum] + ' ', end='')
157.             print()  # Выводим символ новой строки.
158.
159.         # Отображаем типы всех костей (золотая, серебряная, бронзовая):
160.         for diceType in hand:
161.             print(diceType.center(FACE_WIDTH) + ' ', end='')
162.         print()  # Выводим символ новой строки.
163.
164.         print('Stars collected:', stars, '  Skulls collected:', skulls)
165.
166.         # Проверяем, не собрал ли игрок три и более "черепа":
167.         if skulls >= 3:
168.             print('3 or more skulls means you\'ve lost your stars!')
169.             input('Press Enter to continue...')
170.             break
171.
172.         print(playerNames[turn] + ', do you want to roll again? Y/N')
173.         while True:  # Продолжаем спрашивать игрока, пока он не введет Y или N:
174.             response = input('> ').upper()
175.             if response != '' and response[0] in ('Y', 'N'):
176.                 break
177.             print('Please enter Yes or No.')
178.
179.         if response.startswith('N'):
180.             print(playerNames[turn], 'got', stars, 'stars!')
181.             # Добавляем «звезды» к общему счету этого игрока:
182.             playerScores[playerNames[turn]] += stars
183.
184.             # Check if they've reached 13 or more points:
185.             # (!) Попробуйте заменить это значение на 5 или 50 очков.
186.             if (endGameWith == None
187.                 and playerScores[playerNames[turn]] >= 13):
188.                 # Поскольку этот игрок набрал 13 очков, позволяем
189.                 # остальным игрокам сыграть еще один раунд:
190.                 print('\n\n' + ('!' * 60))
191.                 print(playerNames[turn] + ' has reached 13 points!!!')
192.                 print('Everyone else will get one more turn!')
193.                 print(('!' * 60) + '\n\n')
194.                 endGameWith = playerNames[turn]
195.             input('Press Enter to continue...')
196.             break
197.
198.         # Игнорируем "звезды" и "черепа", но сохраняем "знаки вопроса":
199.         nextHand = []
200.         for i in range(3):
201.             if rollResults[i] == QUESTION_FACE:
202.                 nextHand.append(hand[i])  # Сохраняем "знаки вопроса".
203.         hand = nextHand
204.
205.     # Ход переходит к следующему игроку:
206.     turn = (turn + 1) % numPlayers
```

```
207.
208.     # Если игра окончена, выходим из цикла:
209.     if endGameWith == playerNames[turn]:
210.         break  # Конец игры.
211.
212. print('The game has ended...')
213.
214. # Отображаем счет всех игроков:
215. print()
216. print('SCORES: ', end='')
217. for i, name in enumerate(playerNames):
218.     print(name + ' = ' + str(playerScores[name]), end='')
219.     if i != len(playerNames) - 1:
220.         # Имена всех игроков, кроме последнего, отделены запятыми.
221.         print(', ', end='')
222. print('\n')
223.
224. # Определяем победителей:
225. highestScore = 0
226. winners = []
227. for name, score in playerScores.items():
228.     if score > highestScore:
229.         # Максимальный счет — у этого игрока:
230.         highestScore = score
231.         winners = [name]  # Перезаписываем всех предыдущих победителей.
232.     elif score == highestScore:
233.         # У этого игрока — ничья с максимальным счетом.
234.         winners.append(name)
235.
236. if len(winners) == 1:
237.     # Победитель только один:
238.     print('The winner is ' + winners[0] + '!!!')
239. else:
240.     # Несколько победителей с одинаковым счетом:
241.     print('The winners are: ' + ', '.join(winners))
242.
243. print('Thanks for playing!')
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (!).

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `random.shuffle(cup)` в строке 114?
2. Что будет, если `skulls >= 3` в строке 167 заменить на `skulls > 3`?
3. Какое сообщение об ошибке вы получите, если `(turn + 1) % numPlayers` в строке 206 замените на `(turn + 1)`?
4. Что будет, если удалить или закомментировать `break` в строке 170?
5. Что будет, если `playerScores[response] = 0` в строке 76 заменить на `playerScores[response] = 10`?

42

Магический хрустальный шар

Магический хрустальный шар может предсказывать будущее и на 100 % безошибочно отвечать на вопросы, подразумевающие ответ вида «да/нет» благодаря возможностям модуля случайных чисел Python. Эта программа аналогична игрушке Magic 8 Ball, только ее не надо трясти. Вдобавок программа умеет медленно выводить на экран текстовые строки с пробелами между символами, придавая тем самым сообщениям зловещий мистический оттенок. Большая часть кода этой программы посвящена созданию мистической атмосферы. Сама же программа просто выбирает сообщение, отображаемое в ответ на случайное число.

Программа в действии

Результат выполнения `magicfortuneball.py` выглядит следующим образом:

```
M A G i C  F O R T U N E  B A L L ,  B Y  A L  S W E i G A R T

A S K  M E  Y O U R  Y E S / N O  Q U E S T i O N .

> Isn't fortune telling just a scam to trick money out of gullible people?
L E T  M E  T H i N K  O N  T H i S . . .

. . . . . .

i  H A V E  A N  A N S W E R . . .

A F F i R M A T i V E
```

Описание работы

Наш магический хрустальный шар фактически лишь отображает строковое значение, выбранное случайным образом. И полностью игнорирует вопрос пользователя. Правда, в строке 28 вызывается `input('> ')`, но полученное значение не сохраняется ни в какой переменной, поскольку программа на самом деле не задействует этот текст. Возможность вводить вопросы просто дает пользователям ощущение, что программа обладает даром ясновидения.

Функция `slowSpacePrint()` отображает текст в верхнем регистре с буквами *i* в нижнем регистре, что придает сообщению необычный вид. Функция также вставляет между символами текста пробелы, а затем отображает их в замедленном режиме, с паузами. Программа не должна уметь предсказывать будущее, чтобы быть интересной!

```
 1. """Магический хрустальный шар, (c) Эл Свейгарт al@inventwithpython.com
 2. Задавайте вопросы типа да/нет о своем будущем. Вдохновлено игрушкой Magic 8 Ball.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, юмор"""
 5.
 6. import random, time
 7.
 8.
 9. def slowSpacePrint(text, interval=0.1):
10.     """Медленно выводит текст с пробелами между буквами и буквами i
11.     в нижнем регистре."""
12.     for character in text:
13.         if character == 'I':
14.             # Ради стильности I выводятся в нижнем регистре:
15.             print('i ', end='', flush=True)
16.         else:
17.             # Все прочие символы выводятся как обычно:
18.             print(character + ' ', end='', flush=True)
19.         time.sleep(interval)
20.     print()  # Выводим два символа новой строки.
21.     print()
22.
23.
24. # Запрашиваем у пользователя вопрос:
25. slowSpacePrint('MAGIC FORTUNE BALL, BY AL SWEiGART')
26. time.sleep(0.5)
27. slowSpacePrint('ASK ME YOUR YES/NO QUESTION.')
28. input('> ')
29.
30. # Отображаем краткий ответ:
31. replies = [
32.     'LET ME THINK ON THIS...',
33.     'AN INTERESTING QUESTION...',
34.     'HMMM... ARE YOU SURE YOU WANT TO KNOW..?',
```

```
35.     'DO YOU THINK SOME THINGS ARE BEST LEFT UNKNOWN..?',
36.     'I MIGHT TELL YOU, BUT YOU MIGHT NOT LIKE THE ANSWER...',
37.     'YES... NO... MAYBE... I WILL THINK ON IT...',
38.     'AND WHAT WILL YOU DO WHEN YOU KNOW THE ANSWER? WE SHALL SEE...',
39.     'I SHALL CONSULT MY VISIONS...',
40.     'YOU MAY WANT TO SIT DOWN FOR THIS...',
41. ]
42. slowSpacePrint(random.choice(replies))
43.
44. # "Мхатовская" пауза:
45. slowSpacePrint('.' * random.randint(4, 12), 0.7)
46.
47. # Выводим ответ:
48. slowSpacePrint('I HAVE AN ANSWER...', 0.2)
49. time.sleep(1)
50. answers = [
51.     'YES, FOR SURE',
52.     'MY ANSWER IS NO',
53.     'ASK ME LATER',
54.     'I AM PROGRAMMED TO SAY YES',
55.     'THE STARS SAY YES, BUT I SAY NO',
56.     'I DUNNO MAYBE',
57.     'FOCUS AND ASK ONCE MORE',
58.     'DOUBTFUL, VERY DOUBTFUL',
59.     'AFFIRMATIVE',
60.     'YES, THOUGH YOU MAY NOT LIKE IT',
61.     'NO, BUT YOU MAY WISH IT WAS SO',
62. ]
63. slowSpacePrint(random.choice(answers), 0.05)
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- убеждаться, что вопрос пользователя оканчивается вопросительным знаком;
- добавить другие возможные ответы программы.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `random.randint(4, 12)` в строке 45 заменить на `random.randint(4, 9999)`?
2. Какое сообщение об ошибке вы получите, если `time.sleep(1)` в строке 49 замените на `time.sleep(-1)`?

43

Манкала

Настольная игра «Манкала» возникла как минимум 2000 лет назад, так что она почти столь же древняя, как и «Царская игра У́ра» из проекта 63. Она представляет собой «игру в зерна», в которой двое игроков выбирают лунки с зернами для разбрасывания по другим лункам на доске, стараясь при этом собрать как можно больше зерен в своем амбаре. Существует несколько вариантов данной игры[1] в различных культурах. Название ее происходит от арабского слова *naqala* («перемещать»).

Для игры возьмите зерна из лунки на вашей стороне доски и положите по одному в каждую из следующих лунок в направлении против часовой стрелки, пропуская амбар вашего противника. Если ваше последнее зерно попадает в одну из ваших пустых лунок, то перенесите зерна из противоположной лунки в эту. Если последнее из помещенных зерен находится в вашем амбаре, то вы получаете ход вне очереди.

Игра заканчивается, когда все лунки одного из игроков опустеют. Второй игрок при этом забирает все оставшиеся зерна в свой амбар, и победителем считается тот, у кого больше зерен. Больше информации об игре «Манкала» можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Манкала.

---

[1] Собственно говоря, не существует какой-то основной игры с вариантами, манкала — обширнейшее семейство игр, распространенных по всей Африке и Азии. — *Примеч. пер.*

Программа в действии

Результат выполнения `mancala.py` выглядит следующим образом:

```
Mancala, by Al Sweigart al@inventwithpython.com
--сокращено--

+------+------+--<<<<<-Player 2----+------+------+------+
2      |G     |H     |I     |J     |K     |L     |      1
       |   4  |   4  |   4  |   4  |   4  |   4  |
S      |      |      |      |      |      |      |      S
T   0  +------+------+------+------+------+------+   0  T
O      |A     |B     |C     |D     |E     |F     |      O
R      |   4  |   4  |   4  |   4  |   4  |   4  |      R
E      |      |      |      |      |      |      |      E
+------+------+------+-Player 1->>>>>-----+------+------+

Player 1, choose move: A-F (or QUIT)
> f

+------+------+--<<<<<-Player 2----+------+------+------+
2      |G     |H     |I     |J     |K     |L     |      1
       |   4  |   4  |   4  |   5  |   5  |   5  |
S      |      |      |      |      |      |      |      S
T   0  +------+------+------+------+------+------+   1  T
O      |A     |B     |C     |D     |E     |F     |      O
R      |   4  |   4  |   4  |   4  |   4  |   0  |      R
E      |      |      |      |      |      |      |      E
+------+------+------+-Player 1->>>>>-----+------+------+
Player 2, choose move: G-L (or QUIT)
--сокращено--
```

Описание работы

Для рисования доски в программе «Манкала» мы используем ASCII-графику. Обратите внимание, что для каждой лунки нужно отображать не только количество зерен в ней, но и метку. Во избежание недоразумений метками служат буквы от `A` до `L`, чтобы не путать их с количеством зерен в лунках. Ассоциативные массивы `NEXT_PIT` и `OPPOSITE_PIT` задают соответствие букв одной лунки и следующей/противоположной ей лунки. В результате выражение `NEXT_PIT['A']` равно `'B'`, а выражение `OPPOSITE_PIT['A']` равно `'G'`. Обратите внимание на то, как эти ассоциативные массивы используются в коде. Без них наша программа «Манкала» для реализации тех же шагов игры потребовала бы длинных цепочек операторов `if` и `elif`.

```
1. """Манкала, (c) Эл Свейгарт al@inventwithpython.com
2. Древняя игра в зерна
3. Код размещен на https://nostarch.com/big-book-small-python-projects
```

```
 4. Теги: большая, настольная игра, игра, для двух игроков"""
 5.
 6. import sys
 7.
 8. # Кортежи лунок игроков:
 9. PLAYER_1_PITS = ('A', 'B', 'C', 'D', 'E', 'F')
10. PLAYER_2_PITS = ('G', 'H', 'I', 'J', 'K', 'L')
11. # Ассоциативный массив, ключами которого служат лунки, а значениями —
12. # противоположные им лунки:
13. OPPOSITE_PIT = {'A': 'G', 'B': 'H', 'C': 'I', 'D': 'J', 'E': 'K',
14.                   'F': 'L', 'G': 'A', 'H': 'B', 'I': 'C', 'J': 'D',
15.                   'K': 'E', 'L': 'F'}
16. # Ассоциативный массив, ключами которого служат лунки, а значениями —
17. # следующие по порядку лунки:
18. NEXT_PIT = {'A': 'B', 'B': 'C', 'C': 'D', 'D': 'E', 'E': 'F', 'F': '1',
19.             '1': 'L', 'L': 'K', 'K': 'J', 'J': 'I', 'I': 'H', 'H': 'G',
20.             'G': '2', '2': 'A'}
21.
22. # Метки всех лунок в порядке против часовой стрелки, начиная с A:
23. PIT_LABELS = 'ABCDEF1LKJIHG2'
24.
25. # Количество зерен в каждой из лунок в начале новой игры:
26. STARTING_NUMBER_OF_SEEDS = 4 # (!) Попробуйте заменить это значение на 1 или 10
27.
28.
29. def main():
30.     print('''Mancala, by Al Sweigart al@inventwithpython.com
31.
32. The ancient two-player seed-sowing game. Grab the seeds from a pit on
33. your side and place one in each following pit, going counterclockwise
34. and skipping your opponent's store. If your last seed lands in an empty
35. pit of yours, move the opposite pit's seeds into that pit. The
36. goal is to get the most seeds in your store on the side of the board.
37. If the last placed seed is in your store, you get a free turn.
38.
39. The game ends when all of one player's pits are empty. The other player
40. claims the remaining seeds for their store, and the winner is the one
41. with the most seeds.
42.
43. More info at https://en.wikipedia.org/wiki/Mancala
44. ''')
45.     input('Press Enter to begin...')
46.
47.     gameBoard = getNewBoard()
48.     playerTurn = '1'  # Сначала ходит игрок 1.
49.
50.     while True:  # Обрабатываем ход одного из игроков.
51.         # Очищаем экран, выводя множество символов новой строки,
52.         # чтобы убрать с него старую доску.
53.         print('\n' * 60)
54.         # Отображаем доску и получаем ход игрока:
55.         displayBoard(gameBoard)
```

```
        playerMove = askForPlayerMove(playerTurn, gameBoard)

        # Осуществляем ход игрока:
        playerTurn = makeMove(gameBoard, playerTurn, playerMove)

        # Проверяем, не закончилась ли игра и не выиграл ли игрок:
        winner = checkForWinner(gameBoard)
        if winner == '1' or winner == '2':
            displayBoard(gameBoard)  # Отображаем доску последний раз.
            print('Player ' + winner + ' has won!')
            sys.exit()
        elif winner == 'tie':
            displayBoard(gameBoard)  # Отображаем доску последний раз.
            print('There is a tie!')
            sys.exit()


def getNewBoard():
    """Возвращает ассоциативный массив, соответствующий доске "Манкалы"
    в начальном состоянии: четыре зерна в каждой лунке и ноль в амбарах."""

    # Синтаксический сахар — используем сокращенное имя переменной:
    s = STARTING_NUMBER_OF_SEEDS

    # Создаем структуру данных для доски с 0 зерен в амбарах
    # и начальным количеством зерен в лунках:
    return {'1': 0, '2': 0, 'A': s, 'B': s, 'C': s, 'D': s, 'E': s,
            'F': s, 'G': s, 'H': s, 'I': s, 'J': s, 'K': s, 'L': s}


def displayBoard(board):
    """Отображает доску в виде ASCII-графики на основе
    ассоциативного массива board."""

    seedAmounts = []
    # Это строковое значение 'GHIJKL21ABCDEF' описывает порядок
    # лунок слева направо и сверху вниз:
    for pit in 'GHIJKL21ABCDEF':
        numSeedsInThisPit = str(board[pit]).rjust(2)
        seedAmounts.append(numSeedsInThisPit)

    print("""
+------+------+--<<<<<-Player 2----+------+------+------+
2      |G     |H     |I     |J     |K     |L     |      1
       |  {}  |  {}  |  {}  |  {}  |  {}  |  {}  |
S      |      |      |      |      |      |      |      S
T  {}  +------+------+------+------+------+------+  {}  T
O      |A     |B     |C     |D     |E     |F     |      O
R      |  {}  |  {}  |  {}  |  {}  |  {}  |  {}  |      R
E      |      |      |      |      |      |      |      E
+------+------+------+-Player 1->>>>>-----+------+------+
```

```
107.
108. """.format(*seedAmounts))
109.
110. def askForPlayerMove(playerTurn, board):
111.     """Спрашиваем игрока, из какой лунки на его стороне доски
112.     он хочет сеять зерна. Возвращаем метку выбранной лунки в верхнем
113.     регистре в виде строкового значения."""
114.
115.     while True:  # Продолжаем спрашивать игрока, пока он не введет
116.                  # допустимый ход.
117.         # Просим игрока ввести лунку на его стороне:
118.         if playerTurn == '1':
119.             print('Player 1, choose move: A-F (or QUIT)')
120.         elif playerTurn == '2':
121.             print('Player 2, choose move: G-L (or QUIT)')
122.         response = input('> ').upper().strip()
123.
124.         # Проверяем, не хочет ли игрок выйти из игры:
125.         if response == 'QUIT':
126.             print('Thanks for playing!')
127.             sys.exit()
128.
129.         # Проверяем, выбрана ли допустимая лунка:
130.         if (playerTurn == '1' and response not in PLAYER_1_PITS) or (
131.             playerTurn == '2' and response not in PLAYER_2_PITS
132.         ):
133.             print('Please pick a letter on your side of the board.')
134.             continue  # Снова просим игрока сделать ход.
135.         if board.get(response) == 0:
136.             print('Please pick a non-empty pit.')
137.             continue  # Снова просим игрока сделать ход.
138.         return response
139.
140.
141. def makeMove(board, playerTurn, pit):
142.     """Модифицирует структуру данных board в соответствии с выбором
143.     игроком 1 или 2 при его ходе лунки — источника засеиваемых зерен.
144.     Возвращает '1' or '2' в зависимости от того, чей ход следующий."""
145.
146.     seedsToSow = board[pit]  # Получаем количество зерен в выбранной лунке.
147.     board[pit] = 0  # Опустошаем выбранную лунку.
148.
149.     while seedsToSow > 0:  # Сеем, пока зерна не закончатся.
150.         pit = NEXT_PIT[pit]  # Переходим к следующей лунке.
151.         if (playerTurn == '1' and pit == '2') or (
152.             playerTurn == '2' and pit == '1'
153.         ):
154.             continue  # Пропускаем амбар противника.
155.         board[pit] += 1
156.         seedsToSow -= 1
157.
```

```
158.     # Если последнее зерно попало в амбар текущего игрока, он ходит снова.
159.     if (pit == playerTurn == '1') or (pit == playerTurn == '2'):
160.         # Последнее зерно попало в амбар текущего игрока; он ходит еще раз.
161.         return playerTurn
162.     # Проверяем, попало ли последнее зерно в пустую лунку; в этом случае
163.     # захватываем зерна из противоположной лунки.
164.     if playerTurn == '1' and pit in PLAYER_1_PITS and board[pit] == 1:
165.         oppositePit = OPPOSITE_PIT[pit]
166.         board['1'] += board[oppositePit]
167.         board[oppositePit] = 0
168.     elif playerTurn == '2' and pit in PLAYER_2_PITS and board[pit] == 1:
169.         oppositePit = OPPOSITE_PIT[pit]
170.         board['2'] += board[oppositePit]
171.         board[oppositePit] = 0
172.
173.     # Возвращаем номер другого игрока как следующего:
174.     if playerTurn == '1':
175.         return '2'
176.     elif playerTurn == '2':
177.         return '1'
178.
179.
180. def checkForWinner(board):
181.     """Изучаем доску и возвращаем '1' или '2', если один из игроков
182.     победил, либо 'tie' или 'no winner', если нет. Игра заканчивается,
183.     когда все лунки одного игрока пусты; второй игрок забирает все
184.     оставшиеся зерна в свой амбар. Победитель — тот, у кого больше зерен."""
185.
186.     player1Total = board['A'] + board['B'] + board['C']
187.     player1Total += board['D'] + board['E'] + board['F']
188.     player2Total = board['G'] + board['H'] + board['I']
189.     player2Total += board['J'] + board['K'] + board['L']
190.
191.     if player1Total == 0:
192.         # Игрок 2 получает все оставшиеся зерна на стороне противника:
193.         board['2'] += player2Total
194.         for pit in PLAYER_2_PITS:
195.             board[pit] = 0  # Обнуляем все лунки.
196.     elif player2Total == 0:
197.         # Игрок 1 получает все оставшиеся зерна на стороне противника:
198.         board['1'] += player1Total
199.         for pit in PLAYER_1_PITS:
200.             board[pit] = 0  # Обнуляем все лунки.
201.     else:
202.         return 'no winner'  # Никто пока не выиграл.
203.
204.     # Игра закончена, ищем игрока с максимальным счетом.
205.     if board['1'] > board['2']:
206.         return '1'
207.     elif board['2'] > board['1']:
208.         return '2'
```

```
209.     else:
210.         return 'tie'
211.
212.
213. # Если программа не импортируется, а запускается, производим запуск:
214. if __name__ == '__main__':
215.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- изменить количество лунок на доске;
- выбирать случайным образом «бонусную» лунку, которая при попадании в нее последнего зерна дает игроку еще один ход вне очереди;
- создать квадратную доску, рассчитанную на четырех игроков вместо двух.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `return '2'` в строке 175 заменить на `return '1'`?
2. Что будет, если `return '2'` в строке 208 заменить на `return '1'`?
3. Что будет, если `response == 'QUIT'` в строке 125 заменить на `response == 'quit'`?
4. Что будет, если `board[pit] = 0` в строке 147 заменить на `board[pit] = 1`?
5. Что будет, если `print('\n' * 60)` в строке 53 заменить на `print('\n' * 0)`?
6. Что будет, если `playerTurn = '1'` в строке 48 заменить на `playerTurn = '2'`?
7. Что будет, если выражение `board.get(response) == 0` в строке 135 заменить на `board.get(response) == -1`?

44

Бегущий в лабиринте 2D

В этой двумерной игре на прохождение лабиринта игрок видит как бы сверху, с высоты птичьего полета файл лабиринта, созданного вами в текстовом редакторе, например IDE, в котором вы набираете свои файлы `.py`. С помощью клавиш `WASD` игрок может перемещать символ `@` вверх, влево, вниз и вправо, к выходу, обозначенному символом `X`.

Для создания файла лабиринта откройте текстовый редактор и введите следующий узор. Не вводите числа сверху и слева, они просто для справки:

```
123456789
1#########
2#S# # # #
3#########
4# # # # #
5#########
6# # # # #
7#########
8# # # #E#
9#########
```

Символы `#` обозначают стены, `S` — начало, а `E` — выход из лабиринта. Выделенные жирным шрифтом символы **`#`** отмечают стены, которые можно убрать для формирования лабиринта. Не убирайте стены в нечетных столбцах и нечетных строках и не убирайте границы лабиринта. Закончив, сохраните лабиринт в виде текстового файла `.txt`. Он должен выглядеть примерно следующим образом:

```
#########
#S    # #
# ### # #
# #   # #
# ##### #
#   #   #
### # # #
#     #E#
#########
```

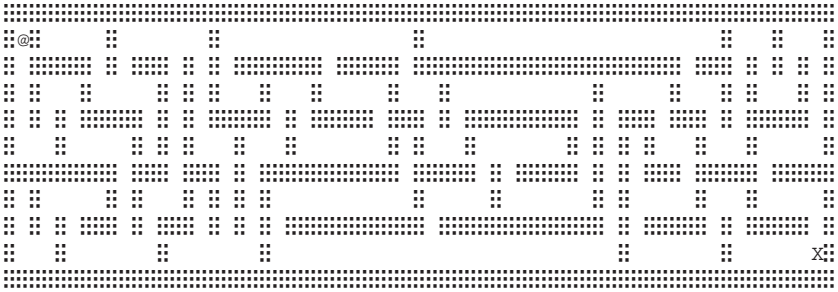
Конечно, это совсем простой лабиринт. Вы можете создавать лабиринты любого размера, лишь бы в них было нечетное число строк и столбцов. Убедитесь только, что он помещается на экран! Скачать файлы лабиринтов можно также по адресу https://invpy.com/mazes/.

Программа в действии

Результат выполнения `mazerunner2d.py` выглядит следующим образом:

```
Maze Runner 2D, by Al Sweigart al@inventwithpython.com

(Maze files are generated by mazemakerrec.py)
Enter the filename of the maze (or LIST or QUIT):
> maze65x11s1.txt
```



```
                          W
Enter direction, or QUIT: ASD
--сокращено--
```

Описание работы

Программа загружает данные о стенках лабиринта из текстового файла в ассоциативный массив, хранимый в переменной `maze`. Ключами в этом ассоциативном массиве служат кортежи `(x, y)`, а значениями — строковые значения констант `WALL`, `EMPTY`, `START` и `EXIT`. Аналогичное представление лабиринта используется в проекте 45. Разница между проектами заключается в коде отображения лабиринта на

экране. Поскольку «Бегущий в лабиринте 2D» проще, я рекомендую сначала разобраться с этой программой, прежде чем переходить к «Бегущему в лабиринте 3D».

```
"""Бегущий в лабиринте 2D, (c) Эл Свейгарт al@inventwithpython.com
Перемещайтесь по лабиринту и попытайтесь выбраться из него.
Файлы лабиринтов сгенерированы с помощью сценария mazemakerrec.py.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, игра, лабиринт"""

import sys, os

# Константы для файла лабиринта:
WALL = '#'
EMPTY = ' '
START = 'S'
EXIT = 'E'

PLAYER = '@'  # (!) Попробуйте заменить это значение на '+' или 'o'.
BLOCK = chr(9617)  # Символ 9617 соответствует '⣿'


def displayMaze(maze):
    # Отображаем лабиринт на экране:
    for y in range(HEIGHT):
        for x in range(WIDTH):
            if (x, y) == (playerx, playery):
                print(PLAYER, end='')
            elif (x, y) == (exitx, exity):
                print('X', end='')
            elif maze[(x, y)] == WALL:
                print(BLOCK, end='')
            else:
                print(maze[(x, y)], end='')
        print()  # Выводим символ новой строки после отображения строки.


print('''Maze Runner 2D, by Al Sweigart al@inventwithpython.com

(Maze files are generated by mazemakerrec.py)''')

# Запрашиваем у пользователя название файла с лабиринтом:
while True:
    print('Enter the filename of the maze (or LIST or QUIT):')
    filename = input('> ')

    # Выводим названия всех файлов лабиринтов в текущем каталоге:
    if filename.upper() == 'LIST':
        print('Maze files found in', os.getcwd())
        for fileInCurrentFolder in os.listdir():
            if (fileInCurrentFolder.startswith('maze') and
            fileInCurrentFolder.endswith('.txt')):
```

```
49.                 print('  ', fileInCurrentFolder)
50.         continue
51.
52.     if filename.upper() == 'QUIT':
53.         sys.exit()
54.
55.     if os.path.exists(filename):
56.         break
57.     print('There is no file named', filename)
58.
59. # Загружаем лабиринт из файла:
60. mazeFile = open(filename)
61. maze = {}
62. lines = mazeFile.readlines()
63. playerx = None
64. playery = None
65. exitx = None
66. exity = None
67. y = 0
68. for line in lines:
69.     WIDTH = len(line.rstrip())
70.     for x, character in enumerate(line.rstrip()):
71.         assert character in (WALL, EMPTY, START, EXIT), 'Invalid character
            at column {}, line {}'.format(x + 1, y + 1)
72.         if character in (WALL, EMPTY):
73.             maze[(x, y)] = character
74.         elif character == START:
75.             playerx, playery = x, y
76.             maze[(x, y)] = EMPTY
77.         elif character == EXIT:
78.             exitx, exity = x, y
79.             maze[(x, y)] = EMPTY
80.     y += 1
81. HEIGHT = y
82.
83. assert playerx != None and playery != None, 'No start in maze file.'
84. assert exitx != None and exity != None, 'No exit in maze file.'
85.
86. while True:  # Основной цикл игры.
87.     displayMaze(maze)
88.
89.     while True:  # Получаем ход пользователя.
90.         print('                            W')
91.         print('Enter direction, or QUIT: ASD')
92.         move = input('> ').upper()
93.
94.         if move == 'QUIT':
95.             print('Thanks for playing!')
96.             sys.exit()
97.
98.         if move not in ['W', 'A', 'S', 'D']:
```

```
 99.             print('Invalid direction. Enter one of W, A, S, or D.')
100.             continue
101.
102.         # Проверяем, может ли игрок идти в этом направлении:
103.         if move == 'W' and maze[(playerx, playery - 1)] == EMPTY:
104.             break
105.         elif move == 'S' and maze[(playerx, playery + 1)] == EMPTY:
106.             break
107.         elif move == 'A' and maze[(playerx - 1, playery)] == EMPTY:
108.             break
109.         elif move == 'D' and maze[(playerx + 1, playery)] == EMPTY:
110.             break
111.
112.         print('You cannot move in that direction.')
113.
114.     # Продолжаем идти в этом направлении, пока не наткнемся на развилку.
115.     if move == 'W':
116.         while True:
117.             playery -= 1
118.             if (playerx, playery) == (exitx, exity):
119.                 break
120.             if maze[(playerx, playery - 1)] == WALL:
121.                 break  # Выходим, если наткнулись на стену.
122.             if (maze[(playerx - 1, playery)] == EMPTY
123.                 or maze[(playerx + 1, playery)] == EMPTY):
124.                 break  # Выходим, если достигли развилки.
125.     elif move == 'S':
126.         while True:
127.             playery += 1
128.             if (playerx, playery) == (exitx, exity):
129.                 break
130.             if maze[(playerx, playery + 1)] == WALL:
131.                 break  # Выходим, если наткнулись на стену.
132.             if (maze[(playerx - 1, playery)] == EMPTY
133.                 or maze[(playerx + 1, playery)] == EMPTY):
134.                 break  # Выходим, если достигли развилки.
135.     elif move == 'A':
136.         while True:
137.             playerx -= 1
138.             if (playerx, playery) == (exitx, exity):
139.                 break
140.             if maze[(playerx - 1, playery)] == WALL:
141.                 break  # Выходим, если наткнулись на стену.
142.             if (maze[(playerx, playery - 1)] == EMPTY
143.                 or maze[(playerx, playery + 1)] == EMPTY):
144.                 break  # Выходим, если достигли развилки.
145.     elif move == 'D':
146.         while True:
147.             playerx += 1
148.             if (playerx, playery) == (exitx, exity):
149.                 break
```

```
150.             if maze[(playerx + 1, playery)] == WALL:
151.                 break  # Выходим, если наткнулись на стену.
152.             if (maze[(playerx, playery - 1)] == EMPTY
153.                 or maze[(playerx, playery + 1)] == EMPTY):
154.                 break  # Выходим, если достигли развилки.
155.
156.     if (playerx, playery) == (exitx, exity):
157.         displayMaze(maze)
158.         print('You have reached the exit! Good job!')
159.         print('Thanks for playing!')
160.         sys.exit()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если `character == START` в строке 74 замените на `character == EXIT`?
2. Что будет, если `playery + 1` в строке 105 заменить на `playery - 1`?
3. Что будет, если `(exitx, exity)` в строке 156 заменить на `(None, None)`?
4. Какое сообщение об ошибке вы получите, если `while True:` в строке 89 замените на `while False:`?
5. Что будет, если `break` в строке 104 заменить на `continue`?
6. Какое сообщение об ошибке вы получите, если `break` в строке 121 замените на `continue`?

45

Бегущий в лабиринте 3D

В этой трехмерной игре на прохождение лабиринта игрок видит лабиринт как бы своими глазами, изнутри. Попробуйте выбраться из него! Вы можете сгенерировать файлы лабиринтов, следуя инструкциям из проекта 44, или просто скачать их по адресу https://invpy.com/mazes/.

Программа в действии

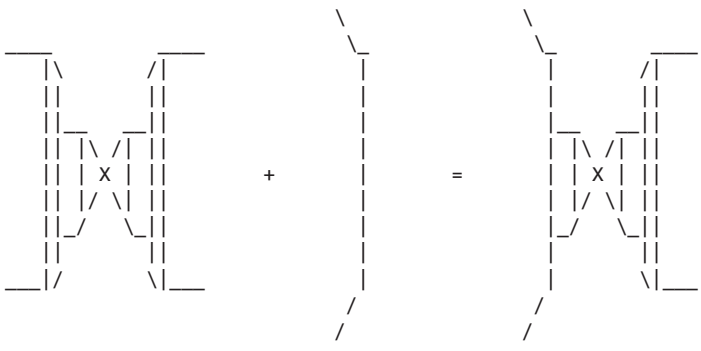
Результат выполнения `mazerunner3d.py` выглядит следующим образом:

```
Maze Runner 3D, by Al Sweigart al@inventwithpython.com
(Maze files are generated by mazemakerrec.py)
Enter the filename of the maze (or LIST or QUIT):
> maze75x11s1.txt
```

```
░░░░░░░░░░░░░░░░░░
░  \           / ░
░   \_________/  ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |       |   ░
░    |_______|   ░
░   /         \  ░
░  /           \ ░
░░░░░░░░░░░░░░░░░░
Location (1, 1)  Direction: NORTH
                   (W)
Enter direction: (A) (D)  or QUIT.
> d
░░░░░░░░░░░░░░░░░░░
░  \              ░
░   \_____________░
░    |            ░
░    |            ░
░    |            ░
░    |            ░
░    |            ░
░    |            ░
░    |            ░
░    |            ░
░    |____________░
░   /             ░
░  /              ░
░░░░░░░░░░░░░░░░░░░
Location (1, 1)  Direction: EAST
--сокращено--
```

Описание работы

В основе нашей ASCII-графики в трехмерной перспективе лежит многострочное строковое значение, хранящееся в `ALL_OPEN`. Это строковое значение изображает позицию, в которой никакие стены не преграждают ему пути. Далее программа отрисовывает поверх многострочного строкового значения `ALL_OPEN` стены, хранящиеся в ассоциативном массиве `CLOSED`, генерируя таким образом ASCII-графику для любых возможных сочетаний заблокированных стенами путей. Например, вот как наша программа генерирует вид, при котором стена располагается слева от игрока:



Точки из ASCII-графики в исходном коде удаляются перед выводом строковых значений на экран; они нужны только для упрощения ввода кода, чтобы не вставлять и не оставлять пустого пространства.

Вот исходный код нашего трехмерного лабиринта:

```
"""Трехмерный лабиринт, (c) Эл Свейгарт al@inventwithpython.com
Перемещайтесь по лабиринту и попытайтесь выбраться из него... в 3D!
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: очень большая, графика, игра, лабиринт"""

import copy, sys, os

# Задаем константы:
WALL = '#'
EMPTY = ' '
START = 'S'
EXIT = 'E'
BLOCK = chr(9617)  # Символ 9617 соответствует '⣿'
NORTH = 'NORTH'
SOUTH = 'SOUTH'
EAST = 'EAST'
WEST = 'WEST'


def wallStrToWallDict(wallStr):
    """Получает на входе строковое представление изображения стены
    (например, такое, как в ALL_OPEN и CLOSED) и возвращает ее представление
    в виде ассоциативного массива с кортежами (x, y) в роли ключей
    и строковых значений из одного символа для вывода на этой позиции x, y."""
    wallDict = {}
    height = 0
    width = 0
    for y, line in enumerate(wallStr.splitlines()):
        if y > height:
            height = y
        for x, character in enumerate(line):
            if x > width:
```

```
33.                 width = x
34.             wallDict[(x, y)] = character
35.     wallDict['height'] = height + 1
36.     wallDict['width'] = width + 1
37.     return wallDict
38.
39. EXIT_DICT = {(0, 0): 'E', (1, 0): 'X', (2, 0): 'I',
40.              (3, 0): 'T', 'height': 1, 'width': 4}
41.
42. # Строковые значения для отображения мы создаем путем преобразования
43. # изображений в этих многострочных строковых значениях с помощью
44. # wallStrToWallDict(). После чего формируем стенку для позиции
45. # и направления движения игрока путем вставки ассоциативных массивов
46. # стенок из CLOSED поверх ассоциативного массива стенки из ALL_OPEN.
47.
48. ALL_OPEN = wallStrToWallDict(r'''
49. .................
50. ____.........____
51. ...|\......./|...
52. ...||.......||...
53. ...||__...__||...
54. ...||.|\./|.||...
55. ...||.|.X.|.||...
56. ...||.|/.\|.||...
57. ...||_/...\_||...
58. ...||.......||...
59. ___|/.......\|___
60. .................
61. .................'''.strip())
62. # Для удаления символа новой строки в начале этого многострочного
63. # строкового значения используется вызов strip().
64.
65. CLOSED = {}
66. CLOSED['A'] = wallStrToWallDict(r'''
67. _____
68. .....
69. .....
70. .....
71. _____'''.strip()) # Вставляем на позиции 6, 4.
72.
73. CLOSED['B'] = wallStrToWallDict(r'''
74. .\.
75. ..\
76. ...
77. ...
78. ...
79. ../
80. ./.'''.strip()) # Вставляем на позиции 4, 3.
81.
82. CLOSED['C'] = wallStrToWallDict(r'''
83. ___________
84. ...........
```

```
 85. ..........
 86. ..........
 87. ..........
 88. ..........
 89. ..........
 90. ..........
 91. ..........
 92. ___________'''.strip()) # Вставляем на позиции 3, 1.
 93.
 94. CLOSED['D'] = wallStrToWallDict(r'''
 95. ./.
 96. /..
 97. ...
 98. ...
 99. ...
100. \..
101. .\.'''.strip()) # Вставляем на позиции 10, 3.
102.
103. CLOSED['E'] = wallStrToWallDict(r'''
104. ..\..
105. ...\_
106. ....|
107. ....|
108. ....|
109. ....|
110. ....|
111. ....|
112. ....|
113. ....|
114. ....|
115. .../.
116. ../..'''.strip()) # Вставляем на позиции 0, 0.
117.
118. CLOSED['F'] = wallStrToWallDict(r'''
119. ../..
120. _/...
121. |....
122. |....
123. |....
124. |....
125. |....
126. |....
127. |....
128. |....
129. |....
130. .\...
131. ..\..'''.strip()) # Вставляем на позиции 12, 0.
132.
133. def displayWallDict(wallDict):
134.     """Отображаем на экране ассоциативный массив стенки,
135.     возвращаемый wallStrToWallDict()."""
```

```
136.     print(BLOCK * (wallDict['width'] + 2))
137.     for y in range(wallDict['height']):
138.         print(BLOCK, end='')
139.         for x in range(wallDict['width']):
140.             wall = wallDict[(x, y)]
141.             if wall == '.':
142.                 wall = ' '
143.             print(wall, end='')
144.         print(BLOCK)  # Выводим блок с символом новой строки.
145.     print(BLOCK * (wallDict['width'] + 2))
146.
147.
148. def pasteWallDict(srcWallDict, dstWallDict, left, top):
149.     """Копируем ассоциативный массив представления стенки из
150.     srcWallDict поверх dstWallDict, смещенный до позиции left, top."""
151.     dstWallDict = copy.copy(dstWallDict)
152.     for x in range(srcWallDict['width']):
153.         for y in range(srcWallDict['height']):
154.             dstWallDict[(x + left, y + top)] = srcWallDict[(x, y)]
155.     return dstWallDict
156.
157.
158. def makeWallDict(maze, playerx, playery, playerDirection, exitx, exity):
159.     """Создаем ассоциативный массив представления стенки в соответствии
160.     с расположением и направлением движения игрока в лабиринте (с выходом
161.     в точке exitx, exity) путем вставки ассоциативных массивов
162.     стенок поверх ALL_OPEN и возвращаем его."""
163.     # "Сегменты" A – F (относительно направления движения игрока)
164.     # показывают, какие стены лабиринта необходимо проверить
165.     # на предмет необходимости добавить их поверх создаваемого нами
166.     # ассоциативного массива представления стенок.
167.     if playerDirection == NORTH:
168.         # Карта сегментов относительно  A
169.         # игрока @:                     BCD (игрок смотрит на север)
170.         #                               E@F
171.         offsets = (('A', 0, -2), ('B', -1, -1), ('C', 0, -1),
172.                    ('D', 1, -1), ('E', -1, 0), ('F', 1, 0))
173.     if playerDirection == SOUTH:
174.         # Карта сегментов относительно F@E
175.         # игрока @:                     DCB (игрок смотрит на юг)
176.         #                                A
177.         offsets = (('A', 0, 2), ('B', 1, 1), ('C', 0, 1),
178.                    ('D', -1, 1), ('E', 1, 0), ('F', -1, 0))
179.     if playerDirection == EAST:
180.         # Карта сегментов относительно EB
181.         # игрока @:                     @CA (игрок смотрит на восток)
182.         #                               FD
183.         offsets = (('A', 2, 0), ('B', 1, -1), ('C', 1, 0),
184.                    ('D', 1, 1), ('E', 0, -1), ('F', 0, 1))
185.     if playerDirection == WEST:
186.         # Карта сегментов относительно  DF
```

```
187.         # игрока @:                     AC@ (игрок смотрит на запад)
188.         #                                BE
189.         offsets = (('A', -2, 0), ('B', -1, 1), ('C', -1, 0),
190.                    ('D', -1, -1), ('E', 0, 1), ('F', 0, -1))
191.
192.     section = {}
193.     for sec, xOff, yOff in offsets:
194.         section[sec] = maze.get((playerx + xOff, playery + yOff), WALL)
195.         if (playerx + xOff, playery + yOff) == (exitx, exity):
196.             section[sec] = EXIT
197.
198.     wallDict = copy.copy(ALL_OPEN)
199.     PASTE_CLOSED_TO = {'A': (6, 4), 'B': (4, 3), 'C': (3, 1),
200.                        'D': (10, 3), 'E': (0, 0), 'F': (12, 0)}
201.     for sec in 'ABDCEF':
202.         if section[sec] == WALL:
203.             wallDict = pasteWallDict(CLOSED[sec], wallDict,
204.                 PASTE_CLOSED_TO[sec][0], PASTE_CLOSED_TO[sec][1])
205.
206.     # Рисуем знак EXIT при необходимости:
207.     if section['C'] == EXIT:
208.         wallDict = pasteWallDict(EXIT_DICT, wallDict, 7, 9)
209.     if section['E'] == EXIT:
210.         wallDict = pasteWallDict(EXIT_DICT, wallDict, 0, 11)
211.     if section['F'] == EXIT:
212.         wallDict = pasteWallDict(EXIT_DICT, wallDict, 13, 11)
213.
214.     return wallDict
215.
216.
217. print('Maze Runner 3D, by Al Sweigart al@inventwithpython.com')
218. print('(Maze files are generated by mazemakerrec.py)')
219.
220. # Получаем от пользователя название файла лабиринта:
221. while True:
222.     print('Enter the filename of the maze (or LIST or QUIT):')
223.     filename = input('> ')
224.
225.     # Выводим список всех файлов лабиринтов в текущем каталоге:
226.     if filename.upper() == 'LIST':
227.         print('Maze files found in', os.getcwd())
228.         for fileInCurrentFolder in os.listdir():
229.             if (fileInCurrentFolder.startswith('maze')
230.             and fileInCurrentFolder.endswith('.txt')):
231.                 print('  ', fileInCurrentFolder)
232.         continue
233.
234.     if filename.upper() == 'QUIT':
235.         sys.exit()
236.
237.     if os.path.exists(filename):
```

```
238.         break
239.     print('There is no file named', filename)
240.
241. # Загружаем лабиринт из файла:
242. mazeFile = open(filename)
243. maze = {}
244. lines = mazeFile.readlines()
245. px = None
246. py = None
247. exitx = None
248. exity = None
249. y = 0
250. for line in lines:
251.     WIDTH = len(line.rstrip())
252.     for x, character in enumerate(line.rstrip()):
253.         assert character in (WALL, EMPTY, START, EXIT), 'Invalid character
             at column {}, line {}'.format(x + 1, y + 1)
254.         if character in (WALL, EMPTY):
255.             maze[(x, y)] = character
256.         elif character == START:
257.             px, py = x, y
258.             maze[(x, y)] = EMPTY
259.         elif character == EXIT:
260.             exitx, exity = x, y
261.             maze[(x, y)] = EMPTY
262.     y += 1
263. HEIGHT = y
264.
265. assert px != None and py != None, 'No start point in file.'
266. assert exitx != None and exity != None, 'No exit point in file.'
267. pDir = NORTH
268.
269.
270. while True:  # Основной цикл игры.
271.     displayWallDict(makeWallDict(maze, px, py, pDir, exitx, exity))
272.
273.     while True: # Получаем ход пользователя.
274.         print('Location ({}, {})  Direction: {}'.format(px, py, pDir))
275.         print('                   (W)')
276.         print('Enter direction: (A) (D)  or QUIT.')
277.         move = input('> ').upper()
278.
279.         if move == 'QUIT':
280.             print('Thanks for playing!')
281.             sys.exit()
282.
283.         if (move not in ['F', 'L', 'R', 'W', 'A', 'D']
284.             and not move.startswith('T')):
285.             print('Please enter one of F, L, or R (or W, A, D).')
286.             continue
287.
```

```
288.         # Передвигаем игрока в соответствии с желаемым ходом:
289.         if move == 'F' or move == 'W':
290.             if pDir == NORTH and maze[(px, py - 1)] == EMPTY:
291.                 py -= 1
292.                 break
293.             if pDir == SOUTH and maze[(px, py + 1)] == EMPTY:
294.                 py += 1
295.                 break
296.             if pDir == EAST and maze[(px + 1, py)] == EMPTY:
297.                 px += 1
298.                 break
299.             if pDir == WEST and maze[(px - 1, py)] == EMPTY:
300.                 px -= 1
301.                 break
302.         elif move == 'L' or move == 'A':
303.             pDir = {NORTH: WEST, WEST: SOUTH,
304.                     SOUTH: EAST, EAST: NORTH}[pDir]
305.             break
306.         elif move == 'R' or move == 'D':
307.             pDir = {NORTH: EAST, EAST: SOUTH,
308.                     SOUTH: WEST, WEST: NORTH}[pDir]
309.             break
310.         elif move.startswith('T'):  # Cheat code: 'T x,y'
311.             px, py = move.split()[1].split(',')
312.             px = int(px)
313.             py = int(py)
314.             break
315.         else:
316.             print('You cannot move in that direction.')
317.
318.     if (px, py) == (exitx, exity):
319.         print('You have reached the exit! Good job!')
320.         print('Thanks for playing!')
321.         sys.exit()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. К возникновению какой программной ошибки приведет замена `move == 'QUIT'` в строке 279 на `move == 'quit'`?
2. Как убрать из этой программы чит с телепортацией?

46

Моделирование статистики за миллион бросков игральных костей

При броске двух шестигранных костей вероятность выпадения 7 составляет около 17 % – намного больше, чем вероятность выпадения 2: всего 3 %. Все дело в том, что только одна из комбинаций дает 2 (когда на обеих костях выпадает 1), а комбинаций, дающих в сумме 7, — довольно много: 1 и 6, 2 и 5, 3 и 4 и т. д.

А как насчет броска трех костей? Или четырех? Или тысячи? Можно математически вычислить теоретические значения вероятности, а можно заставить компьютер «бросить кости» миллион раз и получить эмпирические значения. Именно этот второй подход и реализуется в нашей программе. В ней мы просим компьютер «бросить» *N* костей миллион раз и запомнить результаты, а затем отобразить вероятности выбрасывания каждого количества очков в процентах.

Объемы производимых этой программой вычислений весьма значительны, но сам код несложен для понимания.

Программа в действии

Результат выполнения `milliondicestats.py` выглядит следующим образом:

```
Million Dice Roll Statistics Simulator
By Al Sweigart al@inventwithpython.com
```

```
Enter how many six-sided dice you want to roll:
> 2
Simulating 1,000,000 rolls of 2 dice...
36.2% done...
73.4% done...
TOTAL - ROLLS - PERCENTAGE
  2 - 27590 rolls - 2.8%
  3 - 55730 rolls - 5.6%
  4 - 83517 rolls - 8.4%
  5 - 111526 rolls - 11.2%
  6 - 139015 rolls - 13.9%
  7 - 166327 rolls - 16.6%
  8 - 139477 rolls - 13.9%
  9 - 110268 rolls - 11.0%
  10 - 83272 rolls - 8.3%
  11 - 55255 rolls - 5.5%
  12 - 28023 rolls - 2.8%
```

Описание работы

Моделирование броска отдельной шестигранной игральной кости производится путем вызова `random.randint(1, 6)` в строке 30, который возвращает случайное число от 1 до 6, прибавляемое к промежуточной сумме всех произведенных бросков костей. Функция `random.randint()` обеспечивает нормальное распределение, то есть вероятности всех возвращаемых ею чисел одинаковы.

Результаты броска костей сохраняют в ассоциативном массиве `results`, ключами которого служат все возможные суммы бросков костей, а значениями — количество раз, которое встретилась конкретная сумма. Для получения частоты встречаемости конкретной суммы в процентах мы делим количество ее выпадений на 1 000 000 (общее количество бросков костей в нашем моделировании) и умножаем на 100 (чтобы получить значение в диапазоне от 0 до 100, а не от 0 до 1). Несложный подсчет показывает: это то же самое, что разделить количество ее выпадений на 10 000, что мы и делаем в строке 37.

```
 1. """Моделирование статистики за миллион бросков костей
 2. (c) Эл Свейгарт al@inventwithpython.com
 3. Моделирование миллиона бросков игральных костей.
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, для начинающих, математическая, имитационное моделирование"""
 6.
 7. import random, time
 8.
 9. print('''Million Dice Roll Statistics Simulator
10. By Al Sweigart al@inventwithpython.com
11.
12. Enter how many six-sided dice you want to roll:''')
13. numberOfDice = int(input('> '))
```

```
14.
15. # Подготовка ассоциативного массива для хранения результатов бросков костей:
16. results = {}
17. for i in range(numberOfDice, (numberOfDice * 6) + 1):
18.     results[i] = 0
19.
20. # Моделирование бросков костей:
21. print('Simulating 1,000,000 rolls of {} dice...'.format(numberOfDice))
22. lastPrintTime = time.time()
23. for i in range(1000000):
24.     if time.time() > lastPrintTime + 1:
25.         print('{}% done...'.format(round(i / 10000, 1)))
26.         lastPrintTime = time.time()
27.
28.     total = 0
29.     for j in range(numberOfDice):
30.         total = total + random.randint(1, 6)
31.     results[total] = results[total] + 1
32.
33. # Выводим результаты:
34. print('TOTAL - ROLLS - PERCENTAGE')
35. for i in range(numberOfDice, (numberOfDice * 6) + 1):
36.     roll = results[i]
37.     percentage = round(results[i] / 10000, 1)
38.     print('  {} - {} rolls - {}%'.format(i, roll, percentage))
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- реализовать выбрасывание 8-, 10-, 12- или 20-гранных костей;
- смоделировать подбрасывание двусторонней монеты.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `lastPrintTime + 1` в строке 24 заменить на `lastPrintTime + 2`?
2. Какая программная ошибка возникнет, если удалить или закомментировать `results[total] = results[total] + 1` в строке 31?
3. Какое сообщение об ошибке будет выдано, если пользователь введет буквы вместо числа при запросе количества выбрасываемых шестигранных костей?

47

Генератор картин в стиле Мондриана

Пит Мондриан (Piet Mondrian) — нидерландский художник XX века, основатель неопластицизма, одного из направлений абстрактной живописи. В основе культовых рисунков художника лежат полосы основных цветов (синий, желтый, красный), а также черного и белого. Базируясь на минималистском подходе, Мондриан разделял эти цвета горизонтальными и вертикальными элементами.

Наша программа генерирует случайные картины в стиле Мондриана. Узнать больше о художнике вы можете в статье «Википедии»: https://ru.wikipedia.org/wiki/Мондриан,_Пит.

Программа в действии

Благодаря модулю `bext` программа на Python может отображать на экране яркие основные цвета в текстовом режиме, хотя в этой книге вы видите только черно-белые изображения. На рис. 47.1 показано, как выглядит результат выполнения `mondrian.py`.

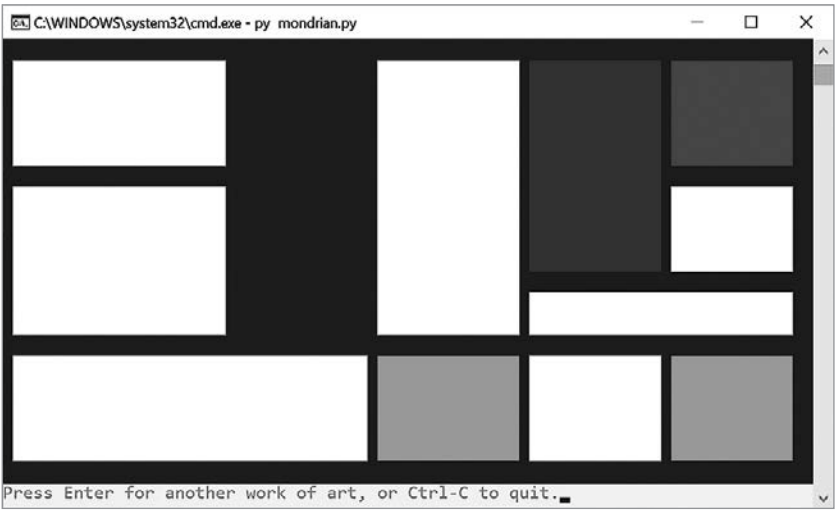


Рис. 47.1. Сгенерированная компьютером картина в стиле Мондриана. При каждом запуске генерируется новая картина

Описание работы

В основе алгоритма лежит создание структуры данных (ассоциативного массива `canvas`) с горизонтальными и вертикальными отрезками, расположенными на случайном расстоянии друг от друга, как показано на рис. 47.2.

Далее некоторые сегменты отрезков удаляются в целях формирования более крупных прямоугольников, как показано на рис. 47.3.

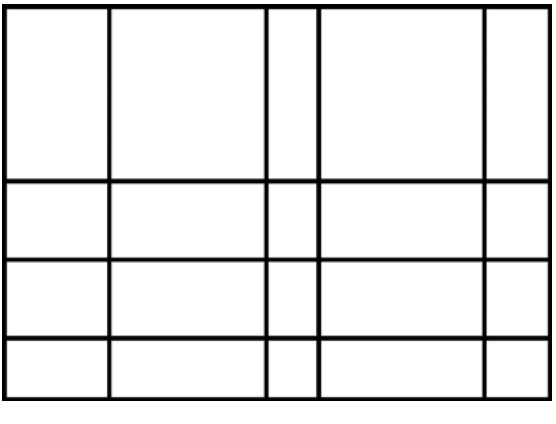


Рис. 47.2. Первый шаг алгоритма создания картин в стиле Мондриана: создание сетки

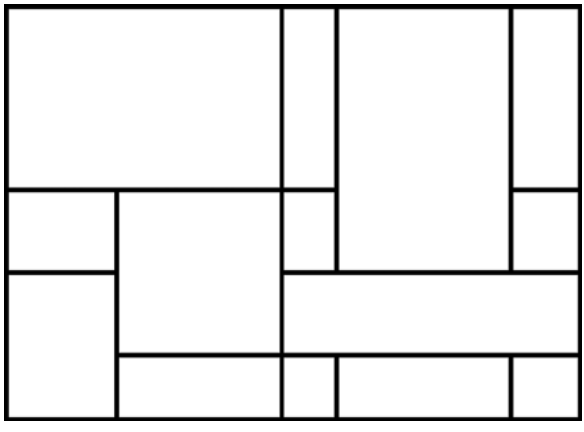


Рис. 47.3. Второй шаг алгоритма создания картин в стиле Мондриана: удаление части отрезков случайным образом

Наконец, алгоритм заливает случайным образом часть прямоугольников желтым, красным, синим или черным цветом, как показано на рис. 47.4.

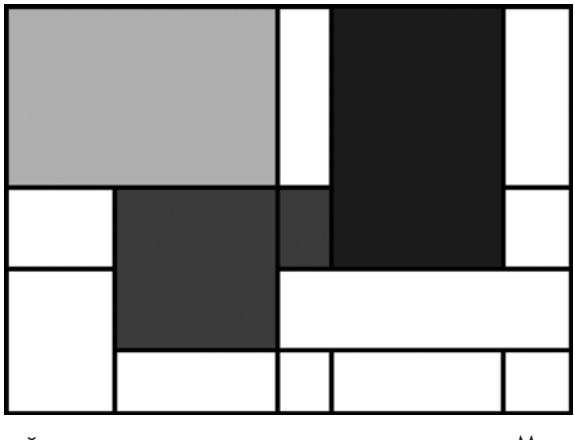


Рис. 47.4. Третий шаг алгоритма создания картин в стиле Мондриана: заливка выбранных случайным образом прямоугольников различными цветами

Найти еще одну версию этого генератора картин в стиле Мондриана, а также несколько примеров картин можно по адресу https://github.com/asweigart/mondrian_art_generator/.

```
1. """Генератор картин в стиле Мондриана, (c) Эл Свейгарт al@inventwithpython.com
2. Генерирует случайным образом картины в стиле Пита Мондриана.
3. Больше информации — в статье https://ru.wikipedia.org/wiki/Мондриан,_Пит
```

```
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: большая, графика, bext"""
 6.
 7. import sys, random
 8.
 9. try:
10.     import bext
11. except ImportError:
12.     print('This program requires the bext module, which you')
13.     print('can install by following the instructions at')
14.     print('https://pypi.org/project/Bext/')
15.     sys.exit()
16.
17. # Задаем константы:
18. MIN_X_INCREASE = 6
19. MAX_X_INCREASE = 16
20. MIN_Y_INCREASE = 3
21. MAX_Y_INCREASE = 6
22. WHITE = 'white'
23. BLACK = 'black'
24. RED = 'red'
25. YELLOW = 'yellow'
26. BLUE = 'blue'
27.
28. # Настройки экрана:
29. width, height = bext.size()
30. # В Windows нельзя вывести что-либо в последнем столбце без добавления
31. # автоматически символа новой строки, поэтому уменьшаем ширину на 1:
32. width -= 1
33.
34. height -= 3
35.
36. while True:  # Основной цикл приложения.
37.     # Заполняем сначала холст белым фоном:
38.     canvas = {}
39.     for x in range(width):
40.         for y in range(height):
41.             canvas[(x, y)] = WHITE
42.
43.     # Генерируем вертикальные отрезки:
44.     numberOfSegmentsToDelete = 0
45.     x = random.randint(MIN_X_INCREASE, MAX_X_INCREASE)
46.     while x < width - MIN_X_INCREASE:
47.         numberOfSegmentsToDelete += 1
48.         for y in range(height):
49.             canvas[(x, y)] = BLACK
50.         x += random.randint(MIN_X_INCREASE, MAX_X_INCREASE)
51.
52.     # Генерируем горизонтальные отрезки:
53.     y = random.randint(MIN_Y_INCREASE, MAX_Y_INCREASE)
54.     while y < height - MIN_Y_INCREASE:
```

```
        numberOfSegmentsToDelete += 1
        for x in range(width):
            canvas[(x, y)] = BLACK
        y += random.randint(MIN_Y_INCREASE, MAX_Y_INCREASE)

    numberOfRectanglesToPaint = numberOfSegmentsToDelete - 3
    numberOfSegmentsToDelete = int(numberOfSegmentsToDelete * 1.5)

    # Выбираем сегменты случайным образом и пытаемся их удалить.
    for i in range(numberOfSegmentsToDelete):
        while True:  # Продолжаем выбирать сегменты для удаления.
            # Выбираем случайную начальную точку сегмента:
            startx = random.randint(1, width - 2)
            starty = random.randint(1, height - 2)
            if canvas[(startx, starty)] == WHITE:
                continue

            # Определяем, сегмент вертикальный или горизонтальный:
            if (canvas[(startx - 1, starty)] == WHITE and
                canvas[(startx + 1, starty)] == WHITE):
                orientation = 'vertical'
            elif (canvas[(startx, starty - 1)] == WHITE and
                canvas[(startx, starty + 1)] == WHITE):
                orientation = 'horizontal'
            else:
                # Начальная точка находится на пересечении,
                # так что нужно выбрать новую:
                continue

            pointsToDelete = [(startx, starty)]

            canDeleteSegment = True
            if orientation == 'vertical':
                # Поднимаемся на один участок от начальной точки
                # и смотрим, можно ли удалить этот сегмент:
                for changey in (-1, 1):
                    y = starty
                    while 0 < y < height - 1:
                        y += changey
                        if (canvas[(startx - 1, y)] == BLACK and
                            canvas[(startx + 1, y)] == BLACK):
                            # Наткнулись на четырехстороннее пересечение.
                            break
                        elif ((canvas[(startx - 1, y)] == WHITE and
                               canvas[(startx + 1, y)] == BLACK) or
                              (canvas[(startx - 1, y)] == BLACK and
                               canvas[(startx + 1, y)] == WHITE)):
                            # Наткнулись на Т-образное пересечение;
                            # удалить этот сегмент нельзя:
                            canDeleteSegment = False
                            break
                        else:
```

```
                            pointsToDelete.append((startx, y))

            elif orientation == 'horizontal':
                # Поднимаемся на один участок от начальной точки
                # и смотрим, можно ли удалить этот сегмент:
                for changex in (-1, 1):
                    x = startx
                    while 0 < x < width - 1:
                        x += changex
                        if (canvas[(x, starty - 1)] == BLACK and
                            canvas[(x, starty + 1)] == BLACK):
                            # Наткнулись на четырехстороннее пересечение.
                            break
                        elif ((canvas[(x, starty - 1)] == WHITE and
                               canvas[(x, starty + 1)] == BLACK) or
                              (canvas[(x, starty - 1)] == BLACK and
                               canvas[(x, starty + 1)] == WHITE)):
                            # Наткнулись на Т-образное пересечение;
                            # удалить этот сегмент нельзя:
                            canDeleteSegment = False
                            break
                        else:
                            pointsToDelete.append((x, starty))
            if not canDeleteSegment:
                continue  # Выбираем новую случайную начальную точку.
            break  # Переходим к удалению данного сегмента.

        # Если сегмент можно удалить, закрашиваем все точки белым:
        for x, y in pointsToDelete:
            canvas[(x, y)] = WHITE

    # Добавляем граничные линии:
    for x in range(width):
        canvas[(x, 0)] = BLACK  # Верхняя граница.
        canvas[(x, height - 1)] = BLACK  # Нижняя граница.
    for y in range(height):
        canvas[(0, y)] = BLACK  # Левая граница.
        canvas[(width - 1, y)] = BLACK  # Правая граница.

    # Рисуем прямоугольники:
    for i in range(numberOfRectanglesToPaint):
        while True:
            startx = random.randint(1, width - 2)
            starty = random.randint(1, height - 2)

            if canvas[(startx, starty)] != WHITE:
                continue  # Выбираем новую случайную начальную точку.
            else:
                break

        # Алгоритм заливки:
        colorToPaint = random.choice([RED, YELLOW, BLUE, BLACK])
```

```
159.          pointsToPaint = set([(startx, starty)])
160.          while len(pointsToPaint) > 0:
161.              x, y = pointsToPaint.pop()
162.              canvas[(x, y)] = colorToPaint
163.              if canvas[(x - 1, y)] == WHITE:
164.                  pointsToPaint.add((x - 1, y))
165.              if canvas[(x + 1, y)] == WHITE:
166.                  pointsToPaint.add((x + 1, y))
167.              if canvas[(x, y - 1)] == WHITE:
168.                  pointsToPaint.add((x, y - 1))
169.              if canvas[(x, y + 1)] == WHITE:
170.                  pointsToPaint.add((x, y + 1))
171.
172.      # Рисуем на экране структуру данных canvas:
173.      for y in range(height):
174.          for x in range(width):
175.              bext.bg(canvas[(x, y)])
176.              print(' ', end='')
177.
178.          print()
179.
180.      # Спрашиваем пользователя, нарисовать ли еще одну картину:
181.      try:
182.          input('Press Enter for another work of art, or Ctrl-C to quit.')
183.      except KeyboardInterrupt:
184.          sys.exit()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создать программы с различными цветовыми палитрами;
- генерировать файлы изображений для картин в стиле Мондриана с помощью модуля `Pillow`. Узнать больше об этом модуле можно из главы 19 книги *Automate the Boring Stuff with Python* по адресу https://automatetheboringstuff.com/2e/chapter19/.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какая ошибка возникнет, если `canvas[(x, y)] = WHITE` в строке 41 заменить на `canvas[(x, y)] = RED`?
2. Что будет, если `print(' ', end='')` в строке 176 заменить на `print('A', end='')`?

48

Парадокс Монти Холла

Парадокс (задача) Монти Холла иллюстрирует неожиданный факт из теории вероятностей. В основе его лежит старая телевизионная игра Let's Make a Deal и ее ведущий Монти Холл. В задаче вы можете выбрать одну из трех дверей. За одной приз — новый автомобиль. За двумя другими — две не имеющие никакой ценности козы. Пусть вы выбрали дверь 1. Прежде чем открыть ее, ведущий открывает другую дверь (2 или 3), за которой находится коза. Вы можете либо все-таки открыть дверь, которую выбрали изначально, либо изменить свой выбор и открыть другую закрытую дверь.

Может показаться, что неважно, менять свой выбор или нет, но окажется, что ваши шансы повысятся, если выбрать другую дверь! Наша программа демонстрирует парадокс Монти Холла в действии, позволяя вам провести несколько экспериментов.

Чтобы понять, почему шансы повысятся, рассмотрим версию задачи Монти Холла с тысячью дверей вместо трех. Игрок выбирает одну дверь, а затем ведущий открывает 998 дверей, за которыми блеют козы. Остается только две неоткрытых двери: выбранная игроком изначально и еще одна. Если игрок изначально выбрал дверь с автомобилем (1 шанс из 1000), то ведущий оставит закрытой какую-то случайную дверь с козой. Если игрок выберет дверь с козой (999 шансов из 1000), то ведущий специально оставит закрытой дверь с автомобилем. Выбор открываемых ведущим дверей не случаен. Ведущий знает, что дверь с автомобилем должна остаться закрытой. Игрок практически наверняка не выберет изначально дверь с автомобилем, поэтому имеет смысл изменить свой выбор, указав на другую дверь.

Или представьте, что у вас есть 1000 коробок, одна из которых — с призом. Если вы угадаете, в какой коробке приз, то ведущий торжественно вам его вручит. Как вы думаете, приз в выбранной вами коробке или в одной из 999 прочих коробок? Вам не нужно, чтобы ведущий открывал 998 из 999 пустых коробок; выбор такой же, как и в случае 1000 дверей. Вероятность угадать правильно с самого начала равна 1 из 1000, а не угадать (то есть что приз — в одной из других коробок) — практически 100%-ная, 999 из 1000.

Больше информации о парадоксе Монти Холла можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Парадокс_Монти_Холла.

Программа в действии

Результат выполнения `montyhall.py` выглядит следующим образом:

```
The Monty Hall Problem, by Al Sweigart al@inventwithpython.com
--сокращено--
+------+  +------+  +------+
|      |  |      |  |      |
|   1  |  |   2  |  |   3  |
|      |  |      |  |      |
|      |  |      |  |      |
|      |  |      |  |      |
+------+  +------+  +------+
Pick a door 1, 2, or 3 (or "quit" to stop):
> 1

+------+  +------+  +------+
|      |  |      |  |  ((  |
|   1  |  |   2  |  |  oo  |
|      |  |      |  | /_/|_|
|      |  |      |  |    | |
|      |  |      |  |GOAT|||
+------+  +------+  +------+
Door 3 contains a goat!
Do you want to swap doors? Y/N
> y

+------+  +------+  +------+
|  ((  |  | CAR! |  |  ((  |
|  oo  |  |    __|  |  oo  |
| /_/|_|  |  _/   |  | /_/|_|
|    | |  | /_ __|  |    | |
|GOAT|||  |   O  |  |GOAT|||
+------+  +------+  +------+
Door 2 has the car!
You won!

Swapping:     1 wins, 0 losses, success rate 100.0%
```

```
Not swapping: 0 wins, 0 losses, success rate 0.0%

Press Enter to repeat the experiment...
--сокращено--
```

Описание работы

Многострочные строковые значения для нарисованных ASCII-графикой дверей хранятся в нескольких константах: `ALL_CLOSED`, `FIRST_GOAT` и `FIRST_CAR_OTHERS_GOAT`. Использующий эти константы код, например `print(FIRST_GOAT)` в строке 125, не изменится, даже если мы внесем какие-либо изменения в графику. Благодаря тому что эти многострочные строковые значения размещаются вместе, вверху файла кода, будет проще сравнить их, чтобы убедиться, что вся графика согласуется между собой.

```
 1. """Проблема Монти Холла, (c) Эл Свейгарт al@inventwithpython.com
 2. Моделирование задачи телевизионной игры Монти Холла
 3. Подробнее — в статье https://ru.wikipedia.org/wiki/Парадокс_Монти_Холла
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: большая, игра, математическая, имитационное моделирование"""
 6.
 7. import random, sys
 8.
 9. ALL_CLOSED = """
10. +------+  +------+  +------+
11. |      |  |      |  |      |
12. |   1  |  |   2  |  |   3  |
13. |      |  |      |  |      |
14. |      |  |      |  |      |
15. |      |  |      |  |      |
16. +------+  +------+  +------+"""
17.
18. FIRST_GOAT = """
19. +------+  +------+  +------+
20. |  ((  |  |      |  |      |
21. |  oo  |  |   2  |  |   3  |
22. | /_/|_|  |      |  |      |
23. |    | |  |      |  |      |
24. |GOAT|||  |      |  |      |
25. +------+  +------+  +------+"""
26.
27. SECOND_GOAT = """
28. +------+  +------+  +------+
29. |      |  |  ((  |  |      |
30. |   1  |  |  oo  |  |   3  |
31. |      |  | /_/|_|  |      |
32. |      |  |    | |  |      |
33. |      |  |GOAT|||  |      |
34. +------+  +------+  +------+"""
```

```
35.
36. THIRD_GOAT = """
37. +------+  +------+  +------+
38. |      |  |      |  |  ((  |
39. |   1  |  |   2  |  |  oo  |
40. |      |  |      |  | /_/|_|
41. |      |  |      |  |    | |
42. |      |  |      |  |GOAT|||
43. +------+  +------+  +------+"""
44.
45. FIRST_CAR_OTHERS_GOAT = """
46. +------+  +------+  +------+
47. | CAR! |  |  ((  |  |  ((  |
48. |    __|  |  oo  |  |  oo  |
49. |  _/  |  | /_/|_|  | /_/|_|
50. | /_ __|  |    | |  |    | |
51. |   O  |  |GOAT|||  |GOAT|||
52. +------+  +------+  +------+"""
53.
54. SECOND_CAR_OTHERS_GOAT = """
55. +------+  +------+  +------+
56. |  ((  |  | CAR! |  |  ((  |
57. |  oo  |  |    __|  |  oo  |
58. | /_/|_|  |  _/  |  | /_/|_|
59. |    | |  | /_ __|  |    | |
60. |GOAT|||  |   O  |  |GOAT|||
61. +------+  +------+  +------+"""
62.
63. THIRD_CAR_OTHERS_GOAT = """
64. +------+  +------+  +------+
65. |  ((  |  |  ((  |  | CAR! |
66. |  oo  |  |  oo  |  |    __|
67. | /_/|_|  | /_/|_|  |  _/  |
68. |    | |  |    | |  | /_ __|
69. |GOAT|||  |GOAT|||  |   O  |
70. +------+  +------+  +------+"""
71.
72. print('''The Monty Hall Problem, by Al Sweigart al@inventwithpython.com
73.
74. In the Monty Hall game show, you can pick one of three doors. One door
75. has a new car for a prize. The other two doors have worthless goats:
76. {}
77. Say you pick Door #1.
78. Before the door you choose is opened, another door with a goat is opened:
79. {}
80. You can choose to either open the door you originally picked or swap
81. to the other unopened door.
82.
83. It may seem like it doesn't matter if you swap or not, but your odds
84. do improve if you swap doors! This program demonstrates the Monty Hall
85. problem by letting you do repeated experiments.
```

```
 86.
 87. You can read an explanation of why swapping is better at
 88. https://en.wikipedia.org/wiki/Monty_Hall_problem
 89. '''.format(ALL_CLOSED, THIRD_GOAT))
 90.
 91. input('Press Enter to start...')
 92.
 93.
 94. swapWins = 0
 95. swapLosses = 0
 96. stayWins = 0
 97. stayLosses = 0
 98. while True:  # Основной цикл программы.
 99.     # Компьютер выбирает, за какой дверью будет автомобиль:
100.     doorThatHasCar = random.randint(1, 3)
101.     # Просим игрока выбрать дверь:
102.     print(ALL_CLOSED)
103.     while True:  # Продолжаем спрашивать, пока игрок не выберет допустимую
104.                  # дверь.
105.         print('Pick a door 1, 2, or 3 (or "quit" to stop):')
106.         response = input('> ').upper()
107.         if response == 'QUIT':
108.             # Завершаем игру.
109.             print('Thanks for playing!')
110.             sys.exit()
111.
112.         if response == '1' or response == '2' or response == '3':
113.             break
114.     doorPick = int(response)
115.
116.     # Выбираем, какую дверь с козой открыть:
117.     while True:
118.         # Выбираем одну из дверей с козой, не выбранную игроком:
119.         showGoatDoor = random.randint(1, 3)
120.         if showGoatDoor != doorPick and showGoatDoor != doorThatHasCar:
121.             break
122.
123.     # Открываем эту дверь с козой:
124.     if showGoatDoor == 1:
125.         print(FIRST_GOAT)
126.     elif showGoatDoor == 2:
127.         print(SECOND_GOAT)
128.     elif showGoatDoor == 3:
129.         print(THIRD_GOAT)
130.
131.     print('Door {} contains a goat!'.format(showGoatDoor))
132.
133.     # Спрашиваем игрока, хочет ли он изменить свое решение:
134.     while True:  # Продолжаем спрашивать, пока игрок не введет Y или N.
135.         print('Do you want to swap doors? Y/N')
136.         swap = input('> ').upper()
```

```
        if swap == 'Y' or swap == 'N':
            break

    # Меняем дверь, если игрок решил изменить свой выбор:
    if swap == 'Y':
        if doorPick == 1 and showGoatDoor == 2:
            doorPick = 3
        elif doorPick == 1 and showGoatDoor == 3:
            doorPick = 2
        elif doorPick == 2 and showGoatDoor == 1:
            doorPick = 3
        elif doorPick == 2 and showGoatDoor == 3:
            doorPick = 1
        elif doorPick == 3 and showGoatDoor == 1:
            doorPick = 2
        elif doorPick == 3 and showGoatDoor == 2:
            doorPick = 1

    # Открываем все двери:
    if doorThatHasCar == 1:
        print(FIRST_CAR_OTHERS_GOAT)
    elif doorThatHasCar == 2:
        print(SECOND_CAR_OTHERS_GOAT)
    elif doorThatHasCar == 3:
        print(THIRD_CAR_OTHERS_GOAT)

    print('Door {} has the car!'.format(doorThatHasCar))

    # Фиксируем количество побед и неудач при изменении и неизменении решения:
    if doorPick == doorThatHasCar:
        print('You won!')
        if swap == 'Y':
            swapWins += 1
        elif swap == 'N':
            stayWins += 1
    else:
        print('Sorry, you lost.')
        if swap == 'Y':
            swapLosses += 1
        elif swap == 'N':
            stayLosses += 1

    # Вычисляем частоту выигрыша при изменении и неизменении решения:
    totalSwaps = swapWins + swapLosses
    if totalSwaps != 0:  # Защищаемся от деления на ноль.
        swapSuccess = round(swapWins / totalSwaps * 100, 1)
    else:
        swapSuccess = 0.0

    totalStays = stayWins + stayLosses
    if (stayWins + stayLosses) != 0:  # Защищаемся от деления на ноль.
```

```
188.         staySuccess = round(stayWins / totalStays * 100, 1)
189.     else:
190.         staySuccess = 0.0
191.
192.     print()
193.     print('Swapping:     ', end='')
194.     print('{} wins, {} losses, '.format(swapWins, swapLosses), end='')
195.     print('success rate {}%'.format(swapSuccess))
196.     print('Not swapping: ', end='')
197.     print('{} wins, {} losses, '.format(stayWins, stayLosses), end='')
198.     print('success rate {}%'.format(staySuccess))
199.     print()
200.     input('Press Enter to repeat the experiment...')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `doorThatHasCar = random.randint(1, 3)` в строке 100 заменить на `doorThatHasCar = 1`?
2. Что будет, если заменить строки с 124-й по 129-ю на `print([FIRST_GOAT, SECOND_GOAT, THIRD_GOAT][showGoatDoor - 1])`?

49

Таблица умножения

Эта программа генерирует таблицу умножения от 0 × 0 до 12 × 12. Будучи очень простой, она тем не менее служит прекрасной иллюстрацией вложенных циклов.

Программа в действии

Результат выполнения `multiplicationtable.py` выглядит следующим образом:

```
Multiplication Table, by Al Sweigart al@inventwithpython.com
   |  0   1   2   3   4   5   6   7   8   9  10  11  12
---+-------------------------------------------------
 0|  0   0   0   0   0   0   0   0   0   0   0   0   0
 1|  0   1   2   3   4   5   6   7   8   9  10  11  12
 2|  0   2   4   6   8  10  12  14  16  18  20  22  24
 3|  0   3   6   9  12  15  18  21  24  27  30  33  36
 4|  0   4   8  12  16  20  24  28  32  36  40  44  48
 5|  0   5  10  15  20  25  30  35  40  45  50  55  60
 6|  0   6  12  18  24  30  36  42  48  54  60  66  72
 7|  0   7  14  21  28  35  42  49  56  63  70  77  84
 8|  0   8  16  24  32  40  48  56  64  72  80  88  96
 9|  0   9  18  27  36  45  54  63  72  81  90  99 108
10|  0  10  20  30  40  50  60  70  80  90 100 110 120
11|  0  11  22  33  44  55  66  77  88  99 110 121 132
12|  0  12  24  36  48  60  72  84  96 108 120 132 144
```

Описание работы

Строка 9 выводит на экран верхнюю строку таблицы. Обратите внимание: мы оставили достаточно большое расстояние между числами, чтобы хватило места для произведений в три цифры длиной. Поскольку наша таблица доходит до 12 × 12, такие поля необходимы, чтобы поместилось максимальное произведение, 144. При создании более обширных таблиц может понадобиться увеличить и ширину столбцов. Учтите, что ширина стандартного окна терминала составляет 80 столбцов, а высота — 24 строки, поэтому создать гораздо бóльшие таблицы умножения, не перенося строки на правом краю окна, не получится.

```
 1. """Таблица умножения, (c) Эл Свейгарт al@inventwithpython.com
 2. Выводит на экран таблицу умножения.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, математическая"""
 5.
 6. print('Multiplication Table, by Al Sweigart al@inventwithpython.com')
 7.
 8. # Выводим горизонтальные метки чисел:
 9. print('  |  0   1   2   3   4   5   6   7   8   9  10  11  12')
10. print('--+---------------------------------------------------')
11.
12. # Построчно выводим на экран произведения:
13. for number1 in range(0, 13):
14.
15.     # Выводим вертикальные метки чисел:
16.     print(str(number1).rjust(2), end='')
17.
18.     # Выводим разделитель:
19.     print('|', end='')
20.
21.     for number2 in range(0, 13):
22.         # Выводит произведение и пробел:
23.         print(str(number1 * number2).rjust(3), end=' ')
24.
25.     print()  # Завершаем строку символом новой строки.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `range(0, 13)` в строке 13 заменить на `range(0, 80)`?
2. Что будет, если `range(0, 13)` в строке 13 заменить на `range(0, 100)`?

50

Девяносто девять бутылок

«Девяносто девять бутылок» — народная песня неизвестного происхождения, знаменитая своей длиной и повторениями. В ней поется: «Девяносто девять бутылок молока на стене, девяносто девять бутылок молока. Возьми одну, пусти по кругу, девяносто девять бутылок молока на стене». По мере повторения текста количество бутылок уменьшается с девяносто восьми до девяносто семи, затем с девяносто семи до девяносто шести и т. д., пока не достигнет нуля: «Одна бутылка молока на стене, одна бутылка молока. Возьми ее, пусти по кругу, и больше нет бутылок молока на стене!»

К счастью для нас, компьютеры прекрасно умеют выполнять повторяющиеся задания, и эта программа воспроизводит весь текст песни программным образом. Расширенную версию данной программы вы найдете в проекте 51.

Программа в действии

Результат выполнения `ninetyninebottles.py` выглядит следующим образом:

```
Ninety-Nine Bottles, by Al Sweigart al@inventwithpython.com

(Press Ctrl-C to quit.)
99 bottles of milk on the wall,
99 bottles of milk,
Take one down, pass it around,
98 bottles of milk on the wall!
```

```
98 bottles of milk on the wall,
98 bottles of milk,
Take one down, pass it around,
97 bottles of milk on the wall!
--сокращено--
```

Описание работы

Повторения текста в этой песне позволяют легко вывести первые 98 куплетов с помощью цикла `for` (строки с 20-й по 30-ю). Однако последний куплет содержит небольшие отличия и его вывод требует отдельного кода (строки с 33-й по 39-ю), поскольку последняя строка `'No more bottles of milk on the wall!'`, отличается от повторяющейся в цикле строки, а также потому, что слово bottle стоит в единственном числе, а не во множественном.

```
 1. """Девяносто девять бутылок молока на стене
 2. (c) Эл Свейгарт al@inventwithpython.com
 3. Выводит полный текст одной из самых длинных песен на свете!
 4. Нажмите Ctrl-C для останова.
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: крошечная, для начинающих, прокрутка"""
 7.
 8. import sys, time
 9.
10. print('Ninety-Nine Bottles, by Al Sweigart al@inventwithpython.com')
11. print()
12. print('(Press Ctrl-C to quit.)')
13.
14. time.sleep(2)
15.
16. bottles = 99  # Начальное количество бутылок.
17. PAUSE = 2  # (!) Замените на 0, чтобы сразу увидеть всю песню.
18.
19. try:
20.     while bottles > 1:  # Отображаем текст песни в цикле.
21.         print(bottles, 'bottles of milk on the wall,')
22.         time.sleep(PAUSE)  # Приостанавливаем на PAUSE секунд.
23.         print(bottles, 'bottles of milk,')
24.         time.sleep(PAUSE)
25.         print('Take one down, pass it around,')
26.         time.sleep(PAUSE)
27.         bottles = bottles - 1  # Уменьшаем количество бутылок на 1.
28.         print(bottles, 'bottles of milk on the wall!')
29.         time.sleep(PAUSE)
30.         print()  # Выводим символ новой строки.
31.
32.     # Выводим последний куплет:
33.     print('1 bottle of milk on the wall,')
34.     time.sleep(PAUSE)
```

```
35.     print('1 bottle of milk,')
36.     time.sleep(PAUSE)
37.     print('Take it down, pass it around,')
38.     time.sleep(PAUSE)
39.     print('No more bottles of milk on the wall!')
40. except KeyboardInterrupt:
41.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создать программу для вывода повторяющейся песни The Twelve Days of Christmas;
- создать программы для прочих кумулятивных песен. Список их можно найти в https://en.wikipedia.org/wiki/Cumulative_song.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `bottles = bottles - 1` в строке 27 заменить на `bottles = bottles - 2`?
2. Что будет, если `while bottles > 1:` в строке 20 заменить на `while bottles < 1:`?

51

ДевяНосто деевяять буутылок

В этой версии песни «Девяносто девять бутылок» программа вносит в каждый куплет небольшие умышленные дефекты, убирая букву, меняя ее регистр, меняя местами две буквы или удваивая букву.

По мере хода песни эти дефекты наращиваются, в результате чего получается совершенно бессмысленный текст. Рекомендую взглянуть на проект 50, прежде чем приступать к этому.

Программа в действии

Результат выполнения `ninetyninebottles2.py` выглядит следующим образом:

```
niNety-nniinE BoOttels, by Al Sweigart al@inventwithpython.com
--сокращено--
99 bottles of milk on the wall,
99 bottles of milk,
Take one down, pass it around,
98 bottles of milk on the wall!
98 bottles of milk on the wall,
98 bottles of milk,
Take one d wn, pass it around,
97 bottles of milk on the wall!
97 bottles of milk on the wall,
97 bottels of milk,
```

```
Take one d wn,  pass it around,
96 bottles of milk on the wall!
--сокращено--
75b otlte  of mIl  on teh wall,
75   ottels  f miLk,
Take one d wn,  pass it ar und,
74 bbOttles of milk on t e wall!
--сокращено--
1  otlE t of iml  oo nteh  lall,
1   o  Tle   FF FmMLIIkk,
Taake on  d wn,  pAasSs itt au nn d,
No more bottles of milk on the wall!
```

Описание работы

Строковые значения в Python — *неизменяемые* (immutable), то есть изменить их невозможно. Если строковое значение `'Hello'` хранится в переменной `greeting`, то код `greeting = greeting + ' world!'` на самом деле не меняет строкового значения `'Hello'`, а создает новое строковое значение `'Hello world!'`, заменяющее в `greeting` строковое значение `'Hello'`. Технические основания этого выходят за рамки данной книги, но важно отдавать себе отчет в этих различиях, поскольку код `greeting[0] = 'h'` недопустим, раз строковые значения неизменяемы. Однако, поскольку списки изменяемы, мы можем создать список строковых значений из одного символа (как мы и делаем в строке 62), изменить символы в списке, а затем на основе данного списка создать строковое значение (строка 85).

```
 1. """ДевяНосто деевяять буутылок млка На те не
 2. (c) Эл Свейгарт al@inventwithpython.com
 3. Выводит полный текст одной из самых длинных песен на свете! С каждым
 4. куплетом текст становится все более бессмысленным. Нажмите Ctrl+C для останова.
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, прокрутка, слова"""
 7.
 8. import random, sys, time
 9.
10. # Задаем константы:
11. # (!) Замените обе эти константы на 0, чтобы сразу увидеть всю песню.
12. SPEED = 0.01  # Пауза между выводом отдельных букв.
13. LINE_PAUSE = 1.5  # Пауза в конце каждой строки.
14.
15.
16. def slowPrint(text, pauseAmount=0.1):
17.     """Медленно выводим символы текста по одному."""
18.     for character in text:
19.         # Укажите здесь flush=True, чтобы текст был выведен весь сразу:
20.         print(character, flush=True, end='')
21.         # end='' означает отсутствие перевода на новую строку.
22.         time.sleep(pauseAmount)  # Паузы между выводом символов.
```

```
23.     print()  # Выводим символ новой строки.
24.
25. print('niNety-nniinE BoOttels, by Al Sweigart al@inventwithpython.com')
26. print()
27. print('(Press Ctrl-C to quit.)')
28.
29. time.sleep(2)
30.
31. bottles = 99  # Начальное количество бутылок.
32.
33. # Список со строковыми значениями текста песни:
34. lines = [' bottles of milk on the wall,',
35.          ' bottles of milk,',
36.          'Take one down, pass it around,',
37.          ' bottles of milk on the wall!']
38.
39. try:
40.     while bottles > 0:  # Отображаем текст песни в цикле.
41.         slowPrint(str(bottles) + lines[0], SPEED)
42.         time.sleep(LINE_PAUSE)
43.         slowPrint(str(bottles) + lines[1], SPEED)
44.         time.sleep(LINE_PAUSE)
45.         slowPrint(lines[2], SPEED)
46.         time.sleep(LINE_PAUSE)
47.         bottles = bottles - 1  # Уменьшаем количество бутылок на 1.
48.
49.         if bottles > 0:  # Выводим последнюю строку текущего куплета.
50.             slowPrint(str(bottles) + lines[3], SPEED)
51.         else:  # Выводим последнюю строку всей песни.
52.             slowPrint('No more bottles of milk on the wall!', SPEED)
53.
54.         time.sleep(LINE_PAUSE)
55.         print()  # Выводим символ новой строки.
56.
57.         # Выбираем случайную строку, которую будем делать "смешной":
58.         lineNum = random.randint(0, 3)
59.
60.         # Делаем список из символов строки текста, чтобы можно было
61.         # редактировать (строковые значения в Python — неизменяемые.)
62.         line = list(lines[lineNum])
63.
64.         effect = random.randint(0, 3)
65.         if effect == 0:  # Заменяем символ пробелом.
66.             charIndex = random.randint(0, len(line) - 1)
67.             line[charIndex] = ' '
68.         elif effect == 1:  # Меняем регистр символа.
69.             charIndex = random.randint(0, len(line) - 1)
70.             if line[charIndex].isupper():
71.                 line[charIndex] = line[charIndex].lower()
72.             elif line[charIndex].islower():
73.                 line[charIndex] = line[charIndex].upper()
```

```
74.         elif effect == 2:  # Меняем два символа местами.
75.             charIndex = random.randint(0, len(line) - 2)
76.             firstChar = line[charIndex]
77.             secondChar = line[charIndex + 1]
78.             line[charIndex] = secondChar
79.             line[charIndex + 1] = firstChar
80.         elif effect == 3:  # Удваиваем символ.
81.             charIndex = random.randint(0, len(line) - 2)
82.             line.insert(charIndex, line[charIndex])
83.
84.         # Преобразуем список обратно в строковое значение и вставляем в lines:
85.         lines[lineNum] = ''.join(line)
86. except KeyboardInterrupt:
87.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи относительно возможных небольших изменений вы найдете в комментариях, помеченных (`!`). Можете также сами попробовать придумать, как сделать следующее:

- менять местами два смежных слова, где под словом понимается фрагмент текста, отделенный пробелами;
- начинать (изредка) отсчет в порядке возрастания на несколько итераций;
- менять регистр целого слова.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `bottles = bottles - 1` в строке 47 заменить на `bottles = bottles - 2`?
2. Что будет, если `effect = random.randint(0, 3)` в строке 64 заменить на `effect = 0`?
3. Какую ошибку вызовет удаление или комментирование `line = list(lines [lineNum])` в строке 62?

52

Счет в различных системах счисления

Мы привыкли считать в десятичной системе счисления, в которой используется десять цифр: от 0 до 9. Вероятно, она возникла потому, что люди считали на пальцах, а у большинства людей десять пальцев. Но существуют и другие системы счисления. Компьютеры используют *двоичную* (binary) систему счисления всего с двумя цифрами, 0 и 1. Программисты также иногда применяют шестнадцатеричную систему счисления, в которой, помимо цифр от 0 до 9, используются еще и буквы от A до F.

Любое число можно представить в любой системе счисления, и наша программа выводит определенный диапазон чисел в десятичной, двоичной и шестнадцатеричной системах счисления.

Программа в действии

Результат выполнения `numeralsystems.py` выглядит следующим образом:

```
Numeral System Counters, by Al Sweigart al@inventwithpython.com

--сокращено--
Enter the starting number (e.g. 0) > 0
```

```
Enter how many numbers to display (e.g. 1000) > 20
DEC: 0    HEX: 0    BIN: 0
DEC: 1    HEX: 1    BIN: 1
DEC: 2    HEX: 2    BIN: 10
DEC: 3    HEX: 3    BIN: 11
DEC: 4    HEX: 4    BIN: 100
DEC: 5    HEX: 5    BIN: 101
DEC: 6    HEX: 6    BIN: 110
DEC: 7    HEX: 7    BIN: 111
DEC: 8    HEX: 8    BIN: 1000
DEC: 9    HEX: 9    BIN: 1001
DEC: 10    HEX: A    BIN: 1010
DEC: 11    HEX: B    BIN: 1011
DEC: 12    HEX: C    BIN: 1100
DEC: 13    HEX: D    BIN: 1101
DEC: 14    HEX: E    BIN: 1110
DEC: 15    HEX: F    BIN: 1111
DEC: 16    HEX: 10    BIN: 10000
DEC: 17    HEX: 11    BIN: 10001
DEC: 18    HEX: 12    BIN: 10010
DEC: 19    HEX: 13    BIN: 10011
```

Описание работы

Получить двоичное и шестнадцатеричное представление числа в Python можно путем вызова функций `bin()` и `hex()` соответственно:

```
>>> bin(42)
'0b101010'
>>> hex(42)
'0x2a'
```

Преобразовать эти строковые значения обратно в десятичные целые числа можно, вызвав функцию `int()` и указав основание системы счисления для преобразования, вот так:

```
>>> int('0b101010', 2)
42
>>> int('0x2a', 16)
42
```

Обратите внимание: на самом деле `bin()` и `hex()` возвращают не числа, а строковые значения: `bin(42)` возвращает строковое значение `'0b101010'`, а `hex(42)` — строковое значение `'0x2a'`. В соответствии с соглашением, в программировании добавляют префикс `0b` к двоичным числам и префикс `0x` — к шестнадцатеричным. Благодаря этому никто не перепутает двоичное число 10 000 (16 в десятичной системе счисления) с десятичным числом «десять тысяч». Наша программа для

работы с системами счисления удаляет эти префиксы перед выводом числа на экран.

```
 1. """Счет в различных системах счисления, (c) Эл Свейгарт al@inventwithpython.com
 2. Отображает эквивалентные числа в десятичной, двоичной и шестнадцатеричной
 3. системах счисления.
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, математическая"""
 6.
 7. print('''Numeral System Counters, by Al Sweigart al@inventwithpython.com
 8.
 9. This program shows you equivalent numbers in decimal (base 10),
10. hexadecimal (base 16), and binary (base 2) numeral systems.
11.
12. (Ctrl-C to quit.)
13. ''')
14.
15. while True:
16.     response = input('Enter the starting number (e.g. 0) > ')
17.     if response == '':
18.         response = '0'  # По умолчанию начинаем с 0.
19.         break
20.     if response.isdecimal():
21.         break
22.     print('Please enter a number greater than or equal to 0.')
23. start = int(response)
24.
25. while True:
26.     response = input('Enter how many numbers to display (e.g. 1000) > ')
27.     if response == '':
28.         response = '1000'  # По умолчанию выводим 1000 чисел.
29.         break
30.     if response.isdecimal():
31.         break
32.     print('Please enter a number.')
33. amount = int(response)
34.
35. for number in range(start, start + amount):  # Основной цикл программы.
36.     # Преобразуем в шестнадцатеричное/двоичное и удаляем префикс:
37.     hexNumber = hex(number)[2:].upper()
38.     binNumber = bin(number)[2:]
39.
40.     print('DEC:', number, '   HEX:', hexNumber, '   BIN:', binNumber)
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- добавить вывод значений в *восьмеричной* системе счисления — системе счисления по основанию 8 — с помощью функции `oct()` языка Python;

- поискать в интернете «преобразование между различными системами счисления» и реализовать собственные функции `bin()`, `oct()` и `hex()`.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `hex(number)[2:].upper()` в строке 37 заменить на `hex(number)[2:]`?
2. Что будет, если `int(response)` в строке 33 заменить на `response`?

53

Периодическая таблица элементов

В периодической системе все известные химические элементы организованы в единую таблицу. Наша программа выводит эту таблицу и дает возможность пользователю получить дополнительную информацию о каждом элементе, в частности атомное число, символ, температуру плавления и т. д. Я собрал эту информацию из «Википедии» и сохранил в файле `periodictable.csv`, который можно скачать по адресу https://inventwithpython.com/periodictable.csv.

Программа в действии

Результат выполнения `periodictable.py` выглядит следующим образом:

```
Periodic Table of Elements
By Al Sweigart al@inventwithpython.com

           Periodic Table of Elements
     1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
   1 H                                                  He
   2 Li Be                                 B  C  N  O  F  Ne
   3 Na Mg                                 Al Si P  S  Cl Ar
   4 K  Ca Sc Ti V  Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr
   5 Rb Sr Y  Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I  Xe
```

```
    6 Cs Ba La Hf Ta W  Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
    7 Fr Ra Ac Rf Db Sg Bh Hs Mt Ds Rg Cn Nh Fl Mc Lv Ts Og

            Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Lu
            Th Pa U  Np Pu Am Cm Bk Cf Es Fm Md No Lr
Enter a symbol or atomic number to examine, or QUIT to quit.
> 42
             Atomic Number: 42
                    Symbol: Mo
                   Element: Molybdenum
            Origin of name: Greek molýbdaina, 'piece of lead', from mólybdos, 'lead'
                     Group: 6
                    Period: 5
             Atomic weight: 95.95(1) u
                   Density: 10.22 g/cm^3
             Melting point: 2896 K
             Boiling point: 4912 K
    Specific heat capacity: 0.251 J/(g*K)
         Electronegativity: 2.16
Abundance in earth's crust: 1.2 mg/kg
Press Enter to continue...
--сокращено--
```

Описание работы

Файл в формате `.csv` (значения, отделенные запятыми, comma-separated values) — это простейшая электронная таблица в виде текстового файла. Строки файла соответствуют строкам таблицы, а столбцы разделяются запятыми. Например, первые три строки файла `periodictable.csv` выглядят следующим образом:

```
1,H,Hydrogen,"Greek elements hydro- and -gen, meaning 'water-forming--сокращено--
2,He,Helium,"Greek hḗlios, 'sun'",18,1,4.002602(2)[III][V],0.0001785--сокращено--
3,Li,Lithium,"Greek líthos, 'stone'",1,2,6.94[III][IV][V][VIII][VI],--сокращено--
```

Модуль `csv` языка Python очень облегчает импорт данных из `.csv`-файлов в список списков строк, как видно из строк кода с 15-й по 18-ю. В строках с 32-й по 58-ю этот список списков преобразуется в ассоциативный массив, позволяя оставшейся части программы легко извлекать информацию по названию элемента или его атомному числу.

```
 1. """Периодическая таблица элементов, (c) Эл Свейгарт al@inventwithpython.com
 2. Отображает информацию обо всех элементах
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, научная"""
 5.
 6. # Данные из https://en.wikipedia.org/wiki/List_of_chemical_elements
 7. # Выделите таблицу, скопируйте ее и вставьте в программу для работы
 8. # с электронными таблицами, например Excel или Google-таблицы, как
```

```
# в https://invpy.com/elements. Сохраните файл как periodictable.csv.
# Или просто скачайте его по адресу https://invpy.com/periodictable.csv

import csv, sys, re

# Считываем все данные из periodictable.csv.
elementsFile = open('periodictable.csv', encoding='utf-8')
elementsCsvReader = csv.reader(elementsFile)
elements = list(elementsCsvReader)
elementsFile.close()

ALL_COLUMNS = ['Atomic Number', 'Symbol', 'Element', 'Origin of name',
               'Group', 'Period', 'Atomic weight', 'Density',
               'Melting point', 'Boiling point',
               'Specific heat capacity', 'Electronegativity',
               'Abundance in earth\'s crust']

# Для выравнивания текста по ширине нужно найти самую длинную строку
# в ALL_COLUMNS.
LONGEST_COLUMN = 0
for key in ALL_COLUMNS:
    if len(key) > LONGEST_COLUMN:
        LONGEST_COLUMN = len(key)
# Помещаем все данные об элементах в структуру данных:
ELEMENTS = {}  # Структура данных со всеми данными об элементах.
for line in elements:
    element = {'Atomic Number':  line[0],
               'Symbol':         line[1],
               'Element':        line[2],
               'Origin of name': line[3],
               'Group':          line[4],
               'Period':         line[5],
               'Atomic weight':  line[6] + ' u', # атомная единица массы
               'Density':        line[7] + ' g/cm^3', # граммов/куб. см
               'Melting point':  line[8] + ' K', # градусов по Кельвину
               'Boiling point':  line[9] + ' K', # градусов по Кельвину
               'Specific heat capacity':      line[10] + ' J/(g*K)',
               'Electronegativity':           line[11],
               'Abundance in earth\'s crust': line[12] + ' mg/kg'}

    # Часть данных включает текст в квадратных скобках из "Википедии",
    # который необходимо удалить, например атомный вес бора:
    # вместо "10.81[III][IV][V][VI]" должно быть "10.81"

    for key, value in element.items():
        # Удаляем римские цифры в квадратных скобках:
        element[key] = re.sub(r'\[(I|V|X)+\]', '', value)

    ELEMENTS[line[0]] = element  # Сопоставляем атомный номер и элемент.
    ELEMENTS[line[1]] = element  # Сопоставляем символ и элемент.

```

```
60. print('Periodic Table of Elements')
61. print('By Al Sweigart al@inventwithpython.com')
62. print()
63.
64. while True:  # Основной цикл программы.
65.     # Выводим таблицу и позволяем пользователю выбрать элемент:
66.     print('''            Periodic Table of Elements
67.       1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
68.     1 H                                                  He
69.     2 Li Be                                 B  C  N  O  F  Ne
70.     3 Na Mg                                 Al Si P  S  Cl Ar
71.     4 K  Ca Sc Ti V  Cr Mn Fe Co Ni Cu Zn Ga Ge As Se Br Kr
72.     5 Rb Sr Y  Zr Nb Mo Tc Ru Rh Pd Ag Cd In Sn Sb Te I  Xe
73.     6 Cs Ba La Hf Ta W  Re Os Ir Pt Au Hg Tl Pb Bi Po At Rn
74.     7 Fr Ra Ac Rf Db Sg Bh Hs Mt Ds Rg Cn Nh Fl Mc Lv Ts Og
75.
76.             Ce Pr Nd Pm Sm Eu Gd Tb Dy Ho Er Tm Yb Lu
77.             Th Pa U  Np Pu Am Cm Bk Cf Es Fm Md No Lr''')
78.     print('Enter a symbol or atomic number to examine, or QUIT to quit.')
79.     response = input('> ').title()
80.
81.     if response == 'Quit':
82.         sys.exit()
83.
84.     # Отображаем информацию о выбранном элементе:
85.     if response in ELEMENTS:
86.         for key in ALL_COLUMNS:
87.             keyJustified = key.rjust(LONGEST_COLUMN)
88.             print(keyJustified + ': ' + ELEMENTS[response][key])
89.         input('Press Enter to continue...')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. К какой программной ошибке приведет замена `response == 'Quit'` в строке 81 на `response == 'quit'`?
2. Что будет, если удалить или закомментировать строки 53 и 55?

54

Поросячья латынь

«Поросячья латынь» — игра, в которой слова на английском языке преобразуются в пародию на латынь. Если слово начинается на согласную, то говорящий переставляет ее в конец слова, добавляя после нее ay. Например, pig превращается в igpay, a latin — в atinlay. Если же слово начинается на гласную, то говорящий просто добавляет в его конец yay. Например, elephant превращается в elephantyay, a umbrella — в umbrellayay.

Программа в действии

Результат выполнения `piglatin.py` выглядит следующим образом:

```
Igpay Atinlay (Pig Latin)
By Al Sweigart al@inventwithpython.com

Enter your message:
> This is a very serious message.
Isthay isyay ayay eryvay erioussay essagemay.
(Copied pig latin to clipboard.)
```

Описание работы

Функция `englishToPigLatin()` принимает на входе строковое значение с текстом на английском языке и возвращает строковое значение с соответствующим текстом на

поросячьей латыни. Функция `main()` вызывается только в случае, если пользователь напрямую запускает программу. Можете также писать собственные программы на Python, импортировать `piglatin.py` с помощью оператора `import piglatin` и вызывать функцию `englishToPigLatin()` с помощью `piglatin.englishToPigLatin()`. Такая методика повторного использования экономит время и усилия, которые понадобились бы для воссоздания этого кода своими руками.

```
 1. """Поросячья латынь, (c) Эл Свейгарт al@inventwithpython.com
 2. Переводит сообщения на английском на поросячью латынь.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, слова"""
 5.
 6. try:
 7.     import pyperclip  # pyperclip копирует текст в буфер обмена.
 8. except ImportError:
 9.     pass  # Если pyperclip не установлен, ничего не делаем. Не проблема.
10.
11. VOWELS = ('a', 'e', 'i', 'o', 'u', 'y')
12.
13.
14. def main():
15.     print('''Igpay Atinlay (Pig Latin)
16. By Al Sweigart al@inventwithpython.com
17.
18. Enter your message:''')
19.     pigLatin = englishToPigLatin(input('> '))
20.
21.     # Объединяем все слова обратно в одно строковое значение:
22.     print(pigLatin)
23.
24.     try:
25.         pyperclip.copy(pigLatin)
26.         print('(Copied pig latin to clipboard.)')
27.     except NameError:
28.         pass  # Если pyperclip не установлена, ничего не делаем.
29.
30.
31. def englishToPigLatin(message):
32.     pigLatin = ''  # Строковое значение с переводом на поросячью латынь.
33.     for word in message.split():
34.         # Отделяем небуквенные символы в начале слова:
35.         prefixNonLetters = ''
36.         while len(word) > 0 and not word[0].isalpha():
37.             prefixNonLetters += word[0]
38.             word = word[1:]
39.         if len(word) == 0:
40.             pigLatin = pigLatin + prefixNonLetters + ' '
41.             continue
42.
43.         # Отделяем небуквенные символы в конце слова:
```

```
44.         suffixNonLetters = ''
45.         while not word[-1].isalpha():
46.             suffixNonLetters = word[-1] + suffixNonLetters
47.             word = word[:-1]
48.         # Запоминаем, находится ли слово полностью или только первые буквы
49.         # в верхнем регистре.
50.         wasUpper = word.isupper()
51.         wasTitle = word.istitle()
52.
53.         word = word.lower()  # Переводим слово в нижний регистр для перевода.
54.
55.         # Отделяем согласные буквы в начале слова:
56.         prefixConsonants = ''
57.         while len(word) > 0 and not word[0] in VOWELS:
58.             prefixConsonants += word[0]
59.             word = word[1:]
60.
61.         # Добавляем в слово "поросячье" окончание:
62.         if prefixConsonants != '':
63.             word += prefixConsonants + 'ay'
64.         else:
65.             word += 'yay'
66.         # Переводим слово полностью или только первые буквы обратно
67.         # в верхний регистр:
68.         if wasUpper:
69.             word = word.upper()
70.         if wasTitle:
71.             word = word.title()
72.
73.         # Добавляем небуквенные символы обратно в начало слова.
74.         pigLatin += prefixNonLetters + word + suffixNonLetters + ' '
75.     return pigLatin
76.
77.
78. if __name__ == '__main__':
79.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `message.split()` в строке 33 заменить на `message`?
2. Что будет, если `('a', 'e', 'i', 'o', 'u', 'y')` в строке 11 заменить на `()`?
3. Что будет, если `('a', 'e', 'i', 'o', 'u', 'y')` в строке 11 заменить на `('A', 'E', 'I', 'O', 'U', 'Y')`?

55

Лотерея Powerball

Лотерея Powerball — замечательный способ потерять немного денег. Купив билет за 2 доллара, вы можете выбрать шесть номеров: пять из диапазона от 1 до 69, и шестой номер Powerball — от 1 до 26. Порядок номеров значения не имеет. Если выпадут выбранные вами числа — вы выиграете 1 миллиард 586 миллионов долларов! Правда, вы не выиграете, поскольку на это есть только один шанс из 292 201 338. Но если вы потратите 200 долларов на 100 билетов, то ваши шансы улучшатся до 1 из 2 922 013. Вы тоже не выиграете, но зато потратите в 100 раз больше денег. Чем больше вы любите терять деньги, тем интереснее участвовать в лотерее!

Чтобы наглядно показать вам, насколько часто вы будете проигрывать в лотерее, эта программа моделирует миллион розыгрышей Powerball, а затем сравнивает их с выбранными вами номерами. Теперь вы сможете совершенно бесплатно получить удовольствие от проигрыша в лотерее.

Любопытный факт: шансы выиграть у всех наборов из шести номеров одинаковы. Так что когда в следующий раз будете покупать лотерейный билет, выберите номера 1, 2, 3, 4, 5 и 6. Вероятность их выпадения точно такая же, как и у любого другого набора.

Программа в действии

Результат выполнения powerballlottery.py выглядит следующим образом:

```
Powerball Lottery, by Al Sweigart al@inventwithpython.com

Each powerball lottery ticket costs $2. The jackpot for this game
is $1.586 billion! It doesn't matter what the jackpot is, though,
because the odds are 1 in 292,201,338, so you won't win.

This simulation gives you the thrill of playing without wasting money.

Enter 5 different numbers from 1 to 69, with spaces between
each number. (For example: 5 17 23 42 50 51)
> 1 2 3 4 5
Enter the powerball number from 1 to 26.
> 6
How many times do you want to play? (Max: 1000000)
> 1000000
It costs $2000000 to play 1000000 times, but don't
worry. I'm sure you'll win it all back.
Press Enter to start...
The winning numbers are: 12 29 48 11 4 and 13  You lost.
The winning numbers are: 54 39 3 42 16 and 12  You lost.
The winning numbers are: 56 4 63 23 38 and 24  You lost.
--сокращено--
The winning numbers are: 46 29 10 62 17 and 21 You lost.
The winning numbers are: 5 20 18 65 30 and 10  You lost.
The winning numbers are: 54 30 58 10 1 and 18  You lost.
You have wasted $2000000
Thanks for playing!
```

Описание работы

Результаты работы этой программы выглядят распределенными вполне однообразно, поскольку код `allWinningNums.ljust(21)` в строке 109 дополняет числа пробелами так, чтобы они занимали 21 столбец, вне зависимости от количества цифр в выигравших номерах. В результате надпись You lost всегда появляется в одном месте экрана и хорошо читается, даже когда программа быстро выводит на экран много строк.

```
1. """Лотерея Powerball, (c) Эл Свейгарт al@inventwithpython.com
2. Моделирование лотереи, позволяющее почувствовать все удовольствие от
3. проигрыша в лотерее, не тратя понапрасну денег.
4. Код размещен на https://nostarch.com/big-book-small-python-projects
5. Теги: короткая, юмор, имитационное моделирование"""
6.
7. import random
```

```
 8.
 9. print('''Powerball Lottery, by Al Sweigart al@inventwithpython.com
10.
11. Each powerball lottery ticket costs $2. The jackpot for this game
12. is $1.586 billion! It doesn't matter what the jackpot is, though,
13. because the odds are 1 in 292,201,338, so you won't win.
14.
15. This simulation gives you the thrill of playing without wasting money.
16. ''')
17.
18. # Запрашиваем у игрока первые пять номеров, от 1 до 69:
19. while True:
20.     print('Enter 5 different numbers from 1 to 69, with spaces between')
21.     print('each number. (For example: 5 17 23 42 50)')
22.     response = input('> ')
23.
24.     # Убеждаемся, что игрок ввел пять номеров:
25.     numbers = response.split()
26.     if len(numbers) != 5:
27.         print('Please enter 5 numbers, separated by spaces.')
28.         continue
29.
30.     # Преобразуем строковые значения в числа:
31.     try:
32.         for i in range(5):
33.             numbers[i] = int(numbers[i])
34.     except ValueError:
35.         print('Please enter numbers, like 27, 35, or 62.')
36.         continue
37.
38.     # Проверяем, что номера — в диапазоне от 1 до 69:
39.     for i in range(5):
40.         if not (1 <= numbers[i] <= 69):
41.             print('The numbers must all be between 1 and 69.')
42.             continue
43.
44.     # Убеждаемся, что номера не повторяются:
45.     # (Создаем из номеров множество, чтобы убрать дубликаты.)
46.     if len(set(numbers)) != 5:
47.         print('You must enter 5 different numbers.')
48.         continue
49.
50.     break
51.
52. # Просим игрока выбрать Powerball, от 1 до 26:
53. while True:
54.     print('Enter the powerball number from 1 to 26.')
55.     response = input('> ')
56.
57.     # Преобразуем строковые значения в числа:
58.     try:
```

```
        powerball = int(response)
    except ValueError:
        print('Please enter a number, like 3, 15, or 22.')
        continue

    # Проверяем, что номер – в диапазоне от 1 до 26:
    if not (1 <= powerball <= 26):
        print('The powerball number must be between 1 and 26.')
        continue

    break

# Введите количество розыгрышей лотереи:
while True:
    print('How many times do you want to play? (Max: 1000000)')
    response = input('> ')

    # Преобразуем строковые значения в числа:
    try:
        numPlays = int(response)
    except ValueError:
        print('Please enter a number, like 3, 15, or 22000.')
        continue

    # Убеждаемся, что число – в диапазоне от 1 до 1 000 000:
    if not (1 <= numPlays <= 1000000):
        print('You can play between 1 and 1000000 times.')
        continue

    break

# Запускаем имитационное моделирование:
price = '$' + str(2 * numPlays)
print('It costs', price, 'to play', numPlays, 'times, but don\'t')
print('worry. I\'m sure you\'ll win it all back.')
input('Press Enter to start...')

possibleNumbers = list(range(1, 70))
for i in range(numPlays):
    # Выбираем выигравшие номера:
    random.shuffle(possibleNumbers)
    winningNumbers = possibleNumbers[0:5]
    winningPowerball = random.randint(1, 26)

    # Отображаем выигравшие номера:
    print('The winning numbers are: ', end='')
    allWinningNums = ''
    for i in range(5):
        allWinningNums += str(winningNumbers[i]) + ' '
    allWinningNums += 'and ' + str(winningPowerball)
    print(allWinningNums.ljust(21), end='')
```

```
110.
111.     # Множества не упорядочены, так что порядок целочисленных
112.     # значений в set(numbers) и set(winningNumbers) неважен.
113.     if (set(numbers) == set(winningNumbers)
114.         and powerball == winningPowerball):
115.             print()
116.             print('You have won the Powerball Lottery! Congratulations,')
117.             print('you would be a billionaire if this was real!')
118.             break
119.     else:
120.         print(' You lost.')  # Пробел в начале фразы тут необходим.
121.
122. print('You have wasted', price)
123. print('Thanks for playing!')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `possibleNumbers[0:5]` в строке 100 заменить на `numbers`, а `random.randint(1, 26)` в строке 101 — на `powerball`?
2. Какое сообщение об ошибке вы получите, если удалите или закомментируете `possibleNumbers = list(range(1, 70))` в строке 96?

56

Простые числа

Простым называется число, делящееся нацело только на единицу и на себя самое. Простые числа применяются на практике во множестве сфер, но предсказать их не способен никакой алгоритм; приходится вычислять их по одному. Впрочем, доказано, что существует бесконечное множество простых чисел.

Эта программа предназначена для поиска простых чисел с помощью прямого поиска. Ее код схож с кодом программы проекта 24 (простое число можно еще описать как такое, единственными множителями которого являются единица и оно само). Узнать больше о простых числах вы можете в статье «Википедии»: https://ru.wikipedia.org/wiki/Простое_число.

Программа в действии

Результат выполнения `primenumbers.py` выглядит следующим образом:

```
Prime Numbers, by Al Sweigart al@inventwithpython.com
--сокращено--
Enter a number to start searching for primes from:
(Try 0 or 1000000000000 (12 zeros) or another number.)
> 0
Press Ctrl-C at any time to quit. Press Enter to begin...
2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71,
73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
```

```
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233,
239, 241, 251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317,
331, 337, 347, 349, 353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419,
421, 431, 433, 439, 443, 449, 457, 461, 463, 467, 479, 487, 491, 499, 503,
509, 521, 523, 541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607,
613, 617, 619, 631, 641, 643, 647, --сокращено--
```

Описание работы

Функция `isPrime()` принимает на входе целое число и возвращает `True`, если оно простое, и `False` в противном случае. Чтобы лучше разобраться в этой программе, имеет смысл изучить проект 24. Фактически функция `isPrime()` ищет множители заданного числа и возвращает `False` в случае их обнаружения.

Алгоритм этой программы может быстро находить большие простые числа. В числе 1 триллион всего десять цифр[1]. Но, чтобы найти простые числа порядка гугола (единица со ста нулями), понадобится более продвинутый алгоритм, например тест Миллера — Рабина. Реализацию данного алгоритма на языке Python можно найти в моей книге *Cracking Codes with Python*.

```
 1. """Простые числа, (c) Эл Свейгарт al@inventwithpython.com
 2. Вычисляет простые числа — числа, делящиеся нацело только на единицу
 3. и на себя самое. Они применяются на практике во множестве сфер.
 4.
 5. Больше информации — в статье https://ru.wikipedia.org/wiki/Простое_число
 6. Код размещен на https://nostarch.com/big-book-small-python-projects
 7. Теги: крошечная, математическая, прокрутка"""
 8.
 9. import math, sys
10.
11. def main():
12.     print('Prime Numbers, by Al Sweigart al@inventwithpython.com')
13.     print('Prime numbers are numbers that are only evenly divisible by')
14.     print('one and themselves. They are used in a variety of practical')
15.     print('applications, but cannot be predicted. They must be')
16.     print('calculated one at a time.')
17.     print()
18.     while True:
19.         print('Enter a number to start searching for primes from:')
20.         print('(Try 0 or 1000000000000 (12 zeros) or another number.)')
21.         response = input('> ')
22.         if response.isdecimal():
23.             num = int(response)
24.             break
25.
26.     input('Press Ctrl-C at any time to quit. Press Enter to begin...')
```

[1] Автор ошибся, на самом деле 13, но это тоже относительно немного. — *Примеч. пер.*

```
27.
28.     while True:
29.         # Выводит все найденные простые числа:
30.         if isPrime(num):
31.             print(str(num) + ', ', end='', flush=True)
32.         num = num + 1  # Переходим к следующему числу.
33.
34.
35. def isPrime(number):
36.     """Возвращает True, если число простое, в противном случае – False."""
37.     # Обрабатываем частные случаи:
38.     if number < 2:
39.         return False
40.     elif number == 2:
41.         return True
42.
43.     # Пытаемся разделить нацело данное число на все числа от 2
44.     # до квадратного корня из него.
45.     for i in range(2, int(math.sqrt(number)) + 1):
46.         if number % i == 0:
47.             return False
48.     return True
49.
50.
51. # Если программа не импортируется, а запускается, производим запуск:
52. if __name__ == '__main__':
53.     try:
54.         main()
55.     except KeyboardInterrupt:
56.         sys.exit()  # Если нажато сочетание клавиш Ctrl+C – завершаем программу.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если `response.isdecimal()` в строке 22 замените на `response` и введете нечисловое значение в качестве числа, с которого нужно начать поиск простых чисел?
2. Что будет, если `number < 2` в строке 38 заменить на `number > 2`?
3. Что будет, если `number % 1 == 0` в строке 46 заменить на `number % i != 0`?

57

Индикатор хода выполнения

Индикатор хода выполнения (progress bar) — визуальный элемент, показывающий, какая доля задачи была выполнена. Такие индикаторы часто применяются при скачивании файлов или установке программного обеспечения. В данном проекте мы создаем функцию `getProgressBar()`, возвращающую в соответствии с переданными аргументами, строковое значение с индикатором хода выполнения. Она моделирует скачивание файла, но вы можете повторно использовать код этого индикатора хода выполнения в своих проектах.

Программа в действии

Результат выполнения `progressbar.py` выглядит следующим образом:

```
Progress Bar Simulation, by Al Sweigart
[████████                                  ] 24.6% 1007/4098
```

Описание работы

Индикатор хода выполнения использует одну уловку, доступную программам, запускаемым в окне терминала. Подобно тому как существуют экранированные символы для новой строки и табуляции (`'\n'` и `'\t'` соответственно), существует и экранированный символ для Backspace (возврат на одну позицию). Если «вывести

на экран» символ возврата на одну позицию, курсор будет перемещен влево, а один ранее выведенный символ — стерт. Эта уловка работает только в рамках текущей строки, на которой находится курсор. Если выполнить код `print('Hello\b\b\b\b\bHowdy')`, то Python выведет на экран текст Hello, переместит курсор на пять позиций влево и выведет текст Howdy. Текст `Howdy` перекроет текст Hello, создавая впечатление, что было написано только `Howdy`.

Мы прибегнем к этой методике, чтобы создать однострочный динамический индикатор хода выполнения: будем выводить на экран одну версию индикатора, затем перемещать курсор на начало и выводить обновленный индикатор. С помощью данного эффекта можно генерировать любые текстовые динамические изображения, не используя модули наподобие `bext`, хотя и ограниченные одной строкой окна терминала.

После создания этой программы вы сможете отображать индикаторы хода выполнения и в других своих программах, используя оператор `import progressbar` и вывод возвращаемого `progressbar.getProgressBar()` строкового значения.

```
 1. """Моделирование индикатора хода выполнения, (c) Эл Свейгарт
 2. al@inventwithpython.com Пример динамического индикатора хода выполнения,
 3. пригодного для использования и в других программах
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, модуль"""
 6. import random, time
 7.
 8. BAR = chr(9608) # Символ 9608 – '█'
 9.
10. def main():
11.     # Имитируем скачивание:
12.     print('Progress Bar Simulation, by Al Sweigart')
13.     bytesDownloaded = 0
14.     downloadSize = 4096
15.     while bytesDownloaded < downloadSize:
16.         # "Скачиваем" случайное количество "байт":
17.         bytesDownloaded += random.randint(0, 100)
18.
19.         # Получаем строковое значение с индикатором для этой стадии выполнения:
20.         barStr = getProgressBar(bytesDownloaded, downloadSize)
21.
22.         # Не выводим символ новой строки в конце и сразу же сбрасываем
23.         # строковое значение на экран:
24.         print(barStr, end='', flush=True)
25.
26.         time.sleep(0.2)  # Небольшая пауза:
27.
28.         # Выводим символы возврата для перевода курсора в начало строки:
29.         print('\b' * len(barStr), end='', flush=True)
30.
31.
```

```
32. def getProgressBar(progress, total, barWidth=40):
33.     """Возвращает строковое значение, соответствующее индикатору хода выполнения
34.     из barWidth полосок, дошедшему до progress из общего количества total."""
35.
36.     progressBar = ''  # Индикатор хода выполнения будет строковым значением.
37.     progressBar += '['  # Добавляем левый конец индикатора хода выполнения.
38.
39.     # Убеждаемся, что progress находится между 0 и total:
40.     if progress > total:
41.         progress = total
42.     if progress < 0:
43.         progress = 0
44.
45.     # Вычисляем количество отображаемых "полосок":
46.     numberOfBars = int((progress / total) * barWidth)
47.
48.     progressBar += BAR * numberOfBars  # Добавляем индикатор хода выполнения.
49.     progressBar += ' ' * (barWidth - numberOfBars)  # Добавляем пустое
                                                        # пространство.
50.     progressBar += ']'  # Добавляем правый конец индикатора хода выполнения.
51.
52.     # Вычисляем процент завершения задачи:
53.     percentComplete = round(progress / total * 100, 1)
54.     progressBar += ' ' + str(percentComplete) + '%'  # Добавляем значение
55.                                                      # в процентах.
55.
56.     # Добавляем числовые значения:
57.     progressBar += ' ' + str(progress) + '/' + str(total)
58.
59.     return progressBar  # Возвращаем строковое значение с индикатором хода
60.                         # выполнения.
61.
62. # Если программа не импортируется, а запускается, выполняем запуск:
63. if __name__ == '__main__':
64.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создать однострочное динамическое изображение, в котором за счет чередования символов |, /, – и \ создается эффект вращения;
- создать программу для отображения бегущей строки с прокруткой, в которой текст движется слева направо;
- создать однострочное динамическое изображение с набором из четырех движущихся единым блоком знаков равенства, подобно красному лучу сканера на робомобиле из телесериала «Рыцарь дорог» или лицу роботоподобных сайлонов из телесериала «Звездный крейсер “Галактика”».

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `print('\b' * len(barStr), end='', flush=True)` в строке 29?
2. Что будет, если поменять местами строки 48 и 49?
3. Что будет, если `round(progress / total * 100, 1)` в строке 53 заменить на `round(progress / total * 100)`?

58

Радуга

«Радуга» — простая программа, отображающая многоцветную радугу, движущуюся вперед и назад по экрану. В этой программе используется тот факт, что при выводе на экран новых строк текста уже существующий текст прокручивается вверх, создавая впечатление движения. Данная программа, схожая с программой проекта 15, хорошо подходит для начинающих.

Программа в действии

Результат выполнения `rainbow.py` выглядит следующим образом (рис. 58.1).

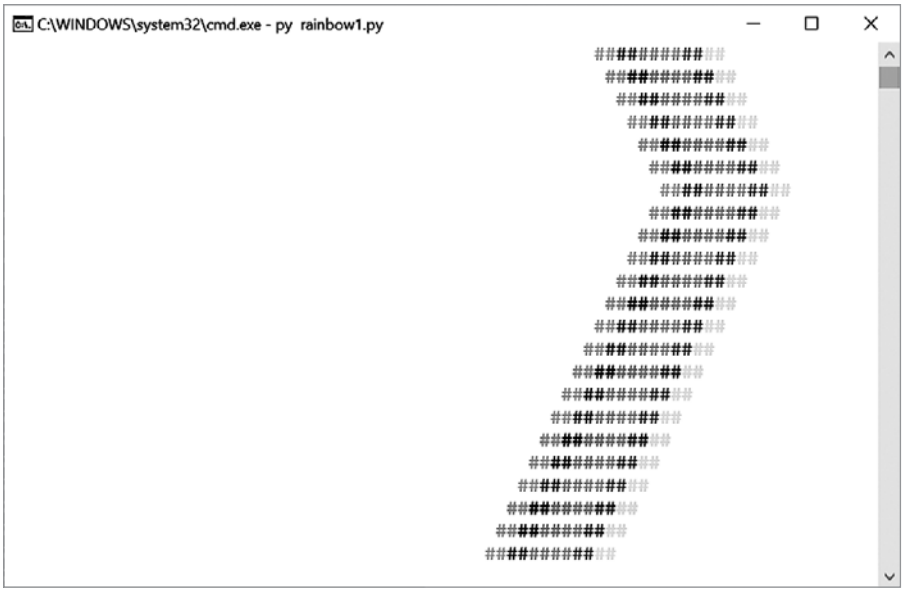


Рис. 58.1. Зигзагообразное движение радуги, на экране отображаемое в цвете

Описание работы

Программа непрерывно выводит на экран один и тот же шаблон радуги. Меняется только количество выводимых слева от него пробелов. При увеличении этого количества радуга смещается вправо, а при уменьшении — влево. Количество пробелов хранится в переменной `indent`. Значение `True` переменной `indentIncreasing` означает, что `indent` должна увеличиваться, пока не достигнет 60, после чего оно меняется на `False`. В оставшейся части кода количество пробелов уменьшается, и когда оно достигнет нуля, значение `indentIncreasing` опять меняется на `True`, чтобы повторить зигзаг радуги.

```
1. """Радуга, (с) Эл Свейгарт al@inventwithpython.com
2. Отображает простое динамическое изображение радуги. Нажмите Ctrl-C для останова.
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: крошечная, графика, bext, для начинающих, прокрутка"""
5.
6. import time, sys
```

```
 7.
 8. try:
 9.     import bext
10. except ImportError:
11.     print('This program requires the bext module, which you')
12.     print('can install by following the instructions at')
13.     print('https://pypi.org/project/Bext/')
14.     sys.exit()
15.
16. print('Rainbow, by Al Sweigart al@inventwithpython.com')
17. print('Press Ctrl-C to stop.')
18. time.sleep(3)
19.
20. indent = 0  # Количество пробелов в полях.
21. indentIncreasing = True  # Растут поля или уменьшаются.
22.
23. try:
24.     while True:  # Основной цикл программы.
25.         print(' ' * indent, end='')
26.         bext.fg('red')
27.         print('##', end='')
28.         bext.fg('yellow')
29.         print('##', end='')
30.         bext.fg('green')
31.         print('##', end='')
32.         bext.fg('blue')
33.         print('##', end='')
34.         bext.fg('cyan')
35.         print('##', end='')
36.         bext.fg('purple')
37.         print('##')
38.
39.         if indentIncreasing:
40.             # Увеличиваем количество пробелов:
41.             indent = indent + 1
42.             if indent == 60:  # (!) Попробуйте заменить это значение на 10 или 30.
43.                 # Меняем направление:
44.                 indentIncreasing = False
45.         else:
46.             # Уменьшаем количество пробелов:
47.             indent = indent - 1
48.             if indent == 0:
49.                 # Меняем направление:
50.                 indentIncreasing = True
51.
52.         time.sleep(0.02)  # Небольшая пауза.
53. except KeyboardInterrupt:
54.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `False` в строке 44 заменить на `True`?
2. Что будет, если заменить аргумент на `'random'` во всех вызовах `bext.fg()`?

59

Камень, ножницы, бумага

В этой версии игры на руках для двух игроков игрок сражается с компьютером. Вы можете выбрать «камень», «бумагу» или «ножницы». «Камень» бьет «ножницы», «ножницы» — «бумагу», а «бумага» бьет «камень». Ради интриги программа делает короткие паузы.

Другой вариант этой игры можно найти в проекте 60.

Программа в действии

Результат выполнения `rockpaperscissors.py` выглядит следующим образом:

```
Rock, Paper, Scissors, by Al Sweigart al@inventwithpython.com
- Rock beats scissors.
- Paper beats rocks.
- Scissors beats paper.

0 Wins, 0 Losses, 0 Ties
Enter your move: (R)ock (P)aper (S)cissors or (Q)uit
> r
ROCK versus...
1...
2...
3...
SCISSORS
You win!
```

```
1 Wins, 0 Losses, 0 Ties
Enter your move: (R)ock (P)aper (S)cissors or (Q)uit
--сокращено--
```

Описание работы

Логика игры «Камень, ножницы, бумага» довольно проста, мы реализуем ее тут с помощью операторов `if-elif`. Ради небольшой интриги в строках с 45-й по 51-ю выполняется обратный отсчет, прежде чем раскрыть ход противника, с краткими паузами между счетом, таким образом наращивая у игрока волнующее предвкушение результатов игры. Без этих пауз результаты выводились бы сразу же после ввода игроком его хода, что довольно скучно. Чтобы улучшить впечатление от игры для игрока, достаточно совсем немного кода.

```
 1. """Камень, ножницы, бумага (c) Эл Свейгарт al@inventwithpython.com
 2. Классическая азартная игра на руках
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, игра"""
 5.
 6. import random, time, sys
 7.
 8. print('''Rock, Paper, Scissors, by Al Sweigart al@inventwithpython.com
 9. - Rock beats scissors.
10. - Paper beats rocks.
11. - Scissors beats paper.
12. ''')
13.
14. # Переменные для отслеживания количества выигрышей, проигрышей и ничьих.
15. wins = 0
16. losses = 0
17. ties = 0
18.
19. while True:  # Основной цикл игры.
20.     while True:  # Запрашиваем, пока игрок не введет R, P, S или Q.
21.         print('{} Wins, {} Losses, {} Ties'.format(wins, losses, ties))
22.         print('Enter your move: (R)ock (P)aper (S)cissors or (Q)uit')
23.         playerMove = input('> ').upper()
24.         if playerMove == 'Q':
25.             print('Thanks for playing!')
26.             sys.exit()
27.
28.         if playerMove == 'R' or playerMove == 'P' or playerMove == 'S':
29.             break
30.         else:
31.             print('Type one of R, P, S, or Q.')
32.
33.     # Отображаем на экране выбранный игроком ход:
34.     if playerMove == 'R':
```

```
35.         print('ROCK versus...')
36.         playerMove = 'ROCK'
37.     elif playerMove == 'P':
38.         print('PAPER versus...')
39.         playerMove = 'PAPER'
40.     elif playerMove == 'S':
41.         print('SCISSORS versus...')
42.         playerMove = 'SCISSORS'
43.
44.     # Считаем до трех с драматическими паузами:
45.     time.sleep(0.5)
46.     print('1...')
47.     time.sleep(0.25)
48.     print('2...')
49.     time.sleep(0.25)
50.     print('3...')
51.     time.sleep(0.25)
52.
53.     # Отображаем на экране выбранный компьютером ход:
54.     randomNumber = random.randint(1, 3)
55.     if randomNumber == 1:
56.         computerMove = 'ROCK'
57.     elif randomNumber == 2:
58.         computerMove = 'PAPER'
59.     elif randomNumber == 3:
60.         computerMove = 'SCISSORS'
61.     print(computerMove)
62.     time.sleep(0.5)
63.
64.     # Отображаем и фиксируем победу/поражение/ничью:
65.     if playerMove == computerMove:
66.         print('It\'s a tie!')
67.         ties = ties + 1
68.     elif playerMove == 'ROCK' and computerMove == 'SCISSORS':
69.         print('You win!')
70.         wins = wins + 1
71.     elif playerMove == 'PAPER' and computerMove == 'ROCK':
72.         print('You win!')
73.         wins = wins + 1
74.     elif playerMove == 'SCISSORS' and computerMove == 'PAPER':
75.         print('You win!')
76.         wins = wins + 1
77.     elif playerMove == 'ROCK' and computerMove == 'PAPER':
78.         print('You lose!')
79.         losses = losses + 1
80.     elif playerMove == 'PAPER' and computerMove == 'SCISSORS':
81.         print('You lose!')
82.         losses = losses + 1
83.     elif playerMove == 'SCISSORS' and computerMove == 'ROCK':
84.         print('You lose!')
85.         losses = losses + 1
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- добавьте в игру ходы «ящерица» и «Спок». Ящерица травит Спока и ест бумагу, но ее давит камень и обезглавливают ножницы. Спок ломает ножницы и испаряет камень, но его травит ящерица, а бумага содержит улики против него;
- пусть игрок получает очко за каждую победу и теряет — за поражение. При выигрыше игрок может также выбирать «пан или пропал» и, возможно, выиграть больше очков в 2, 4, 8, 16 и т. д. раз в последующих раундах.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какое сообщение об ошибке вы получите, если `random.randint(1, 3)` в строке 54 замените на `random.randint(1, 300)`?
2. Что будет, если `playerMove == computerMove` в строке 65 заменить на `True`?

60

Камень, ножницы, бумага (беспроигрышная версия)

Этот вариант игры «Камень, ножницы, бумага» идентичен проекту 59, за исключением того что игрок всегда выигрывает. Выбирающий ход компьютера код устроен так, чтобы всегда выбирать проигрышный. Можете предложить сыграть в эту игру своим друзьям, наверное, они будут очень радоваться... сначала. Посмотрим, сколько времени им понадобится, чтобы заметить жульничество игры в их пользу.

Программа в действии

Результат выполнения `rockpaperscissorsalwayswin.py` выглядит следующим образом:

```
Rock, Paper, Scissors, by Al Sweigart al@inventwithpython.com
- Rock beats scissors.
- Paper beats rocks.
- Scissors beats paper.

0 Wins, 0 Losses, 0 Ties
Enter your move: (R)ock (P)aper (S)cissors or (Q)uit
> p
PAPER versus...
```

```
1...
2...
3...
ROCK
You win!
1 Wins, 0 Losses, 0 Ties
Enter your move: (R)ock (P)aper (S)cissors or (Q)uit
> s
SCISSORS versus...
1...
2...
3...
PAPER
You win!
2 Wins, 0 Losses, 0 Ties
--сокращено--
SCISSORS versus...
1...
2...
3...
PAPER
You win!
413 Wins, 0 Losses, 0 Ties
Enter your move: (R)ock (P)aper (S)cissors or (Q)uit
--сокращено--
```

Описание работы

Возможно, вы обратите внимание, что эта версия программы короче, чем в проекте 59. Логично: когда не требуется выбирать ход компьютера случайным образом и вычислять результаты игры, можно убрать довольно много кода. Вдобавок отсутствуют переменные для отслеживания количества проигрышей и ничьих, ведь они все равно были бы всегда равны нулю.

```
 1. """ Камень, ножницы, бумага (беспроигрышная версия)
 2. (c) Эл Свейгарт al@inventwithpython.com
 3. Классическая азартная игра на руках, в которой вы всегда выигрываете
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, игра, юмор"""
 6.
 7. import time, sys
 8.
 9. print('''Rock, Paper, Scissors, by Al Sweigart al@inventwithpython.com
10. - Rock beats scissors.
11. - Paper beats rocks.
12. - Scissors beats paper.
13. ''')
14.
```

```
15. # Переменная для отслеживания количества выигрышей.
16. wins = 0
17.
18. while True:  # Основной цикл программы.
19.     while True:  # Запрашиваем, пока игрок не введет R, P, S или Q.
20.         print('{} Wins, 0 Losses, 0 Ties'.format(wins))
21.         print('Enter your move: (R)ock (P)aper (S)cissors or (Q)uit')
22.         playerMove = input('> ').upper()
23.         if playerMove == 'Q':
24.             print('Thanks for playing!')
25.             sys.exit()
26.
27.         if playerMove == 'R' or playerMove == 'P' or playerMove == 'S':
28.             break
29.         else:
30.             print('Type one of R, P, S, or Q.')
31.
32.     # Отображаем на экране выбранный игроком ход:
33.     if playerMove == 'R':
34.         print('ROCK versus...')
35.     elif playerMove == 'P':
36.         print('PAPER versus...')
37.     elif playerMove == 'S':
38.         print('SCISSORS versus...')
39.
40.     # Считаем до трех с драматическими паузами:
41.     time.sleep(0.5)
42.     print('1...')
43.     time.sleep(0.25)
44.     print('2...')
45.     time.sleep(0.25)
46.     print('3...')
47.     time.sleep(0.25)
48.
49.     # Отображаем на экране выбранный компьютером ход:
50.     if playerMove == 'R':
51.         print('SCISSORS')
52.     elif playerMove == 'P':
53.         print('ROCK')
54.     elif playerMove == 'S':
55.         print('PAPER')
56.
57.     time.sleep(0.5)
58.
59.     print('You win!')
60.     wins = wins + 1
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- добавьте в игру ходы «ящерица» и «Спок». Ящерица травит Спока и ест бумагу, но ее давит камень и обезглавливают ножницы. Спок ломает ножницы и испаряет камень, но его травит ящерица, а бумага содержит улики против него;
- пусть игрок получает очко за каждую победу. При выигрыше игрок может также выбирать «пан или пропал» и, конечно, выиграть больше очков в 2, 4, 8, 16 и т. д. раз в последующих раундах.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать строки с 33-й по 57-ю?
2. Что будет, если `input('> ').upper()` в строке 22 заменить на `input('> ')`?

61

Шифр ROT13

Название шифра ROT13, одного из простейших алгоритмов шифрования, означает rotate 13 spaces («сдвинуть на 13 позиций»). В этом шифре буквам от A до Z соответствуют числа от 0 до 25 таким образом, что зашифрованная буква отстоит на 13 позиций от незашифрованной: A превращается в N, B — в O и т. д. Процесс дешифрования совпадает с процессом шифрования, благодаря чему написание программы становится тривиальным. Однако и взломать этот шифр очень просто. Поэтому чаще всего ROT13 используется для сокрытия несекретной информации, например спойлеров или ответов в викторинах, чтобы просто не прочитать их случайно. Дополнительную информацию о шифре ROT13 можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/ROT13. Если вас интересует более общая информация о шифрах и их взломе, то можете прочитать мою книгу *Cracking Codes with Python*.

Программа в действии

Результат выполнения `rot13cipher.py` выглядит следующим образом:

```
ROT13 Cipher, by Al Sweigart al@inventwithpython.com

Enter a message to encrypt/decrypt (or QUIT):
> Meet me by the rose bushes tonight.
The translated message is:
```

```
Zrrg zr ol gur ebfr ohfurf gbavtug.

(Copied to clipboard.)
Enter a message to encrypt/decrypt (or QUIT):
--сокращено--
```

Описание работы

Значительная часть кода программы ROT13 совпадает с программой проекта 6, хотя она намного проще, ведь в ней всегда применяется ключ 13. А поскольку за шифрование и расшифровку отвечает один и тот же код (строки с 27-й по 39-ю), можно не спрашивать пользователя, какой режим необходим.

Единственное отличие: эта программа сохраняет регистр исходного сообщения, а не преобразует автоматически его в верхний регистр. Например, Hello в зашифрованном виде выглядит как Uryyb, а HELLO — как URYYB.

```
 1. """Шифр ROT13, (c) Эл Свейгарт al@inventwithpython.com
 2. Простейший шифр сдвига для шифрования и дешифровки текста.
 3. Подробнее — в статье https://ru.wikipedia.org/wiki/ROT13
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, криптография"""
 6.
 7. try:
 8.     import pyperclip  # pyperclip копирует текст в буфер обмена.
 9. except ImportError:
10.     pass  # Если pyperclip не установлена, ничего не делаем. Не проблема.
11.
12. # Задаем константы:
13. UPPER_LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
14. LOWER_LETTERS = 'abcdefghijklmnopqrstuvwxyz'
15.
16. print('ROT13 Cipher, by Al Sweigart al@inventwithpython.com')
17. print()
18.
19. while True:  # Основной цикл программы.
20.     print('Enter a message to encrypt/decrypt (or QUIT):')
21.     message = input('> ')
22.
23.     if message.upper() == 'QUIT':
24.         break  # Выходим из основного цикла программы.
25.
26.     # Сдвигаем буквы в сообщении на 13 позиций.
27.     translated = ''
28.     for character in message:
29.         if character.isupper():
30.             # Выполняем конкатенацию символа в верхнем регистре.
31.             transCharIndex = (UPPER_LETTERS.find(character) + 13) % 26
```

```
32.             translated += UPPER_LETTERS[transCharIndex]
33.         elif character.islower():
34.             # Выполняем конкатенацию символа в нижнем регистре.
35.             transCharIndex = (LOWER_LETTERS.find(character) + 13) % 26
36.             translated += LOWER_LETTERS[transCharIndex]
37.         else:
38.             # Выполняем конкатенацию символа в исходном виде.
39.             translated += character
40.
41.     # Отображаем преобразованное сообщение:
42.     print('The translated message is:')
43.     print(translated)
44.     print()
45.
46.     try:
47.         # Копируем преобразованное сообщение в буфер обмена:
48.         pyperclip.copy(translated)
49.         print('(Copied to clipboard.)')
50.     except:
51.         pass
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `character.isupper()` в строке 29 заменить на `character.islower()`?
2. Что будет, если `print(translated)` в строке 43 заменить на `print(message)`?

62

Вращающийся куб

Этот проект демонстрирует на экране динамическое изображение вращающегося трехмерного куба, используя тригонометрические функции. При желании вы можете приспособить математику вращения трехмерной точки и функцию `line()` к собственным программам.

И хотя текстовые символы блоков, с помощью которых мы будем рисовать куб, не слишком напоминают тонкие прямые линии, подобные рисунки называются *каркасной моделью* (wireframe model), поскольку визуализируют только ребра поверхностей объекта. На рис. 62.1 показана каркасная модель куба и икосферы — приближения сферы, составленной из треугольников.

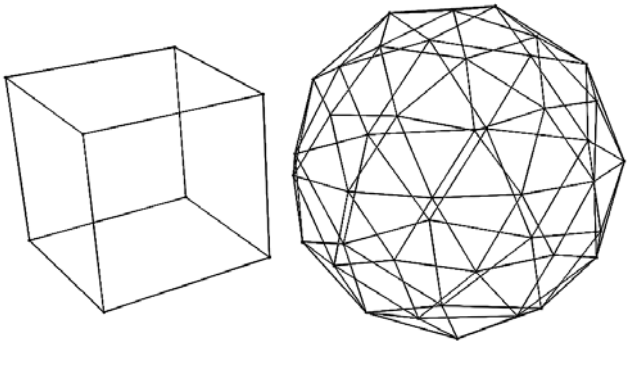


Рис. 62.1. Каркасные модели для куба *(слева)* и икосферы *(справа)*

Программа в действии

На рис. 62.2 показан результат выполнения `rotatingcube.py`.

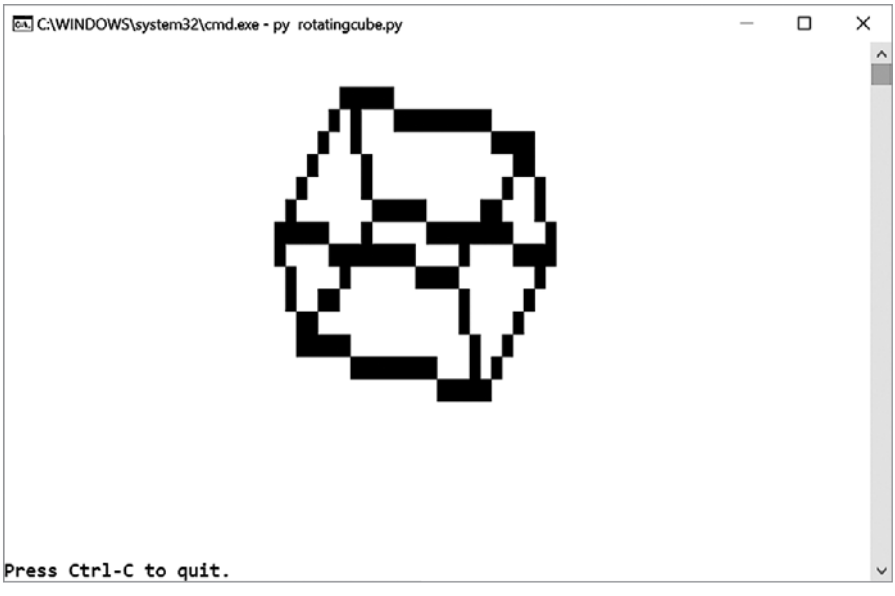


Рис. 62.2. Каркасная модель куба, которую наша программа рисует на экране

Описание работы

Наш алгоритм состоит из двух основных составляющих: функции `line()` и функции `rotatePoint()`. У куба — восемь точек, по одной для каждого угла. Программа хранит эти углы в виде кортежей `(x, y, z)` в списке `CUBE_CORNERS`. Кроме того, эти точки задают связи граней куба. При вращении всех точек в одном направлении на одинаковый угол возникает иллюзия вращения куба.

```
"""Вращающийся куб, (c) Эл Свейгарт al@inventwithpython.com
Динамическое изображение вращающегося куба. Нажмите Ctrl+C для останова.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, графика, математическая"""

# Эту программу следует запускать в окне терминала/командной оболочки.

import math, time, sys, os

# Задаем константы:
```

```
11. PAUSE_AMOUNT = 0.1  # Пауза на 1/10 секунды.
12. WIDTH, HEIGHT = 80, 24
13. SCALEX = (WIDTH - 4) // 8
14. SCALEY = (HEIGHT - 4) // 8
15. # Высота текстовых ячеек в два раза превышает ширину,
    # так что задаем масштаб по оси y:
16. SCALEY *= 2
17. TRANSLATEX = (WIDTH - 4) // 2
18. TRANSLATEY = (HEIGHT - 4) // 2
19.
20. # (!) Попробуйте заменить это значение на '#', или '*',
    # или еще какой-либо символ:
21. LINE_CHAR = chr(9608)  # Символ 9608 — '█'
22.
23. # (!) Попробуйте обнулить два из этих значений, чтобы вращать куб
24. # относительно только одной оси координат:
25. X_ROTATE_SPEED = 0.03
26. Y_ROTATE_SPEED = 0.08
27. Z_ROTATE_SPEED = 0.13
28.
29. # Координаты XYZ данная программа хранит в списках, координату X —
30. # по индексу 0, Y — 1, а Z — 2. Эти константы повышают удобочитаемость
31. # кода при обращении к координатам в этих списках.
32. X = 0
33. Y = 1
34. Z = 2
35.
36.
37. def line(x1, y1, x2, y2):
38.     """Возвращает список точек, лежащих на прямой между заданными точками.
39.
40.     Использует алгоритм Брезенхэма. Подробнее — в статье на
41.     https://ru.wikipedia.org/wiki/Алгоритм_Брезенхэма"""
42.     points = []  # Содержит точки прямой.
43.     # "Steep" означает, что уклон прямой больше 45 градусов
44.     # или меньше -45 градусов:
45.
46.     # Проверяем на предмет частного случая, когда начальная и конечная точки
47.     # являются в определенном смысле соседними, что данная функция не умеет
48.     # хорошо обрабатывать, поэтому возвращаем заранее подготовленный список:
49.     if (x1 == x2 and y1 == y2 + 1) or (y1 == y2 and x1 == x2 + 1):
50.         return [(x1, y1), (x2, y2)]
51.
52.     isSteep = abs(y2 - y1) > abs(x2 - x1)
53.     if isSteep:
54.         # Этот алгоритм умеет работать только с некрутыми прямыми,
55.         # так что делаем уклон не таким крутым, а потом возвращаем обратно.
56.         x1, y1 = y1, x1  # Меняем местами x1 и y1
57.         x2, y2 = y2, x2  # Меняем местами x2 и y2
58.     isReversed = x1 > x2  # True, если прямая идет справа налево.
59.
```

```
    if isReversed:  # Получаем точки на прямой, идущей справа налево.
        x1, x2 = x2, x1  # Меняем местами x1 и x2
        y1, y2 = y2, y1  # Меняем местами y1 и y2

        deltax = x2 - x1
        deltay = abs(y2 - y1)
        extray = int(deltax / 2)
        currenty = y2
        if y1 < y2:
            ydirection = 1
        else:
            ydirection = -1
        # Вычисляем y для каждого x в этой прямой:
        for currentx in range(x2, x1 - 1, -1):
            if isSteep:
                points.append((currenty, currentx))
            else:
                points.append((currentx, currenty))
            extray -= deltay
            if extray <= 0:  # Меняем y, только если extray <= 0.
                currenty -= ydirection
                extray += deltax
    else:  # Получаем точки на прямой, идущей слева направо.
        deltax = x2 - x1
        deltay = abs(y2 - y1)
        extray = int(deltax / 2)
        currenty = y1
        if y1 < y2:
            ydirection = 1
        else:
            ydirection = -1
        # Вычисляем y для каждого x в этой прямой:
        for currentx in range(x1, x2 + 1):
            if isSteep:
                points.append((currenty, currentx))
            else:
                points.append((currentx, currenty))
            extray -= deltay
            if extray < 0:  # Меняем y, только если extray < 0.
                currenty += ydirection
                extray += deltax
    return points


def rotatePoint(x, y, z, ax, ay, az):
    """Возвращает кортеж (x, y, z) из повернутых аргументов x, y, z.

    Вращение на углы ax, ay, az (в радианах) относительно начала
    координат 0, 0, 0.
        Направления осей координат:
         -y
```

```
111.           |
112.           +-- +x
113.          /
114.         +z
115.     """
116.
117.     # Вращаем относительно оси x:
118.     rotatedX = x
119.     rotatedY = (y * math.cos(ax)) - (z * math.sin(ax))
120.     rotatedZ = (y * math.sin(ax)) + (z * math.cos(ax))
121.     x, y, z = rotatedX, rotatedY, rotatedZ
122.
123.     # Вращаем относительно оси y:
124.     rotatedX = (z * math.sin(ay)) + (x * math.cos(ay))
125.     rotatedY = y
126.     rotatedZ = (z * math.cos(ay)) - (x * math.sin(ay))
127.     x, y, z = rotatedX, rotatedY, rotatedZ
128.
129.     # Вращаем относительно оси z:
130.     rotatedX = (x * math.cos(az)) - (y * math.sin(az))
131.     rotatedY = (x * math.sin(az)) + (y * math.cos(az))
132.     rotatedZ = z
133.
134.     return (rotatedX, rotatedY, rotatedZ)
135.
136.
137. def adjustPoint(point):
138.     """Преобразуем трехмерную точку XYZ в двумерную точку XY, подходящую для
139.     отображения на экране. Для этого масштабируем данную двумерную точку
140.     на SCALEX и SCALEY, а затем сдвигаем ее на TRANSLATEX и TRANSLATEY."""
141.     return (int(point[X] * SCALEX + TRANSLATEX),
142.             int(point[Y] * SCALEY + TRANSLATEY))
143.
144.
145. """В CUBE_CORNERS хранятся координаты XYZ углов куба.
146. Индексы всех углов в CUBE_CORNERS отмечены на следующей схеме:
147.       0---1
148.      /|  /|
149.     2---3 |
150.     | 4-|-5
151.     |/  |/
152.     6---7"""
153. CUBE_CORNERS = [[-1, -1, -1], # Точка 0
154.                 [ 1, -1, -1], # Точка 1
155.                 [-1, -1,  1], # Точка 2
156.                 [ 1, -1,  1], # Точка 3
157.                 [-1,  1, -1], # Точка 4
158.                 [ 1,  1, -1], # Точка 5
159.                 [-1,  1,  1], # Точка 6
160.                 [ 1,  1,  1]] # Точка 7
161. # В rotatedCorners хранятся координаты XYZ из CUBE_CORNERS
```

```
# после вращения на rx, ry и rz:
rotatedCorners = [None, None, None, None, None, None, None, None]
# Вращение для каждой оси:
xRotation = 0.0
yRotation = 0.0
zRotation = 0.0

try:
    while True:  # Основной цикл программы.
        # Вращение куба относительно различных осей на различный угол:
        xRotation += X_ROTATE_SPEED
        yRotation += Y_ROTATE_SPEED
        zRotation += Z_ROTATE_SPEED
        for i in range(len(CUBE_CORNERS)):
            x = CUBE_CORNERS[i][X]
            y = CUBE_CORNERS[i][Y]
            z = CUBE_CORNERS[i][Z]
            rotatedCorners[i] = rotatePoint(x, y, z, xRotation,
                yRotation, zRotation)

        # Получаем точки линий куба:
        cubePoints = []
        for fromCornerIndex, toCornerIndex in ((0, 1), (1, 3), (3, 2),
        (2, 0), (0, 4), (1, 5), (2, 6), (3, 7), (4, 5), (5, 7), (7, 6),
        (6, 4)):
            fromX, fromY = adjustPoint(rotatedCorners[fromCornerIndex])
            toX, toY = adjustPoint(rotatedCorners[toCornerIndex])
            pointsOnLine = line(fromX, fromY, toX, toY)
            cubePoints.extend(pointsOnLine)

        # Избавляемся от дублирующихся точек:
        cubePoints = tuple(frozenset(cubePoints))

        # Отображаем куб на экране:
        for y in range(HEIGHT):
            for x in range(WIDTH):
                if (x, y) in cubePoints:
                    # Отображаем полный блок:
                    print(LINE_CHAR, end='', flush=False)
                else:
                    # Отображаем пустое пространство:
                    print(' ', end='', flush=False)
            print(flush=False)
        print('Press Ctrl-C to quit.', end='', flush=True)

        time.sleep(PAUSE_AMOUNT)  # Небольшая пауза.

        # Очищаем экран:
        if sys.platform == 'win32':
            os.system('cls')  # В Windows для этого служит команда cls.
        else:
```

```
            os.system('clear')  # В macOS и Linux – команда clear.

except KeyboardInterrupt:
    print('Rotating Cube, by Al Sweigart al@inventwithpython.com')
    sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- модифицировать CUBE_CORNERS и кортеж в строке 184, чтобы создать различные каркасные модели, например пирамиду и плоский шестиугольник;
- увеличить координаты CUBE_CORNERS в 1,5 раза, чтобы куб вращался около центра экрана, а не вокруг собственного центра.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать строки с 208-й по 211-ю?
2. Что будет, если кортежи в строке 184 заменить на `<((0, 1), (1, 3), (3, 2), (2, 0), (0,4), (4, 5), (5, 1))>`?

63

Царская игра Ура

«Царская игра Ура» — игра 5000-летней давности из Месопотамии. Археологи обнаружили ее на Царском кладбище Ура на юге нынешнего Ирака во время раскопок в 1922–1934 годах. Правила, напоминающие парчиси («двадцать пять»), были воссозданы по игральной доске (показанной на рис. 63.1) и вавилонской глиняной табличке. Чтобы выиграть, нужны как удача, так и мастерство.

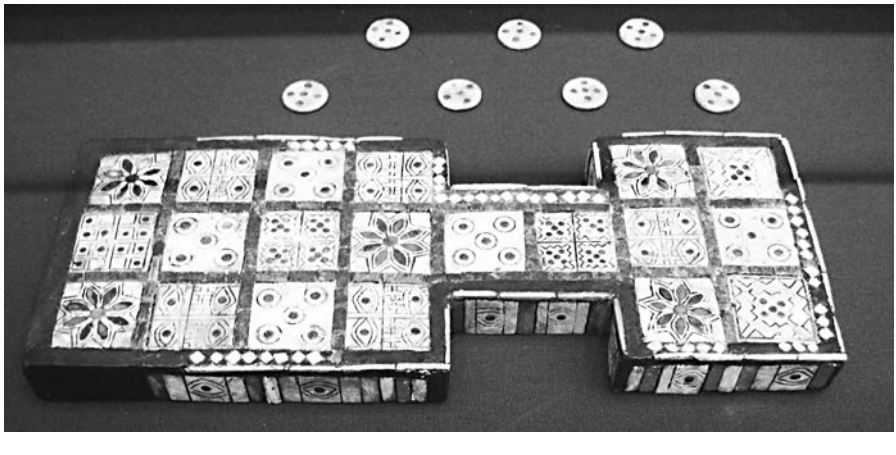


Рис. 63.1. Одна из пяти игральных досок, найденных на Царском кладбище в Уре

Два игрока начинают с пятью фишками в доме, побеждает тот, кто первый доведет все семь до финиша. Игроки по очереди бросают четыре игральные кости, представляющие собой четырехконечные пирамиды — тетраэдры. У каждой кости есть две помеченные вершины, так что с равной вероятностью может выпасть помеченная или непомеченная вершина. Вместо игральных костей в нашей игре используются монеты, а роль помеченной вершины играет решка. Каждая выпавшая помеченная вершина дает игроку право передвинуть фишку на одну клетку. Это значит, что потенциально можно передвинуть одну фишку на четыре клетки, хотя две — более вероятный вариант.

Фишки движутся по показанному на рис. 63.2 пути. В клетке может одновременно находиться только одна фишка. Если фишка попадает на поле, уже занятое фишкой противника, то последняя удаляется обратно в свой дом. Фишка, попавшая на розетку посередине доски, считается защищенной от удаления. Если же она попадает на любую из четырех остальных розеток, то игроку предоставляется право дополнительного хода. В нашей игре для представления фишек будут использоваться буквы X и O.

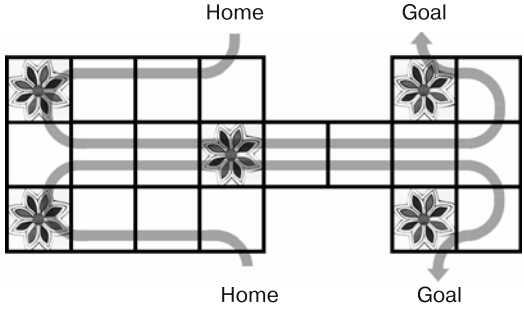


Рис. 63.2. Путь фишек игроков от дома до финиша

На этом видео YouTube-блогер Том Скотт (Tom Scott) и куратор Британского музея Ирвин Финкель (Irving Finkel) обсуждают «Царскую игру Ура»: https://www.youtube.com/watch?v=WZskjLq040I.

Программа в действии

Результат выполнения `royalgameofur.py` выглядит следующим образом:

```
The Royal Game of Ur, by Al Sweigart
--сокращено--
                   XXXXXXX           .......
                   Home               Goal
                     v                  ^
+-----+-----+-----+--v--+           +--^--+-----+
|*****|     |     |     |           |*****|     |
|*   *<     <     <     |           |*   *<     |
|****h|    g|    f|    e|           |****t|    s|
+--v--+-----+-----+-----+-----+-----+-----+--^--+
|     |     |     |*****|     |     |     |     |
|     >     >     >*   *>     >     >     >     |
|    i|    j|    k|****l|    m|    n|    o|    p|
+--^--+-----+-----+-----+-----+-----+-----+--v--+
|*****|     |     |     |           |*****|     |
|*   *<     <     <     |           |*   *<     |
|****d|    c|    b|    a|           |****r|    q|
+-----+-----+-----+--^--+           +--v--+-----+
                     ^                  v
                   Home               Goal
                   OOOOOOO           .......

It is O's turn. Press Enter to flip...
Flips: H-H-H-H  Select token to move 4 spaces: home quit
> home
O landed on a flower space and gets to go again.
Press Enter to continue...
--сокращено--
```

Описание работы

Подобно проекту 43, клетки на доске, нарисованной с помощью ASCII-графики, помечены буквами от a до t. Бросив кости, игрок может выбрать клетку со своей фишкой, чтобы пойти ею, либо выбрать `home`, чтобы начать ходить фишкой из дома на доску. Доска в программе представлена с помощью ассоциативного массива с ключами от `'a'` до `'t'` и значениями `'X'` и `'O'` для фишек (и `' '` для пустых клеток).

Кроме того, в этом ассоциативном массиве есть ключи `'x_home'`, `'o_home'`, `'x_goal'` и `'o_goal'`, значениями для которых служат строки из семи символов, отражающие заполнение домов и финишных полей. Символы `'X'` и `'O'` в этих строковых значениях обозначают фишки в доме или на финише, а `'.'` обозначает незанятое место. Функция `displayBoard()` отображает эти строковые значения из семи символов на экране.

```
 1. """Царская игра Ура, (c) Эл Свейгарт al@inventwithpython.com
 2. Игра 5000-летней давности из Месопотамии. Два игрока бьют фишки
 3. друг друга, стремясь достичь финиша.
 4. Подробнее — в статье на https://ru.wikipedia.org/wiki/Царская_игра_Ура
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: большая, настольная игра, игра, для двух игроков
 7. """
 8.
 9. import random, sys
10.
11. X_PLAYER = 'X'
12. O_PLAYER = 'O'
13. EMPTY = ' '
14.
15. # Задаем константы для меток клеток доски:
16. X_HOME = 'x_home'
17. O_HOME = 'o_home'
18. X_GOAL = 'x_goal'
19. O_GOAL = 'o_goal'
20.
21. # Клетки доски слева направо, сверху вниз:
22. ALL_SPACES = 'hgfetsijklmnopdcbarq'
23. X_TRACK = 'HefghijklmnopstG'  # (H означает Дом, G означает Финиш.)
24. O_TRACK = 'HabcdijklmnopqrG'
25.
26. FLOWER_SPACES = ('h', 't', 'l', 'd', 'r')
27.
28. BOARD_TEMPLATE = """
29.                    {}           {}
30.                    Home              Goal
31.                      v                 ^
32. +-----+-----+-----+--v--+           +--^--+-----+
33. |*****|     |     |     |           |*****|     |
34. |* {} *<  {}  <  {}  <  {}  |           |* {} *<  {}  |
35. |****h|    g|    f|    e|           |****t|    s|
36. +--v--+-----+-----+-----+-----+-----+-----+--^--+
37. |     |     |     |*****|     |     |     |     |
38. |  {}  >  {}  >  {}  >* {} *>  {}  >  {}  >  {}  >  {}  |
39. |    i|    j|    k|****l|    m|    n|    o|    p|
40. +--^--+-----+-----+-----+-----+-----+-----+--v--+
41. |*****|     |     |     |           |*****|     |
42. |* {} *<  {}  <  {}  <  {}  |           |* {} *<  {}  |
43. |****d|    c|    b|    a|           |****r|    q|
44. +-----+-----+-----+--^--+           +--v--+-----+
45.                      ^                 v
46.                    Home              Goal
47.                    {}           {}
48. """
49.
50.
51. def main():
52.     print('''The Royal Game of Ur, by Al Sweigart
```

```
53.
54. This is a 5,000 year old game. Two players must move their tokens
55. from their home to their goal. On your turn you flip four coins and can
56. move one token a number of spaces equal to the heads you got.
57.
58. Ur is a racing game; the first player to move all seven of their tokens
59. to their goal wins. To do this, tokens must travel from their home to
60. their goal:
61.
62.             X Home      X Goal
63.               v           ^
64. +---+---+---+-v-+       +-^-+---+
65. |v<<<<<<<<<<<< |       | ^<|<< |
66. |v  |   |   |   |       |   | ^ |
67. +v--+---+---+---+---+---+---+-^-+
68. |>>>>>>>>>>>>>>>>>>>>>>>>>>>>^ |
69. |>>>>>>>>>>>>>>>>>>>>>>>>>>>>v |
70. +^--+---+---+---+---+---+---+-v-+
71. |^  |   |   |   |       |   | v |
72. |^<<<<<<<<<<<< |       | v<<<< |
73. +---+---+---+-^-+       +-v-+---+
74.               ^           v
75.             O Home      O Goal
76.
77. If you land on an opponent's token in the middle track, it gets sent
78. back home. The **flower** spaces let you take another turn. Tokens in
79. the middle flower space are safe and cannot be landed on.''')
80.     input('Press Enter to begin...')
81.
82.     gameBoard = getNewBoard()
83.     turn = O_PLAYER
84.     while True:  # Основной цикл игры.
85.         # Переменные для этого хода:
86.         if turn == X_PLAYER:
87.             opponent = O_PLAYER
88.             home = X_HOME
89.             track = X_TRACK
90.             goal = X_GOAL
91.             opponentHome = O_HOME
92.         elif turn == O_PLAYER:
93.             opponent = X_PLAYER
94.             home = O_HOME
95.             track = O_TRACK
96.             goal = O_GOAL
97.             opponentHome = X_HOME
98.
99.         displayBoard(gameBoard)
100.
101.        input('It is ' + turn + '\'s turn. Press Enter to flip...')
102.
103.        flipTally = 0
```

```
104.         print('Flips: ', end='')
105.         for i in range(4):  # Подбрасываем четыре монеты.
106.             result = random.randint(0, 1)
107.             if result == 0:
108.                 print('T', end='')  # Решка.
109.             else:
110.                 print('H', end='')  # Орел.
111.             if i != 3:
112.                 print('-', end='')  # Разделитель.
113.             flipTally += result
114.         print('  ', end='')
115.
116.         if flipTally == 0:
117.             input('You lose a turn. Press Enter to continue...')
118.             turn = opponent  # Переход хода к другому игроку.
119.             continue
120.
121.         # Просим игрока сделать ход:
122.         validMoves = getValidMoves(gameBoard, turn, flipTally)
123.
124.         if validMoves == []:
125.             print('There are no possible moves, so you lose a turn.')
126.             input('Press Enter to continue...')
127.             turn = opponent  # Переход хода к другому игроку.
128.             continue
129.
130.         while True:
131.             print('Select move', flipTally, 'spaces: ', end='')
132.             print(' '.join(validMoves) + ' quit')
133.             move = input('> ').lower()
134.
135.             if move == 'quit':
136.                 print('Thanks for playing!')
137.                 sys.exit()
138.             if move in validMoves:
139.                 break  # Выход из цикла после выбора допустимого хода.
140.
141.             print('That is not a valid move.')
142.         # Производим выбранный ход на доске:
143.         if move == 'home':
144.             # Если ход делается из дома, вычитаем соответствующее
145.             # количество фишек:
146.             gameBoard[home] -= 1
147.             nextTrackSpaceIndex = flipTally
148.         else:
149.             gameBoard[move] = EMPTY  # Очищаем клетку, из которой делался ход.
150.             nextTrackSpaceIndex = track.index(move) + flipTally
151.
152.         movingOntoGoal = nextTrackSpaceIndex == len(track) - 1
153.         if movingOntoGoal:
154.             gameBoard[goal] += 1
```

```
155.             # Проверяем, не выиграл ли игрок:
156.             if gameBoard[goal] == 7:
157.                 displayBoard(gameBoard)
158.                 print(turn, 'has won the game!')
159.                 print('Thanks for playing!')
160.                 sys.exit()
161.         else:
162.             nextBoardSpace = track[nextTrackSpaceIndex]
163.             # Проверяем наличие в клетке фишки оппонента:
164.             if gameBoard[nextBoardSpace] == opponent:
165.                 gameBoard[opponentHome] += 1
166.
167.             # Заносим фишку игрока в клетку, в которую произведен ход:
168.             gameBoard[nextBoardSpace] = turn
169.
170.         # Проверяем, попал ли игрок на розетку, а значит, может ходить снова:
171.         if nextBoardSpace in FLOWER_SPACES:
172.             print(turn, 'landed on a flower space and goes again.')
173.             input('Press Enter to continue...')
174.         else:
175.             turn = opponent  # Переход хода к другому игроку.
176.
177. def getNewBoard():
178.     """
179.     Возвращает ассоциативный массив, отражающий состояние доски.
180.     Ключами служат строковые значения меток клеток, значения —
181.     X_PLAYER, O_PLAYER или EMPTY. Также содержит счетчики количества
182.     фишек в доме и на финише для обоих игроков.
183.     """
184.     board = {X_HOME: 7, X_GOAL: 0, O_HOME: 7, O_GOAL: 0}
185.     # В начале игры все клетки пусты:
186.     for spaceLabel in ALL_SPACES:
187.         board[spaceLabel] = EMPTY
188.     return board
189.
190.
191. def displayBoard(board):
192.     """Отображает игральную доску на экране."""
193.     # Очищает экран путем вывода множества символов новой строки,
194.     # делая старую доску невидимой для пользователя.
195.     print('\n' * 60)
196.
197.     xHomeTokens = ('X' * board[X_HOME]).ljust(7, '.')
198.     xGoalTokens = ('X' * board[X_GOAL]).ljust(7, '.')
199.     oHomeTokens = ('O' * board[O_HOME]).ljust(7, '.')
200.     oGoalTokens = ('O' * board[O_GOAL]).ljust(7, '.')
201.
202.     # Добавляем строковые значения для заполнения BOARD_TEMPLATE
203.     # по порядку слева направо, сверху вниз.
204.     spaces = []
205.     spaces.append(xHomeTokens)
```

```
206.     spaces.append(xGoalTokens)
207.     for spaceLabel in ALL_SPACES:
208.         spaces.append(board[spaceLabel])
209.     spaces.append(oHomeTokens)
210.     spaces.append(oGoalTokens)
211.
212.     print(BOARD_TEMPLATE.format(*spaces))
213.
214.
215. def getValidMoves(board, player, flipTally):
216.     validMoves = []  # Клетки с фишками, способными двигаться.
217.     if player == X_PLAYER:
218.         opponent = O_PLAYER
219.         track = X_TRACK
220.         home = X_HOME
221.     elif player == O_PLAYER:
222.         opponent = X_PLAYER
223.         track = O_TRACK
224.         home = O_HOME
225.
226.     # Проверяем, может ли игрок пойти фишкой из дома:
227.     if board[home] > 0 and board[track[flipTally]] == EMPTY:
228.         validMoves.append('home')
229.
230.     # Проверяем, в каких клетках есть фишки, которыми игрок может пойти:
231.     for trackSpaceIndex, space in enumerate(track):
232.         if space == 'H' or space == 'G' or board[space] != player:
233.             continue
234.         nextTrackSpaceIndex = trackSpaceIndex + flipTally
235.         if nextTrackSpaceIndex >= len(track):
236.             # Необходимо сделать ход так, чтобы не перескочить
237.             # финишную клетку, иначе ходить нельзя.
238.             continue
239.         else:
240.             nextBoardSpaceKey = track[nextTrackSpaceIndex]
241.             if nextBoardSpaceKey == 'G':
242.                 # Эта фишка может уйти с доски:
243.                 validMoves.append(space)
244.                 continue
245.         if board[nextBoardSpaceKey] in (EMPTY, opponent):
246.             # Сделать ход в защищенную среднюю клетку можно,
247.             # только если она пуста:
248.             if nextBoardSpaceKey == 'l' and board['l'] == opponent:
249.                 continue  # Пропускаем ход, клетка защищена.
250.             validMoves.append(space)
251.
252.     return validMoves
253.
254.
255. if __name__ == '__main__':
256.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `nextTrackSpaceIndex == len(track) - 1` в строке 152 заменить на `nextTrackSpaceIndex == 1`?
2. Что будет, если `result = random.randint(0, 1)` в строке 106 заменить на `result = 1`?
3. Какая ошибка возникнет, если `board = {X_HOME: 7, X_GOAL: 0, O_HOME: 7, O_GOAL: 0}` в строке 184 заменить на `board = {}`?

64

Семисегментный модуль индикации

Семисегментный модуль индикации — LCD-компонент, используемый для отображения чисел в карманных калькуляторах, микроволновых печах и прочих мелких электронных устройствах. С помощью различных сочетаний семи отрезкоподобных сегментов на LCD семисегментный модуль индикации может отображать цифры от 0 до 9. Выглядят они следующим образом:

```
 _     _   _       _   _   _   _   _
| |  | _| _| |_| |_  |_    | |_| |_|
|_|  | |_   _|   |  _| |_|   | |_|  _|
```

Преимущество этой программы состоит в том, что другие программы могут импортировать ее в виде модуля. Проекты 14 и 19 импортируют файл `sevseg.py` и пользуются его функцией `getSevSegStr()`. Подробнее о семисегментных модулях индикации и прочих их видах можно прочитать в статье «Википедии»: https://ru.wikipedia.org/wiki/Семисегментный_индикатор.

Программа в действии

При непосредственном запуске программы, будучи модулем, `sevseg.py` демонстрирует на экране примеры генерируемых цифр. Выглядит это следующим образом:

```
This module is meant to be imported rather than run.
For example, this code:
    import sevseg
    myNumber = sevseg.getSevSegStr(42, 3)
    print(myNumber)

Will print 42, zero-padded to three digits:

 __   __
|  | |__|  __|
|__|    | |__
```

Описание работы

Сначала функция `getSevSegStr()` создает список из трех строковых значений, соответствующих верхней, средней и нижней строке цифр. В строках с 27-й по 75-ю содержится длинный перечень операторов `if-elif` для всех цифр (а также десятичной точки и знака минус), для конкатенации частей отдельных цифр в эти строковые значения. В строке 84 эти три строковых значения объединяются с символом новой строки и возвращаются функцией в виде многострочного строкового значения, подходящего для передачи функции `print()`.

```
 1. """Sevseg, (c) Эл Свейгарт al@inventwithpython.com
 2. Семисегментный модуль индикации, используемый в программах "Обратный
 3. отсчет" и "Цифровые часы"
 4. Подробнее — в статье https://ru.wikipedia.org/wiki/Семисегментный_индикатор
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, модуль"""
 7.
 8. """Семисегментный индикатор, сегменты обозначены буквами от A до G:
 9.  __A__
10. |     |    Все цифры, отображаемые на семисегментном индикаторе:
11. F     B      __        __   __        __   __  __   __   __
12. |__G__|    |  |   |  __|  __| |__| |__  |__    | |__| |__|
13. |     |    |__|   | |__   __|    |  __| |__|   | |__|  __|
14. E     C
15. |__D__|"""
16.
17.
18. def getSevSegStr(number, minWidth=0):
19.     """Возвращает строковое значение для цифры на семисегментном
20.     индикаторе. Если возвращаемое строковое значение меньше minWidth,
21.     оно дополняется нулями."""
22.     # Преобразуем число в строковое значение на случай, если это не int или
23.     # не float:
24.     number = str(number).zfill(minWidth)
25.     rows = ['', '', '']
26.     for i, numeral in enumerate(number):
27.         if numeral == '.':  # Визуализируем десятичную точку.
```

```
            rows[0] += ' '
            rows[1] += ' '
            rows[2] += '.'
            continue  # Пропускаем место между цифрами.
        elif numeral == '-':  # Выводим знак минуса:
            rows[0] += '    '
            rows[1] += ' __ '
            rows[2] += '    '
        elif numeral == '0':  # Визуализируем 0.
            rows[0] += ' __ '
            rows[1] += '|  |'
            rows[2] += '|__|'
        elif numeral == '1':  # Визуализируем 1.
            rows[0] += '    '
            rows[1] += '   |'
            rows[2] += '   |'
        elif numeral == '2':  # Визуализируем 2.
            rows[0] += ' __ '
            rows[1] += ' __|'
            rows[2] += '|__ '
        elif numeral == '3':  # Визуализируем 3.
            rows[0] += ' __ '
            rows[1] += ' __|'
            rows[2] += ' __|'
        elif numeral == '4':  # Визуализируем 4.
            rows[0] += '    '
            rows[1] += '|__|'
            rows[2] += '   |'
        elif numeral == '5':  # Визуализируем 5.
            rows[0] += ' __ '
            rows[1] += '|__ '
            rows[2] += ' __|'
        elif numeral == '6':  # Визуализируем 6.
            rows[0] += ' __ '
            rows[1] += '|__ '
            rows[2] += '|__|'
        elif numeral == '7':  # Визуализируем 7.
            rows[0] += ' __ '
            rows[1] += '   |'
            rows[2] += '   |'
        elif numeral == '8':  # Визуализируем 8.
            rows[0] += ' __ '
            rows[1] += '|__|'
            rows[2] += '|__|'
        elif numeral == '9':  # Визуализируем 9.
            rows[0] += ' __ '
            rows[1] += '|__|'
            rows[2] += ' __|'

        # Добавляем пробел (для пространства между цифрами),
        # если эта цифра — не последняя:
```

```
79.         if i != len(number) - 1:
80.             rows[0] += ' '
81.             rows[1] += ' '
82.             rows[2] += ' '
83.
84.     return '\n'.join(rows)
85.
86. # Если эта программа не импортируется, а запускается, отображаем число
87. # от 0 до 99.
88. if __name__ == '__main__':
89.     print('This module is meant to be imported rather than run.')
90.     print('For example, this code:')
91.     print('    import sevseg')
92.     print('    myNumber = sevseg.getSevSegStr(42, 3)')
93.     print('    print(myNumber)')
94.     print()
95.     print('...will print 42, zero-padded to three digits:')
96.     print(' __          __ ')
97.     print('|  | |__|  __|')
98.     print('|__|    | |__ ')
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создайте новые шрифты для цифр, например использующие пять строк вместо трех и символ блока, возвращаемый `chr(9608)`;
- прочитайте в статье «Википедии» о семисегментных индикаторах, как отображать буквы, и добавьте эту возможность в `sevseg.py`;
- прочитайте в статье «Википедии» https://en.wikipedia.org/wiki/Sixteen-segment_display о шестнадцатисегментных индикаторах и создайте модуль `sixteenseg.py` для генерации чисел в этом стиле.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если содержащие один пробел строковые значения в строках 80, 81 и 82 заменить на пустые строковые значения?
2. Что будет, если аргумент по умолчанию `minWidth=0` в строке 18 заменить на `minWidth=8`?

65

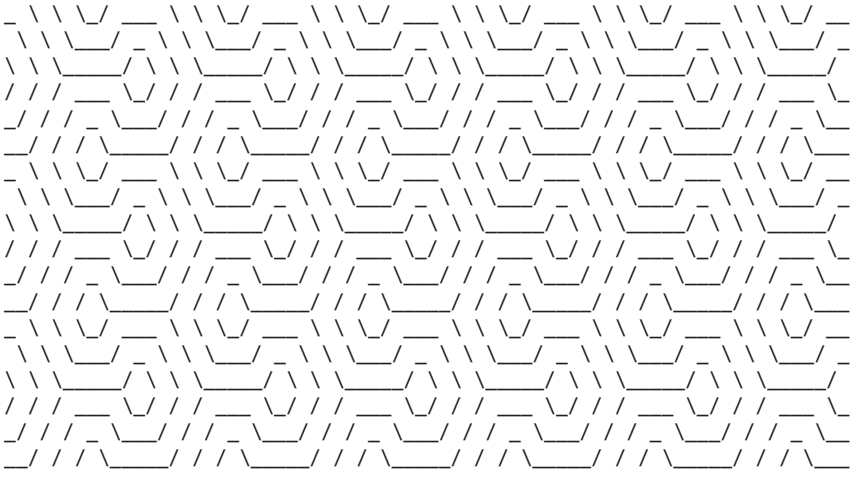
Ковер из «Сияния»

Действие «Сияния», психологического фильма ужасов 1980 года режиссера Стэнли Кубрика, происходит в населенном призраками отеле «Оверлук». Шестиугольный узор ковра данного отеля — неотъемлемая часть этого знаменитого фильма. На ковре изображены перемежающиеся и переплетенные шестиугольники, гипнотический эффект которых идеально подходит для такого нервного фильма. Короткая программа из этого проекта, аналогичная программе проекта 35, выводит этот монотонный узор на экран.

Отмечу, что в этой программе используются неформатированные строки, в которых открывающая кавычка предваряется буквой `r` в нижнем регистре, поэтому обратные косые черты не интерпретируются как символы экранирования.

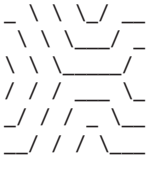
Программа в действии

Результат выполнения `shiningcarpet.py` выглядит следующим образом:



Описание работы

Создание подобной программы (или аналогичного ей проекта 35) начинается не с написания кода, а с рисования мозаичных изображений в текстовом редакторе. А когда узор будет нарисован, можно будет разбить его на мельчайшие элементы мозаики:



После копирования и вставки этого текста в исходный код можно будет написать и остальную часть программы. Для создания программного обеспечения недостаточно просто сесть и написать код от начала до конца. Любой профессиональный разработчик ПО проходит через несколько итераций исправления огрехов, экспериментов и отладки. Конечный результат может состоять всего из девяти строк кода, но маленькая программа вовсе не означает, что в ее создание было вложено мало усилий.

```
1. """Ковер из "Сияния", (c) Эл Свейгарт al@inventwithpython.com
2. Выводит на экран мозаичный узор ковра из фильма "Сияние"
3. Код размещен на https://nostarch.com/big-book-small-python-projects
4. Теги: крошечная, для начинающих, графика"""
5.
6. # Задаем константы:
7. X_REPEAT = 6  # Количество ячеек по горизонтали.
```

```
 8. Y_REPEAT = 4  # Количество ячеек по вертикали.
 9.
10. for i in range(Y_REPEAT):
11.     print(r'_ \ \ \_/ __' * X_REPEAT)
12.     print(r' \ \ \___/ _' * X_REPEAT)
13.     print(r'\ \ \_____/ ' * X_REPEAT)
14.     print(r'/ / / ___ \_' * X_REPEAT)
15.     print(r'_/ / / _ \__' * X_REPEAT)
16.     print(r'__/ / / \___' * X_REPEAT)
```

Исследование программы

Чтобы попрактиковаться, попробуйте создать следующие узоры:

```
___|___|___|___|___|___|___|___|___|___|___|___|___|___|
_|___|___|___|___|___|___|___|___|___|___|___|___|___|__
___|___|___|___|___|___|___|___|___|___|___|___|___|___|
_|___|___|___|___|___|___|___|___|___|___|___|___|___|__
___|___|___|___|___|___|___|___|___|___|___|___|___|___|
_|___|___|___|___|___|___|___|___|___|___|___|___|___|__

((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )
 ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(
((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )
 ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(
((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )((  )
 ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(  ))(

 / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/
/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____
\ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __
 \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \
 / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/
/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____
\ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __ \ \__/ / __
 \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \ \____/ /  \

  \__   \__   \__   \__   \__   \__   \__   \__   \__   \__
__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \
  \     \     \     \     \     \     \     \     \     \
__/   __/   __/   __/   __/   __/   __/   __/   __/   __/
  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/
__/     /     /     /     /     /     /     /     /     /
  \__   \__   \__   \__   \__   \__   \__   \__   \__   \__
__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \__/  \

/ ___ \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^
 /   \ vvv /   \ vvv /   \ vvv /   \ vvv /   \ vvv /   \ vvv
|() ()|   |() ()|   |() ()|   |() ()|   |() ()|   |() ()|
 \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^ / ___ \ ^ / ___
\ vvv /   \ vvv /   \ vvv /   \ vvv /   \ vvv /   \ vvv /
)|   |() ()|   |() ()|   |() ()|   |() ()|   |() ()|   |() (
```

66

Простой шифр подстановки

Наш простой шифр подстановки заменяет одни буквы другими. Поскольку существует 26 возможных подстановок для буквы A, 25 возможных подстановок для буквы B, 24 — для C и т. д., общее количество возможных ключей составляет 26 × 25 × 24 × 23 × ... × 1, то есть 403 291 461 126 605 635 584 000 000 ключей! Слишком много, чтобы их можно было перебрать прямым перебором даже на суперкомпьютере, поэтому метод взлома, представленный в проекте 7, для этого простого шифра не подойдет. К сожалению, коварные злоумышленники могут воспользоваться известными слабыми местами шифра для его взлома. Если вам хотелось бы узнать больше о шифрах и методах их взлома — можете прочитать мою книгу *Cracking Codes with Python*.

Программа в действии

Результат выполнения `simplesubcipher.py` выглядит следующим образом:

```
Simple Substitution Cipher, by Al Sweigart
A simple substitution cipher has a one-to-one translation for each
symbol in the plaintext and each symbol in the ciphertext.
Do you want to (e)ncrypt or (d)ecrypt?
> e
Please specify the key to use.
Or enter RANDOM to have one generated for you.
> random
```

```
The key is WNOMTRCEHDXBFVSLKAGZIPYJQU. KEEP THIS SECRET!
Enter the message to encrypt.
> Meet me by the rose bushes tonight.
The encrypted message is:
Fttz ft nq zet asgt nigetg zsvhcez.
Full encrypted text copied to clipboard.

Simple Substitution Cipher, by Al Sweigart
A simple substitution cipher has a one-to-one translation for each
symbol in the plaintext and each symbol in the ciphertext.
Do you want to (e)ncrypt or (d)ecrypt?
> d
Please specify the key to use.
> WNOMTRCEHDXBFVSLKAGZIPYJQU
Enter the message to decrypt.
> Fttz ft nq zet asgt nigetg zsvhcez.
The decrypted message is:
Meet me by the rose bushes tonight.
Full decrypted text copied to clipboard.
```

Описание работы

Позиция каждой из 26 букв ключа соответствует букве алфавита на той же позиции (рис. 66.1).

```
A B C D E F G H I  J K L M N O P Q R S T U V W X Y Z
↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕ ↕
W N O M T R C E H D X B F V S L K A G Z I P Y J Q U
```

Рис. 66.1. Шифрование букв алфавита с помощью ключа, начинающегося с WNOM. Для расшифровки замените буквы внизу соответствующими буквами вверху

При использовании этого ключа буква A шифруется буквой W (а расшифрованная W превращается в A), B — N и т. д. Переменные `LETTERS` и `key` присваиваются переменным `charsA` и `charsB` (или наоборот при расшифровке). Все символы сообщения в `charsA` заменяются соответствующими символами из `charsB` для получения итогового преобразованного сообщения.

```
1. """Простой шифр подстановки, (c) Эл Свейгарт al@inventwithpython.com
2. Простой шифр подстановки с взаимнооднозначным преобразованием всех
3. символов открытого и зашифрованного текста.
4. Подробнее — в статье на https://ru.wikipedia.org/wiki/Шифр_подстановки
5. Код размещен на https://nostarch.com/big-book-small-python-projects
6. Теги: короткая, криптография, математическая"""
7.
8. import random
9.
```

```
try:
    import pyperclip  # pyperclip копирует текст в буфер обмена.
except ImportError:
    pass  # Если pyperclip не установлена, ничего не делаем. Не проблема.

# Все возможные символы для шифрования/дешифровки:
LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'

def main():
    print('''Simple Substitution Cipher, by Al Sweigart
A simple substitution cipher has a one-to-one translation for each
symbol in the plaintext and each symbol in the ciphertext.''')

    # Спрашиваем у пользователя, хочет ли он шифровать или расшифровывать:
    while True:  # Повторяем вопрос, пока пользователь не введет e или d.
        print('Do you want to (e)ncrypt or (d)ecrypt?')
        response = input('> ').lower()
        if response.startswith('e'):
            myMode = 'encrypt'
            break
        elif response.startswith('d'):
            myMode = 'decrypt'
            break
        print('Please enter the letter e or d.')

    # Просим пользователя ввести ключ шифрования:
    while True: # Повторяем вопрос, пока пользователь не введет допустимый ключ.
        print('Please specify the key to use.')
        if myMode == 'encrypt':
            print('Or enter RANDOM to have one generated for you.')
        response = input('> ').upper()
        if response == 'RANDOM':
            myKey = generateRandomKey()
            print('The key is {}. KEEP THIS SECRET!'.format(myKey))
            break
        else:
            if checkKey(response):
                myKey = response
                break

    # Просим пользователя ввести сообщение для шифрования/расшифровки:
    print('Enter the message to {}.'.format(myMode))
    myMessage = input('> ')

    # Производим шифрование/расшифровку:
    if myMode == 'encrypt':
        translated = encryptMessage(myMessage, myKey)
    elif myMode == 'decrypt':
        translated = decryptMessage(myMessage, myKey)

    # Отображаем результат:
```

```
61.     print('The %sed message is:' % (myMode))
62.     print(translated)
63.
64.     try:
65.         pyperclip.copy(translated)
66.         print('Full %sed text copied to clipboard.' % (myMode))
67.     except:
68.         pass  # Если pyperclip не установлена, ничего не делаем.
69.
70. def checkKey(key):
71.     """Возвращает True, если ключ допустимый. В противном случае
72.     возвращает False."""
73.     keyList = list(key)
74.     lettersList = list(LETTERS)
75.     keyList.sort()
76.     lettersList.sort()
77.     if keyList != lettersList:
78.         print('There is an error in the key or symbol set.')
79.         return False
80.     return True
81.
82.
83. def encryptMessage(message, key):
84.     """Шифрует сообщение в соответствии с ключом key."""
85.     return translateMessage(message, key, 'encrypt')
86.
87.
88. def decryptMessage(message, key):
89.     """Расшифровывает сообщение в соответствии с ключом key."""
90.     return translateMessage(message, key, 'decrypt')
91.
92. def translateMessage(message, key, mode):
93.     """Зашифровывает или расшифровывает сообщение в соответствии
94.     с ключом key."""
95.     translated = ''
96.     charsA = LETTERS
97.     charsB = key
98.     if mode == 'decrypt':
99.         # Для расшифровки можно использовать тот же код, что и для
100.        # шифрования. Нужно просто поменять местами key и LETTERS.
101.        charsA, charsB = charsB, charsA
102.
103.    # Проходим в цикле по всем символам сообщения:
104.    for symbol in message:
105.        if symbol.upper() in charsA:
106.            # Зашифровываем/расшифровываем символ:
107.            symIndex = charsA.find(symbol.upper())
108.            if symbol.isupper():
109.                translated += charsB[symIndex].upper()
110.            else:
111.                translated += charsB[symIndex].lower()
```

```
112.         else:
113.             # Если символ не входит в LETTERS, добавляем его без изменения.
114.             translated += symbol
115.
116.     return translated
117.
118.
119. def generateRandomKey():
120.     """Генерирует и возвращает случайный ключ шифрования."""
121.     key = list(LETTERS)  # Преобразуем строковое значение LETTERS в список.
122.     random.shuffle(key)  # Перетасовываем этот список случайным образом.
123.     return ''.join(key)  # Преобразуем список в строковое значение.
124.
125.
126. # Если программа не импортируется, а запускается, производим запуск:
127. if __name__ == '__main__':
128.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `random.shuffle(key)` в строке 122 и ввести в качестве ключа `RANDOM`?
2. Что будет, если расширить строковое значение `LETTERS` в строке 16 до `'ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890'`?

67

Синусовидное сообщение

Данная программа отображает на экране выбранное пользователем сообщение синусовидной волной по мере прокрутки текста. Это возможно благодаря функции `math.sin()`, реализующей тригонометрическую волновую функцию синус. Но даже если математическая сторона вам непонятна, программа довольно коротка и скопировать ее легко.

Программа в действии

Результат выполнения `sinemessage.py` выглядит следующим образом:

```
Sine Message, by Al Sweigart al@inventwithpython.com
(Press Ctrl-C to quit.)

What message do you want to display? (Max 39 chars.)
> I <3 Programming!
                                  I <3 Programming!
                                          I <3 Programming!
                                                  I <3 Programming!
                                                          I <3 Programming!
                                                                  I <3 Programming!
                                                                   I <3 Programming!
                                                                   I <3 Programming!
                                                                  I <3 Programming!
                                                              I <3 Programming!
                                                          I <3 Programming!
                                                      I <3 Programming!
                                                  I <3 Programming!
                                              I <3 Programming!
                                          I <3 Programming!
                                      I <3 Programming!
                                  I <3 Programming!
                              I <3 Programming!
                          I <3 Programming!
                      I <3 Programming!
                  I <3 Programming!
              I <3 Programming!
          I <3 Programming!
      I <3 Programming!
   I <3 Programming!
I <3 Programming!
I <3 Programming!
 I <3 Programming!
    I <3 Programming!
         I <3 Programming!
--сокращено--
```

Описание работы

Функция `math.sin()` из модуля `math` языка Python принимает аргумент, который мы назовем *x*, и возвращает другое число, называемое *синусом x*. Синус используется во множестве математических приложений; в нашей программе он служит просто для создания красивого эффекта волны. В функцию `math.sin()` мы передаем переменную `step`, значение которой начинается с `0` и увеличивается на `0.25` на каждой итерации основного цикла программы.

На основе значения, возвращаемого функцией `math.sin()`, мы будем вычислять, сколько пробелов в качестве полей необходимо вывести с каждой из сторон пользовательского сообщения. Поскольку `math.sin()` возвращает число с плавающей точкой в диапазоне от `-1.0` до `1.0`, а минимальный размер полей должен составлять 0 пробелов, а не отрицательное значение, в строке 31 мы прибавляем

к возвращаемому функцией `math.sin()` значению 1, и диапазон сдвигается до от `0` до `2.0`. Конечно, нам нужно больше пробелов, чем от 0 до 2, так что в строке 31 мы умножаем это значение на переменную `multiplier`, чтобы увеличить поля. Полученное произведение и равно количеству пробелов, которые необходимо добавить слева перед выводом пользовательского сообщения.

В результате и получается волнообразное динамическое изображение, видимое при запуске нашей программы.

```
 1. """Синусовидное сообщение, (c) Эл Свейгарт al@inventwithpython.com
 2. Выводит сообщение синусовидной волной.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, графика"""
 5.
 6. import math, shutil, sys, time
 7.
 8. # Получаем размер окна терминала:
 9. WIDTH, HEIGHT = shutil.get_terminal_size()
10. # В Windows нельзя вывести что-либо в последнем столбце без добавления
11. # автоматически символа новой строки, поэтому уменьшаем ширину на 1:
12. WIDTH -= 1
13.
14. print('Sine Message, by Al Sweigart al@inventwithpython.com')
15. print('(Press Ctrl-C to quit.)')
16. print()
17. print('What message do you want to display? (Max', WIDTH // 2, 'chars.)')
18. while True:
19.     message = input('> ')
20.     if 1 <= len(message) <= (WIDTH // 2):
21.         break
22.     print('Message must be 1 to', WIDTH // 2, 'characters long.')
23.
24. step = 0.0  # step определяет, в каком месте синусоиды мы находимся.
25. # Синус принимает значения от -1.0 до 1.0, так что необходимо умножить
26. # на коэффициент:
27. multiplier = (WIDTH - len(message)) / 2
28. try:
29.     while True:  # Основной цикл программы.
30.         sinOfStep = math.sin(step)
31.         padding = ' ' * int((sinOfStep + 1) * multiplier)
32.         print(padding + message)
33.         time.sleep(0.1)
34.         step += 0.25  # (!) Попробуйте заменить это значение на 0.1 или 0.5.
35. except KeyboardInterrupt:
36.     sys.exit()  # Если нажато сочетание клавиш Ctrl+C — завершаем программу.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных `(!)`.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `math.sin(step)` в строке 30 заменить на `math.cos(step)`?
2. Что будет, если `math.sin(step)` в строке 30 заменить на `math.sin(0)`?

68

Игра в 15

Размер доски в этой классической головоломке — 4 × 4, с 15 пронумерованными костяшками и одной пустой клеткой. Цель игры — перемещая костяшки по доске, упорядочить их, считая слева направо и сверху вниз. Костяшки можно только двигать по доске, поднимать их и непосредственно перемещать нельзя. Некоторые версии данной головоломки содержат зашифрованные изображения, проявляющиеся после ее решения.

Подробнее об игре можно прочитать в статье «Википедии»: https://ru.wikipedia.org/wiki/Игра_в_15.

Программа в действии

Результат выполнения `slidingtilepuzzle.py` выглядит следующим образом:

```
Sliding Tile Puzzle, by Al Sweigart al@inventwithpython.com

    Use the WASD keys to move the tiles
    back into their original order:
           1  2  3  4
           5  6  7  8
           9 10 11 12
          13 14 15
Press Enter to begin...
```

```
+------+------+------+------+
|      |      |      |      |
|  5   |  10  |      |  11  |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  6   |  3   |  7   |  2   |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  14  |  1   |  15  |  8   |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  9   |  13  |  4   |  12  |
|      |      |      |      |
+------+------+------+------+

                          (W)
Enter WASD (or QUIT): (A) ( ) (D)
> w

+------+------+------+------+
|      |      |      |      |
|  5   |  10  |  7   |  11  |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  6   |  3   |      |  2   |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  14  |  1   |  15  |  8   |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  9   |  13  |  4   |  12  |
|      |      |      |      |
+------+------+------+------+

                          (W)
Enter WASD (or QUIT): (A) (S) (D)
--сокращено--
```

Описание работы

Структура данных, отражающая игральную доску, представляет собой список списков. Внутренние списки соответствуют отдельным столбцам доски размером 4 × 4 и содержат строковые значения для пронумерованных костяшек (или строковое

значение BLANK для пустой клетки). Функция getNewBoard() возвращает этот список списков со всеми костяшками в начальных позициях и пустой клеткой в правом нижнем углу.

Python позволяет менять значения в двух переменных местами вот так: a, b = b, a. Наша программа с помощью этого приема меняет местами в строках с 101-й по 108-ю пустую клетку и соседнюю костяшку, имитируя передвижение пронумерованной костяшки на незанятое место. Функция getNewPuzzle() генерирует новую игру, производя случайным образом 200 подобных перестановок.

```
  1. """Игра в 15, (c) Эл Свейгарт al@inventwithpython.com
  2. Расставьте пронумерованные костяшки в правильном порядке
  3. Код размещен на https://nostarch.com/big-book-small-python-projects
  4. Теги: большая, игра, головоломка"""
  5.
  6. import random, sys
  7.
  8. BLANK = '  '  # Обратите внимание: это строковое значение состоит
  9.               # из двух пробелов.
 10.
 11. def main():
 12.     print('''Sliding Tile Puzzle, by Al Sweigart al@inventwithpython.com
 13.
 14.     Use the WASD keys to move the tiles
 15.     back into their original order:
 16.            1  2  3  4
 17.            5  6  7  8
 18.            9 10 11 12
 19.           13 14 15   ''')
 20.     input('Press Enter to begin...')
 21.
 22.     gameBoard = getNewPuzzle()
 23.
 24.     while True:
 25.         displayBoard(gameBoard)
 26.         playerMove = askForPlayerMove(gameBoard)
 27.         makeMove(gameBoard, playerMove)
 28.
 29.         if gameBoard == getNewBoard():
 30.             print('You won!')
 31.             sys.exit()
 32.
 33.
 34. def getNewBoard():
 35.     """Возвращает список списков, соответствующий новой игре."""
 36.     return [['1 ', '5 ', '9 ', '13'], ['2 ', '6 ', '10', '14'],
 37.             ['3 ', '7 ', '11', '15'], ['4 ', '8 ', '12', BLANK]]
 38.
 39.
```

```
def displayBoard(board):
    """Отображает заданную доску на экране."""
    labels = [board[0][0], board[1][0], board[2][0], board[3][0],
              board[0][1], board[1][1], board[2][1], board[3][1],
              board[0][2], board[1][2], board[2][2], board[3][2],
              board[0][3], board[1][3], board[2][3], board[3][3]]
    boardToDraw = """
+------+------+------+------+
|      |      |      |      |
|  {}  |  {}  |  {}  |  {}  |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  {}  |  {}  |  {}  |  {}  |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  {}  |  {}  |  {}  |  {}  |
|      |      |      |      |
+------+------+------+------+
|      |      |      |      |
|  {}  |  {}  |  {}  |  {}  |
|      |      |      |      |
+------+------+------+------+
""".format(*labels)
    print(boardToDraw)


def findBlankSpace(board):
    """Возвращает кортеж (x, y) с местоположением пустой клетки."""
    for x in range(4):
        for y in range(4):
            if board[x][y] == '  ':
                return (x, y)


def askForPlayerMove(board):
    """Запрашивает у игрока, какую костяшку передвигать."""
    blankx, blanky = findBlankSpace(board)

    w = 'W' if blanky != 3 else ' '
    a = 'A' if blankx != 3 else ' '
    s = 'S' if blanky != 0 else ' '
    d = 'D' if blankx != 0 else ' '

    while True:
        print('                          ({})'.format(w))
        print('Enter WASD (or QUIT): ({}) ({}) ({})'.format(a, s, d))

        response = input('> ').upper()
```

```
 90.         if response == 'QUIT':
 91.             sys.exit()
 92.         if response in (w + a + s + d).replace(' ', ''):
 93.             return response
 94.
 95.
 96. def makeMove(board, move):
 97.     """Производит заданный ход move на заданной доске board."""
 98.     # Примечание: эта функция предполагает, что ход допустим.
 99.     bx, by = findBlankSpace(board)
100.
101.     if move == 'W':
102.         board[bx][by], board[bx][by+1] = board[bx][by+1], board[bx][by]
103.     elif move == 'A':
104.         board[bx][by], board[bx+1][by] = board[bx+1][by], board[bx][by]
105.     elif move == 'S':
106.         board[bx][by], board[bx][by-1] = board[bx][by-1], board[bx][by]
107.     elif move == 'D':
108.         board[bx][by], board[bx-1][by] = board[bx-1][by], board[bx][by]
109.
110.
111. def makeRandomMove(board):
112.     """Передвигает костяшку случайным образом."""
113.     blankx, blanky = findBlankSpace(board)
114.     validMoves = []
115.     if blanky != 3:
116.         validMoves.append('W')
117.     if blankx != 3:
118.         validMoves.append('A')
119.     if blanky != 0:
120.         validMoves.append('S')
121.     if blankx != 0:
122.         validMoves.append('D')
123.
124.     makeMove(board, random.choice(validMoves))
125.
126. def getNewPuzzle(moves=200):
127.     """Генерируем новую игру с помощью случайных ходов из упорядоченного
128.     состояния."""
129.     board = getNewBoard()
130.
131.     for i in range(moves):
132.         makeRandomMove(board)
133.     return board
134.
135.
136. # Если программа не импортируется, а запускается, выполняем запуск:
137. if __name__ == '__main__':
138.     main()
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- создать более сложный вариант этой головоломки, на доске 5 × 5;
- создать режим «автоматического решения головоломки», в котором сохраняется текущее расположение костяшек, после чего программа выполняет до 40 случайных ходов и прекращает работу, если головоломка решена. В противном случае программа загружает сохраненное состояние и делает еще 40 случайных ходов.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `getNewPuzzle()` в строке 22 заменить на `getNewPuzzle(1)`?
2. Что будет, если `getNewPuzzle()` в строке 22 заменить на `getNewPuzzle(0)`?
3. Что будет, если удалить или закомментировать `sys.exit()` в строке 31?

69

Бега улиток

Вряд ли вы получите много адреналина от бегущих... улиток. Но нехватку скорости они с лихвой компенсируют очаровательностью ASCII-графики. Все улитки (изображаемые с помощью символа @ в качестве раковины и v в качестве двух глазных стебельков) медленно, но верно движутся к финишной прямой. В гонках может участвовать до восьми улиток, оставляющих позади след слизи, каждая со своим, выбранным пользователем именем. Эта программа хорошо подходит для начинающих.

Программа в действии

Результат выполнения `snailrace.py` выглядит следующим образом:

```
Snail Race, by Al Sweigart al@inventwithpython.com

    @v <-- snail

How many snails will race? Max: 8
> 3
Enter snail #1's name:
> Alice
Enter snail #2's name:
> Bob
Enter snail #3's name:
> Carol
```

```
START                                   FINISH
|                                       |
      Alice
......@v
     Bob
.....@v
       Carol
.......@v
--сокращено--
```

Описание работы

В этой программе используются две структуры данных, хранящихся в двух переменных: `snailNames` представляет собой список строковых значений для имен улиток, а `snailProgress` — ассоциативный массив, ключами которого служат имена улиток, а значения представляют собой целые числа, показывающие, сколько знако-мест проползла каждая улитка. В строках с 79-й по 82-ю на основе данных из этих переменных все улитки отрисовываются в соответствующих местах экрана.

```
 1. """Бега улиток, (c) Эл Свейгарт al@inventwithpython.com
 2. Бега быстроногих улиток!
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, графика, для начинающих, игра, многопользовательская"""
 5.
 6. import random, time, sys
 7.
 8. # Задаем константы:
 9. MAX_NUM_SNAILS = 8
10. MAX_NAME_LENGTH = 20
11. FINISH_LINE = 40  # (!) Попробуйте изменить это число.
12.
13. print('''Snail Race, by Al Sweigart al@inventwithpython.com
14.
15.     @v <-- snail
16.
17. ''')
18.
19. # Спрашиваем, сколько улиток должно участвовать в бегах:
20. while True:  # Спрашиваем снова, пока игрок не введет число.
21.     print('How many snails will race? Max:', MAX_NUM_SNAILS)
22.     response = input('> ')
23.     if response.isdecimal():
24.         numSnailsRacing = int(response)
25.         if 1 < numSnailsRacing <= MAX_NUM_SNAILS:
26.             break
27.     print('Enter a number between 2 and', MAX_NUM_SNAILS)
28.
29. # Ввод имен всех улиток:
30. snailNames = []  # Список имен улиток в виде строковых значений.
```

```
for i in range(1, numSnailsRacing + 1):
    while True:  # Продолжаем спрашивать, пока игрок не введет допустимое имя.
        print('Enter snail #' + str(i) + "'s name:")
        name = input('> ')
        if len(name) == 0:
            print('Please enter a name.')
        elif name in snailNames:
            print('Choose a name that has not already been used.')
        else:
            break  # Введено приемлемое имя.
    snailNames.append(name)

# Отображаем всех улиток на старте.
print('\n' * 40)
print('START' + (' ' * (FINISH_LINE - len('START')) + 'FINISH'))
print('|' + (' ' * (FINISH_LINE - len('|')) + '|'))
snailProgress = {}
for snailName in snailNames:
    print(snailName[:MAX_NAME_LENGTH])
    print('@v')
    snailProgress[snailName] = 0

time.sleep(1.5)  # Пауза перед началом гонок.

while True:  # Основной цикл программы.
    # Выбираем случайным образом, каких улиток двигать вперед:
    for i in range(random.randint(1, numSnailsRacing // 2)):
        randomSnailName = random.choice(snailNames)
        snailProgress[randomSnailName] += 1

        # Проверяем, не достигла ли улитка финишной прямой:
        if snailProgress[randomSnailName] == FINISH_LINE:
            print(randomSnailName, 'has won!')
            sys.exit()

    # (!) Эксперимент: добавьте небольшой жульнический трюк:
    # увеличение скорости улитки, которую зовут так же, как вас.

    time.sleep(0.5)  # (!) Эксперимент: попробуйте изменить это значение.

    # (!) Эксперимент: что будет, если закомментировать эту строку?
    print('\n' * 40)

    # Отображает стартовую и финишную прямые:
    print('START' + (' ' * (FINISH_LINE - len('START')) + 'FINISH'))
    print('|' + (' ' * (FINISH_LINE - 1) + '|'))

    # Отображает улиток (с метками имен):
    for snailName in snailNames:
        spaces = snailProgress[snailName]
        print((' ' * spaces) + snailName[:MAX_NAME_LENGTH])
        print(('.' * snailProgress[snailName]) + '@v')
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (`!`). Можете также сами попробовать придумать, как сделать следующее:

- добавить случайный режим резкого повышения скорости, при котором улитка прыгает сразу на четыре знако-места вместо одного;
- добавить «спящий режим», в который улитки могут входить случайным образом во время гонки;
- добавить поддержку ничьих на случай, если улитки достигнут финишной прямой одновременно.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `snailName[:MAX_NAME_LENGTH]` в строке 81 заменить на `snailNames[0]`?
2. Что будет, если `print('@v')` в строке 50 заменить на `print('v@')`?

70

Соробан — японский абак

Абак (счетная доска) представляет собой счетный инструмент, встречавшийся во множестве культур задолго до изобретения калькуляторов. На рис. 70.1 показана японская разновидность абака — соробан. Каждая проволока соответствует разряду в позиционной системе счисления, а костяшки на проволоке — цифрам в этом разряде. Например, соробан, на котором отодвинуты две костяшки на крайней справа проволоке и три костяшки на предыдущей перед ней, отражает число 32. Наша программа моделирует соробан (нельзя не отметить иронию моделирования доисторического счетного инструмента с помощью компьютера).

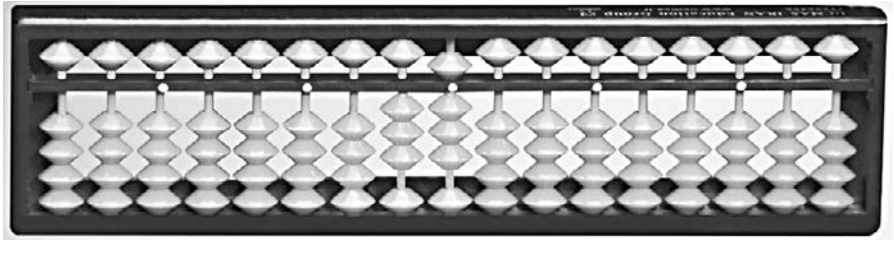


Рис. 70.1. Соробан

Каждому столбцу в соробане соответствует своя цифра. Крайний справа столбец соответствует единицам, столбец слева от него — десяткам, следующий — сотням и т. д. Клавиши Q, W, E, R, T, Y, U, I, O и P вверху клавиатуры позволяют увеличивать

значение в соответствующем разряде, а A, S, D, F, G, H, J, K, L и ; — уменьшать его. Отражая текущее число, костяшки на нашем виртуальном соробане будут сдвигаться. Можно также вводить числа напрямую.

Четыре костяшки под горизонтальным разделителем считаются костяшками «земли», передвижение их наверх к разделителю считается за 1 в этом разряде. Костяшка над разделителем считается костяшкой «неба», и передвижение ее вниз к разделителю считается за 5, так что передвинутые вниз одна костяшка «неба» и вверх три костяшки «земли» в столбце десятков означают число 80. Больше информации об абаках и их использовании можно найти в статье «Википедии»: https://ru.wikipedia.org/wiki/Абак.

Программа в действии

Результат выполнения `soroban.py` выглядит следующим образом:

```
Soroban - The Japanese Abacus
By Al Sweigart al@inventwithpython.com

+================================+
I  O  O  O  O  O  O  O  O  O  O  I
I  |  |  |  |  |  |  |  |  |  |  I
I  |  |  |  |  |  |  |  |  |  |  I
+================================+
I  |  |  |  |  |  |  |  |  |  |  I
I  |  |  |  |  |  |  |  |  |  |  I
I  O  O  O  O  O  O  O  O  O  O  I
I  O  O  O  O  O  O  O  O  O  O  I
I  O  O  O  O  O  O  O  O  O  O  I
I  O  O  O  O  O  O  O  O  O  O  I
+==0==0==0==0==0==0==0==0==0==0==+
  +q  w  e  r  t  y  u  i  o  p
  -a  s  d  f  g  h  j  k  l  ;
(Enter a number, "quit", or a stream of up/down letters.)
> pppiiiii
+================================+
I  O  O  O  O  O  O  O  |  O  O  I
I  |  |  |  |  |  |  |  |  |  |  I
I  |  |  |  |  |  |  |  O  |  |  I
+================================+
I  |  |  |  |  |  |  |  |  |  O  I
I  |  |  |  |  |  |  |  |  |  O  I
I  O  O  O  O  O  O  O  O  O  O  I
I  O  O  O  O  O  O  O  O  O  |  I
I  O  O  O  O  O  O  O  O  O  |  I
I  O  O  O  O  O  O  O  O  O  O  I
+==0==0==0==0==0==0==0==5==0==3==+
  +q  w  e  r  t  y  u  i  o  p
```

```
  -a  s  d  f  g  h  j  k  l  ;
(Enter a number, "quit", or a stream of up/down letters.)
--сокращено--
```

Описание работы

Функция `displayAbacus()` принимает аргумент `number`, на основе которого и визуализируются костяшки на абаке. На соробане всегда ровно 80 возможных позиций для костяшек `'0'` и сегментов прутьев `'|'`, как видно из фигурных скобок (`{}`) в многострочном строковом значении в строках кода с 127-й по 139-ю. Еще десять пар фигурных скобок соответствуют цифрам в аргументе `number`.

Необходимо создать список строковых значений, чтобы заполнить эти фигурные скобки, в порядке слева направо, сверху вниз. Код из функции `displayAbacus()` заполняет список `hasBead` значениями `True` для отображения `'0'` и `False` для отображения `'|'`. Первые десять значений в списке — верхний ряд «небес». Мы будем ставить костяшку в этот ряд, только если цифра в столбце — 0, 1, 2, 3 или 4, поскольку костяшка «неба» не окажется в этом ряду, если цифра в столбце — от 0 до 4. Для оставшихся рядов мы добавим булевы значения в `hasBead`.

В строках с 118-й по 123-ю `hasBead` используется для создания списка `abacusChar`, содержащего сами строковые значения `'0'` и `'|'`. В сочетании с `numberList` в строке 126 программа формирует список `chars`, заполняющий фигурные скобки (`{}`) в многострочном строковом значении ASCII-графики соробана.

```
  1. """Японский абак – соробан, (c) Эл Свейгарт al@inventwithpython.com
  2. Моделирование японского счетного инструмента типа абака.
  3. Больше информации – в статье https://ru.wikipedia.org/wiki/Соробан
  4. Код размещен на https://nostarch.com/big-book-small-python-projects
  5. Теги: большая, графика, математическая, имитационное моделирование"""
  6.
  7. NUMBER_OF_DIGITS = 10
  8.
  9.
 10. def main():
 11.     print('Soroban - The Japanese Abacus')
 12.     print('By Al Sweigart al@inventwithpython.com')
 13.     print()
 14.
 15.     abacusNumber = 0  # Число, отображаемое на абаке.
 16.
 17.     while True:  # Основной цикл программы.
 18.         displayAbacus(abacusNumber)
 19.         displayControls()
 20.
 21.         commands = input('> ')
 22.         if commands == 'quit':
```

```
            # Выходим из программы:
            break
        elif commands.isdecimal():
            # Задаем число на абаке:
            abacusNumber = int(commands)
        else:
            # Обрабатываем команды увеличения/уменьшения:
            for letter in commands:
                if letter == 'q':
                    abacusNumber += 1000000000
                elif letter == 'a':
                    abacusNumber -= 1000000000
                elif letter == 'w':
                    abacusNumber += 100000000
                elif letter == 's':
                    abacusNumber -= 100000000
                elif letter == 'e':
                    abacusNumber += 10000000
                elif letter == 'd':
                    abacusNumber -= 10000000
                elif letter == 'r':
                    abacusNumber += 1000000
                elif letter == 'f':
                    abacusNumber -= 1000000
                elif letter == 't':
                    abacusNumber += 100000
                elif letter == 'g':
                    abacusNumber -= 100000
                elif letter == 'y':
                    abacusNumber += 10000
                elif letter == 'h':
                    abacusNumber -= 10000
                elif letter == 'u':
                    abacusNumber += 1000
                elif letter == 'j':
                    abacusNumber -= 1000
                elif letter == 'i':
                    abacusNumber += 100
                elif letter == 'k':
                    abacusNumber -= 100
                elif letter == 'o':
                    abacusNumber += 10
                elif letter == 'l':
                    abacusNumber -= 10
                elif letter == 'p':
                    abacusNumber += 1
                elif letter == ';':
                    abacusNumber -= 1

        # Абак не может отображать отрицательные числа:
        if abacusNumber < 0:
```

```
74.             abacusNumber = 0  # Заменяем все отрицательные числа на 0.
75.         # Абак не может отображать числа, превышающие 9 999 999 999:
76.         if abacusNumber > 9999999999:
77.             abacusNumber = 9999999999
78.
79.
80. def displayAbacus(number):
81.     numberList = list(str(number).zfill(NUMBER_OF_DIGITS))
82.
83.     hasBead = []  # Содержит для каждой позиции костяшек True или False.
84.
85.     # Костяшка в верхнем ряду "неба" соответствует цифрам 0, 1, 2, 3 или 4.
86.     for i in range(NUMBER_OF_DIGITS):
87.         hasBead.append(numberList[i] in '01234')
88.
89.     # Костяшка в верхнем ряду "неба" соответствует цифрам 5, 6, 7, 8 или 9.
90.     for i in range(NUMBER_OF_DIGITS):
91.         hasBead.append(numberList[i] in '56789')
92.
93.     # Костяшка в первом (верхнем) ряду "земли" выражает все цифры, кроме 0.
94.     for i in range(NUMBER_OF_DIGITS):
95.         hasBead.append(numberList[i] in '12346789')
96.
97.     # Костяшка в втором ряду "земли" выражает цифры 2, 3, 4, 7, 8 или 9.
98.     for i in range(NUMBER_OF_DIGITS):
99.         hasBead.append(numberList[i] in '234789')
100.
101.    # Костяшка в третьем ряду "земли" выражает цифры 0, 3, 4, 5, 8 или 9.
102.    for i in range(NUMBER_OF_DIGITS):
103.        hasBead.append(numberList[i] in '034589')
104.
105.    # Костяшка в четвертом ряду "земли" выражает цифры 0, 1, 2, 4, 5, 6 или 9.
106.    for i in range(NUMBER_OF_DIGITS):
107.        hasBead.append(numberList[i] in '014569')
108.
109.    # Костяшка в пятом ряду "земли" выражает цифры 0, 1, 2, 5, 6 или 7.
110.    for i in range(NUMBER_OF_DIGITS):
111.        hasBead.append(numberList[i] in '012567')
112.
113.    # Костяшка в шестом ряду "земли" выражает цифры 0, 1, 2, 3, 5, 6, 7 или  8.
114.    for i in range(NUMBER_OF_DIGITS):
115.        hasBead.append(numberList[i] in '01235678')
116.
117.    # Преобразуем эти значения True/False в символы O или |.
118.    abacusChar = []
119.    for i, beadPresent in enumerate(hasBead):
120.        if beadPresent:
121.            abacusChar.append('O')
122.        else:
123.            abacusChar.append('|')
124.
```

```
125.     # Рисуем абак с символами O/|.
126.     chars = abacusChar + numberList
127.     print("""
128. +================================+
129. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
130. I  |  |  |  |  |  |  |  |  |  |  I
131. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
132. +================================+
133. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
134. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
135. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
136. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
137. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
138. I  {}  {}  {}  {}  {}  {}  {}  {}  {}  {}  I
139. +==={}==={}==={}==={}==={}==={}==={}==={}==+""".format(*chars))
140.
141.
142. def displayControls():
143.     print('  +q  w  e  r  t  y  u  i  o  p')
144.     print('  -a  s  d  f  g  h  j  k  l  ;')
145.     print('(Enter a number, "quit", or a stream of up/down letters.)')
146.
147.
148. if __name__ == '__main__':
149.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `abacusNumber = 0` в строке 15 заменить на `abacusNumber = 9999`?
2. Что будет, если `abacusChar.append('O')` в строке 121 заменить на `abacusChar.append('@')`?

71

Повторение музыки

Эта игра на запоминание напоминает электронную игрушку «Саймон» (Simon) и использует сторонний модуль `playsound` для проигрывания четырех различных звуков, соответствующих клавишам A, S, D и F на клавиатуре. По мере того как вы правильно воспроизводите выдаваемую игрой мелодию, она становится все длиннее и длиннее. Сколько звуков вы можете удержать в кратковременной памяти?

Если взглянуть на код, то можно заметить, что в функцию `playsound.playsound()` передается имя звукового файла для проигрывания. Скачать соответствующие звуковые файлы можно по следующим URL:

- https://inventwithpython.com/soundA.wav;
- https://inventwithpython.com/soundS.wav;
- https://inventwithpython.com/soundD.wav;
- https://inventwithpython.com/soundF.wav.

Перед запуском программы поместите эти файлы в тот же каталог, в котором расположен `soundmimic.py`. Подробнее о модуле `playsound` можно узнать на сайте https://pypi.org/project/playsound/. Пользователям, работающим в macOS, для работы `playsound` необходимо также установить модуль `pyobjc` с https://pypi.org/project/pyobjc/.

Программа в действии

Результат выполнения `soundmimic.py` выглядит следующим образом:

```
Sound Mimic, by Al Sweigart al@inventwithpython.com
Try to memorize a pattern of A S D F letters (each with its own sound)
as it gets longer and longer.
Press Enter to begin...
<экран очищается>
Pattern: S
<экран очищается>
Enter the pattern:
> s
Correct!
<экран очищается>
Pattern: S F
<экран очищается>
Enter the pattern:
> sf
Correct!
<экран очищается>
Pattern: S F F
<экран очищается>
Enter the pattern:
> sff
Correct!
<экран очищается>
Pattern: S F F D
--сокращено--
```

Описание работы

Эта программа импортирует модуль `playsound`, проигрывающий звуковые файлы. Модуль включает одну функцию, `playsound()`, которой можно передать имя файла типа `.wav` или `.mp3` для проигрывания. С каждым новым раундом игры программа добавляет в список `pattern` выбранную случайным образом букву (A, S, D или F) и проигрывает содержащиеся в списке звуки. По мере роста списка `pattern` растет и длина мелодии, которую должен запомнить игрок.

```
1. """Повторение музыки, (c) Эл Свейгарт al@inventwithpython.com
2. Игра с подбором звуковых соответствий. Попробуйте запомнить все
3. возрастающую последовательность букв. Создана под впечатлением
4. от электронной игры "Саймон".
5. Код размещен на https://nostarch.com/big-book-small-python-projects
6. Теги: короткая, для начинающих, игра"""
7.
8. import random, sys, time
9.
```

```
10. # Скачайте звуковые файлы с этих URL (или воспользуйтесь своими):
11. # https://inventwithpython.com/soundA.wav
12. # https://inventwithpython.com/soundS.wav
13. # https://inventwithpython.com/soundD.wav
14. # https://inventwithpython.com/soundF.wav
15.
16. try:
17.     import playsound
18. except ImportError:
19.     print('The playsound module needs to be installed to run this')
20.     print('program. On Windows, open a Command Prompt and run:')
21.     print('pip install playsound')
22.     print('On macOS and Linux, open a Terminal and run:')
23.     print('pip3 install playsound')
24.     sys.exit()
25.
26.
27. print('''Sound Mimic, by Al Sweigart al@inventwithpython.com
28. Try to memorize a pattern of A S D F letters (each with its own sound)
29. as it gets longer and longer.''')
30.
31. input('Press Enter to begin...')
32.
33. pattern = ''
34. while True:
35.     print('\n' * 60)  # Очищаем экран, выводя несколько символов новой строки.
36.
37.     # Добавляем в pattern случайную букву:
38.     pattern = pattern + random.choice('ASDF')
39.
40.     # Выводим pattern на экран (и проигрываем соответствующие звуки):
41.     print('Pattern: ', end='')
42.     for letter in pattern:
43.         print(letter, end=' ', flush=True)
44.         playsound.playsound('sound' + letter + '.wav')
45.
46.     time.sleep(1)  # Добавляем в конце небольшую паузу.
47.     print('\n' * 60)  # Очищаем экран с помощью вывода нескольких символов
48.                       # новой строки.
49.     # Просим пользователя ввести последовательность:
50.     print('Enter the pattern:')
51.     response = input('> ').upper()
52.
53.     if response != pattern:
54.         print('Incorrect!')
55.         print('The pattern was', pattern)
56.     else:
57.         print('Correct!')
58.
59.     for letter in pattern:
60.         playsound.playsound('sound' + letter + '.wav')
```

```
61.
62.     if response != pattern:
63.         print('You scored', len(pattern) - 1, 'points.')
64.         print('Thanks for playing!')
65.         break
66.
67.     time.sleep(1)
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать `print('\n' * 60)` в строке 47?
2. Что будет, если `response != pattern` в строке 62 заменить на `False`?

72

Губкорегистр

Наверное, вы видели мем «Язвительный Губка Боб»: картинку с Губкой Бобом Квадратные Штаны, заголовок которой написан в регистре, где саркастично чередуются строчные и заглавные буквы, вот так: «мЕмЫ с ГуБкОй БоБоМ нЕ дЕлАюТ вАс ОсТрОуМнЕе». Ради разнообразия иногда в тексте регистр не чередуется.

Сообщения преобразуются в «губкорегистр» в этой программе с помощью строковых методов `upper()` и `lower()`. Вдобавок программа устроена так, что другие программы могут импортировать ее в качестве модуля, задействуя оператор `import spongecase`, после чего вызывать функцию `spongecase.englishToSpongecase()`.

пРоГрАмМа В дЕйСтВиИ

Результат выполнения `spongecase.py` выглядит следующим образом:

```
sPoNgEcAsE, bY aL sWeIGaRt Al@iNvEnTwItHpYtHoN.cOm

eNtEr YoUr MeSsAgE:
> Using SpongeBob memes does not make you witty.

uSiNg SpOnGeBoB MeMeS dOeS NoT mAkE YoU wItTy.
(cOpIed SpOnGeTexT to ClIpbOaRd.)
```

оПиСаНиЕ рАбОтЫ

В строке 35 кода этой программы используется цикл `for` для обхода всех символов строки `message`. Переменная `useUpper` содержит булево значение, указывающее, нужно ли перевести символ в верхний регистр (`True`) или нижний (`False`). Строки 46 и 47 *переключают* (то есть меняют на противоположное) булево значение в переменной `useUpper` в 90 % итераций. В результате регистр практически всегда чередуется с верхнего на нижний.

```
 1. """гУбКоРеГиСтР, (c) Эл Свейгарт al@inventwithpython.com
 2. Преобразует сообщения на английском языке в гУбКоТеКсТ.
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: крошечная, для начинающих, слова"""
 5.
 6. import random
 7.
 8. try:
 9.     import pyperclip  # pyperclip копирует текст в буфер обмена.
10. except ImportError:
11.     pass  # Если pyperclip не установлена, ничего не делаем. Не проблема.
12.
13.
14. def main():
15.     """Запускаем программу "ГубкоТЕКСТ"."""
16.     print('''sPoNgEcAsE, bY aL sWeIGaRt Al@iNvEnTwItHpYtHoN.cOm
17.
18. eNtEr YoUr MeSsAgE:''')
19.     spongetext = englishToSpongecase(input('> '))
20.     print()
21.     print(spongetext)
22.
23.     try:
24.         pyperclip.copy(spongetext)
25.         print('(cOpIed SpOnGeTexT to ClIpbOaRd.)')
26.     except:
27.         pass  # Если pyperclip не установлена, ничего не делаем.
28.
29.
30. def englishToSpongecase(message):
31.     """Возвращаем заданную строку в губкорегистре."""
32.     spongetext = ''
33.     useUpper = False
34.
35.     for character in message:
36.         if not character.isalpha():
37.             spongetext += character
38.             continue
39.
40.         if useUpper:
41.             spongetext += character.upper()
```

```
42.         else:
43.             spongetext += character.lower()
44.
45.         # Меняем регистр в 90 % случаев.
46.         if random.randint(1, 100) <= 90:
47.             useUpper = not useUpper  # Меняем регистр.
48.     return spongetext
49.
50.
51. # Если программа не импортируется, а запускается, производим запуск:
52. if __name__ == '__main__':
53.     main()
```

ИсСлЕдОвАнИе ПрОгРаММы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `random.randint(1, 100)` в строке 46 заменить на `random.randint(80, 100)`?
2. Что будет, если удалить или закомментировать `useUpper = not useUpper` в строке 47?

73

Головоломка судоку

Судоку — часто встречающаяся в газетах и мобильных приложениях головоломка. Доска судоку представляет собой поле размером 9 × 9, в которой игрок должен расположить цифры от 1 до 9, по одной и только одной в каждом столбце, строке и подсетке 3 × 3. Игра начинается с нескольких клеток — так называемых *подсказок* (givens), уже заполненных цифрами. Правильно составленные судоку имеют только одно решение.

Программа в действии

Результат выполнения `sudoku.py` выглядит следующим образом:

```
Sudoku Puzzle, by Al Sweigart al@inventwithpython.com
--сокращено--
   A B C   D E F   G H I
1  . . . | . . . | . . .
2  . 7 9 | . 5 . | 1 8 .
3  8 . . | . . . | . . 7
   ------+-------+------
4  . . 7 | 3 . 6 | 8 . .
5  4 5 . | 7 . 8 | . 9 6
6  . . 3 | 5 . 2 | 7 . .
   ------+-------+------
7  7 . . | . . . | . . 5
8  . 1 6 | . 3 . | 4 2 .
9  . . . | . . . | . . .
```

```
Enter a move, or RESET, NEW, UNDO, ORIGINAL, or QUIT:
(For example, a move looks like "B4 9".)
--сокращено--
```

Описание работы

Объекты класса `SudokuGrid` представляют собой структуры данных, отражающие поле судоку. Вызывая их методы, можно вносить изменения в поле или извлекать информацию о нем. Например, метод `makeMove()` позволяет поместить в поле новое число, метод `resetGrid()` восстанавливает первоначальное состояние поля, а `isSolved()` возвращает `True`, если все числа решения были расставлены по полю.

В главной части программы, начинающейся со строки 141, с помощью объекта `SudokuGrid` и его методов реализуется игра в судоку, но вы можете повторно использовать функциональность этого класса, скопировав и вставив его в другие созданные вами программы для игры в судоку.

```
  1. """Головоломка судоку, (c) Эл Свейгарт al@inventwithpython.com
  2. Классическая головоломка на расстановку цифр на доске 9 × 9.
  3. Подробнее — в статье https://ru.wikipedia.org/wiki/Судоку
  4. Код размещен на https://nostarch.com/big-book-small-python-projects
  5. Теги: большая, игра, объектно-ориентированная, головоломка"""
  6.
  7. import copy, random, sys
  8.
  9. # Для этой игры необходим файл sudokupuzzle.txt с головоломками.
 10. # Скачать его можно с https://inventwithpython.com/sudokupuzzles.txt
 11. # Пример содержимого этого файла:
 12. # ..3.2.6..9..3.5..1..18.64....81.29..7.......8..67.82....26.95..8..2.3..9..5.
 13. # 1.3..2...8.3...6..7..84.3.5..2.9...1.54.8.........4.27.6...3.1..7.4.72..4..6
 14. # ...4.1...3......9.7...42.18....7.5.261..9.4....5.....4....5.7..992.1.8....34
 15. # .59...5.7.......3..5..4...8.1.5..46.....12.7.5.2.8....6.3....4.1.9.3.25.....
 16. # 98..1.2.6...8..6..2.
 17. # Задаем константы:
 18. EMPTY_SPACE = '.'
 19. GRID_LENGTH = 9
 20. BOX_LENGTH = 3
 21. FULL_GRID_SIZE = GRID_LENGTH * GRID_LENGTH
 22.
 23.
 24. class SudokuGrid:
 25.     def __init__(self, originalSetup):
 26.         # originalSetup — строка из 81 символа для начального состояния
 27.         # головоломки, содержащего числа и точки (на месте пустых клеток).
 28.         # См. https://inventwithpython.com/sudokupuzzles.txt
 29.         self.originalSetup = originalSetup
 30.
```

```
        # Состояние поля судоку представлено в виде ассоциативного
        # массива с ключами (x, y) и значениями — числами (в виде
        # строковых значений) в соответствующей клетке.
        self.grid = {}
        self.resetGrid()  # Устанавливаем поле в начальное состояние.
        self.moves = []  # Отслеживаем все ходы для возможности отката.

    def resetGrid(self):
        """Восстанавливаем состояние поля, отслеживаемое в self.grid,
        до состояния из self.originalSetup."""
        for x in range(1, GRID_LENGTH + 1):
            for y in range(1, GRID_LENGTH + 1):
                self.grid[(x, y)] = EMPTY_SPACE

        assert len(self.originalSetup) == FULL_GRID_SIZE
        i = 0  # i проходит значения от 0 до 80
        y = 0  # y проходит значения от 0 до 8
        while i < FULL_GRID_SIZE:
            for x in range(GRID_LENGTH):
                self.grid[(x, y)] = self.originalSetup[i]
                i += 1
            y += 1

    def makeMove(self, column, row, number):
        """Помещаем число в столбец column (буква от A до I) и строку
        row (число от 1 до 9) в поле."""
        x = 'ABCDEFGHI'.find(column)  # Преобразуем в числовое значение.
        y = int(row) - 1

        # Проверяем, не производится ли ход над клеткой с "подсказкой":
        if self.originalSetup[y * GRID_LENGTH + x] != EMPTY_SPACE:
            return False

        self.grid[(x, y)] = number  # Помещаем данное число на сетку.

        # Необходимо сохранить отдельную копию объекта ассоциативного массива:
        self.moves.append(copy.copy(self.grid))
        return True

    def undo(self):
        """Устанавливаем текущее состояние поля равным предыдущему
        состоянию из списка self.moves."""
        if self.moves == []:
            return  # В self.moves отсутствуют состояния, так что ничего
                    # не делаем.
        self.moves.pop()  # Удаляем текущее состояние.

        if self.moves == []:
            self.resetGrid()
        else:
            # Задаем такое состояние поля, какое было ход назад.
```

```
 82.             self.grid = copy.copy(self.moves[-1])
 83.
 84.     def display(self):
 85.         """Отображаем текущее состояние поля на экране."""
 86.         print('   A B C   D E F   G H I')  # Отображаем метки столбцов.
 87.         for y in range(GRID_LENGTH):
 88.             for x in range(GRID_LENGTH):
 89.                 if x == 0:
 90.                     # Отображаем метку строки:
 91.                     print(str(y + 1) + '  ', end='')
 92.
 93.                 print(self.grid[(x, y)] + ' ', end='')
 94.                 if x == 2 or x == 5:
 95.                     # Выводим на экран вертикальную линию:
 96.                     print('| ', end='')
 97.             print()  # Выводим символ новой строки.
 98.
 99.             if y == 2 or y == 5:
100.                 # Выводим на экран горизонтальную линию:
101.                 print('   ------+-------+------')
102.
103.     def _isCompleteSetOfNumbers(self, numbers):
104.         """Возвращает True, если numbers содержит цифры от 1 до 9."""
105.         return sorted(numbers) == list('123456789')
106.
107.     def isSolved(self):
108.         """Возвращает True, если текущее поле находится в решенном состоянии."""
109.         # Проверяем каждую из строк:
110.         for row in range(GRID_LENGTH):
111.             rowNumbers = []
112.             for x in range(GRID_LENGTH):
113.                 number = self.grid[(x, row)]
114.                 rowNumbers.append(number)
115.             if not self._isCompleteSetOfNumbers(rowNumbers):
116.                 return False
117.
118.         # Проверяем каждый из столбцов:
119.         for column in range(GRID_LENGTH):
120.             columnNumbers = []
121.             for y in range(GRID_LENGTH):
122.                 number = self.grid[(column, y)]
123.                 columnNumbers.append(number)
124.             if not self._isCompleteSetOfNumbers(columnNumbers):
125.                 return False
126.
127.         # Проверяем все субполя 3 × 3:
128.         for boxx in (0, 3, 6):
129.             for boxy in (0, 3, 6):
130.                 boxNumbers = []
131.                 for x in range(BOX_LENGTH):
132.                     for y in range(BOX_LENGTH):
```

```
                        number = self.grid[(boxx + x, boxy + y)]
                        boxNumbers.append(number)
                if not self._isCompleteSetOfNumbers(boxNumbers):
                    return False

        return True


print('''Sudoku Puzzle, by Al Sweigart al@inventwithpython.com

Sudoku is a number placement logic puzzle game. A Sudoku grid is a 9x9
grid of numbers. Try to place numbers in the grid such that every row,
column, and 3x3 box has the numbers 1 through 9 once and only once.

For example, here is a starting Sudoku grid and its solved form:

    5 3 . | . 7 . | . . .     5 3 4 | 6 7 8 | 9 1 2
    6 . . | 1 9 5 | . . .     6 7 2 | 1 9 5 | 3 4 8
    . 9 8 | . . . | . 6 .     1 9 8 | 3 4 2 | 5 6 7
    ------+-------+------     ------+-------+------
    8 . . | . 6 . | . . 3     8 5 9 | 7 6 1 | 4 2 3
    4 . . | 8 . 3 | . . 1 --> 4 2 6 | 8 5 3 | 7 9 1
    7 . . | . 2 . | . . 6     7 1 3 | 9 2 4 | 8 5 6
    ------+-------+------     ------+-------+------
    . 6 . | . . . | 2 8 .     9 6 1 | 5 3 7 | 2 8 4
    . . . | 4 1 9 | . . 5     2 8 7 | 4 1 9 | 6 3 5
    . . . | . 8 . | . 7 9     3 4 5 | 2 8 6 | 1 7 9
''')
input('Press Enter to begin...')


# Загружаем файл sudokupuzzles.txt:
with open('sudokupuzzles.txt') as puzzleFile:
    puzzles = puzzleFile.readlines()

# Удаляем символы новой строки в конце всех головоломок:
for i, puzzle in enumerate(puzzles):
    puzzles[i] = puzzle.strip()

grid = SudokuGrid(random.choice(puzzles))

while True:  # Основной цикл программы.
    grid.display()

    # Проверяем, решена ли головоломка.
    if grid.isSolved():
        print('Congratulations! You solved the puzzle!')
        print('Thanks for playing!')
        sys.exit()
    # Запрашиваем действие пользователя:
```

```
183.     while True:  # Продолжаем спрашивать, пока пользователь не введет
184.                  # допустимое действие.
185.         print()  # Выводим символ новой строки.
186.         print('Enter a move, or RESET, NEW, UNDO, ORIGINAL, or QUIT:')
187.         print('(For example, a move looks like "B4 9".)')
188.
189.         action = input('> ').upper().strip()
190.
191.         if len(action) > 0 and action[0] in ('R', 'N', 'U', 'O', 'Q'):
192.             # Игрок ввел допустимое действие.
193.             break
194.
195.         if len(action.split()) == 2:
196.             space, number = action.split()
197.             if len(space) != 2:
198.                 continue
199.
200.             column, row = space
201.             if column not in list('ABCDEFGHI'):
202.                 print('There is no column', column)
203.                 continue
204.             if not row.isdecimal() or not (1 <= int(row) <= 9):
205.                 print('There is no row', row)
206.                 continue
207.             if not (1 <= int(number) <= 9):
208.                 print('Select a number from 1 to 9, not ', number)
209.                 continue
210.             break  # Игрок ввел допустимый ход.
211.
212.     print()  # Выводим символ новой строки.
213.
214.     if action.startswith('R'):
215.         # Восстанавливаем поле:
216.         grid.resetGrid()
217.         continue
218.
219.     if action.startswith('N'):
220.         # Создаем новую головоломку:
221.         grid = SudokuGrid(random.choice(puzzles))
222.         continue
223.
224.     if action.startswith('U'):
225.         # Возвращаемся на ход назад:
226.         grid.undo()
227.         continue
228.
229.     if action.startswith('O'):
230.         # Просмотр исходных чисел в поле:
231.         originalGrid = SudokuGrid(grid.originalSetup)
232.         print('The original grid looked like this:')
233.         originalGrid.display()
```

```
234.         input('Press Enter to continue...')
235.
236.     if action.startswith('Q'):
237.         # Выходим из игры.
238.         print('Thanks for playing!')
239.         sys.exit()
240.
241.     # Производим выбранный игроком ход.
242.     if grid.makeMove(column, row, number) == False:
243.         print('You cannot overwrite the original grid\'s numbers.')
244.         print('Enter ORIGINAL to view the original grid.')
245.         input('Press Enter to continue...')
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Какая ошибка возникнет, если удалить или переименовать файл `sudokupuzzles.txt` и запустить программу снова?
2. Что будет, если `str(y + 1)` в строке 91 заменить на `str(y)`?
3. Что будет, если `if y == 2 or y == 5:` в строке 99 заменить на `if y == 1` или `y == 6:`?

74

Преобразование текста в речь

В этой программе показано, как использовать (сторонний) модуль `pyttsx3`. Любое введенное сообщение будет произнесено вслух с помощью средств звукового воспроизведения текста операционной системы. И хотя компьютерный синтез речи — чрезвычайно сложная сфера компьютерных наук, модуль `pyttsx3` предоставляет удобный интерфейс, делающий эту программу доступной даже для начинающих. Научившись использовать его, вы сможете встраивать синтез речи в собственные программы.

Подробнее о модуле `pyttsx3` можно прочитать на https://pypi.org/project/pyttsx3/.

Программа в действии

Результат выполнения `texttospeechtalker.py` выглядит следующим образом:

```
Text To Speech Talker, by Al Sweigart al@inventwithpython.com
Text-to-speech using the pyttsx3 module, which in turn uses
the NSSpeechSynthesizer (on macOS), SAPI5 (on Windows), or
eSpeak (on Linux) speech engines.

Enter the text to speak, or QUIT to quit.
> Hello. My name is Guido van Robot.
<компьютер произносит текст>
> quit
Thanks for playing!
```

Описание работы

Своей краткостью данная программа обязана тому, что модуль `pyttsx3` берет на себя все преобразование текста в речь. Для работы с этим модулем необходимо его установить, следуя инструкциям из введения к данной книге. После этого модуль можно будет импортировать в сценарии Python с помощью оператора `import pyttsx3` и вызвать функцию `pyttsc3.init()`. Она возвращает объект `Engine`, который, собственно, и представляет собой механизм преобразования текста в речь. Если передать в метод `say()` этого объекта строковое значение, то компьютер произнесет его вслух при вызове метода `runAndWait()`.

```
 1. """Преобразование текста в речь, (c) Эл Свейгарт al@inventwithpython.com
 2. Пример программы, использующей возможности преобразования текста в речь
 3. модуля pyttsx3.
 4. Код размещен на https://nostarch.com/big-book-small-python-projects
 5. Теги: крошечная, для начинающих"""
 6.
 7. import sys
 8.
 9. try:
10.     import pyttsx3
11. except ImportError:
12.     print('The pyttsx3 module needs to be installed to run this')
13.     print('program. On Windows, open a Command Prompt and run:')
14.     print('pip install pyttsx3')
16.     print('On macOS and Linux, open a Terminal and run:')
16.     print('pip3 install pyttsx3')
17.     sys.exit()
18.
19. tts = pyttsx3.init()  # Инициализация механизма TTS.
20.
21. print('Text To Speech Talker, by Al Sweigart al@inventwithpython.com')
22. print('Text-to-speech using the pyttsx3 module, which in turn uses')
23. print('the NSSpeechSynthesizer (on macOS), SAPI5 (on Windows), or')
24. print('eSpeak (on Linux) speech engines.')
25. print()
26. print('Enter the text to speak, or QUIT to quit.')
27. while True:
28.     text = input('> ')
29.
30.     if text.upper() == 'QUIT':
31.         print('Thanks for playing!')
32.         sys.exit()
33.
34.     tts.say(text)  # Текст, который должен произнести механизм TTS.
35.     tts.runAndWait()  # Запускаем произнесение этого текста механизмом TTS.
```

Исследование программы

Это очень простая программа, так что вариантов ее модификации не так уж много. Вместо этого подумайте, в каких других ваших программах могло бы пригодиться преобразование текста в речь.

75

Три карты Монте

«Три карты Монте» — мошенничество, часто применяемое для облапошивания легковерных туристов и прочих простофиль. На картонную коробку кладутся рубашкой вверх три игральные карты, одна из которых — дама червей. Сдающий карты проворно их перетасовывает, после чего просит простофилю показать, какая из них — дама червей. При этом сдающий применяет все возможные уловки, чтобы спрятать карту или сжульничать каким-либо другим образом, лишая жертву всякого шанса выиграть. У сдающего также часто бывают сообщники в толпе зрителей, якобы выигрывающие игру (чтобы жертва думала, что тоже может выиграть) или специально с треском проигрывающие (чтобы жертва думала, что может сыграть намного лучше).

Наша программа демонстрирует три карты, а затем быстро производит ряд их перестановок. В конце она очищает экран и просит игрока выбрать карту. Сможете ли вы уследить за «красной леди»? Чтобы почувствовать по-настоящему, как играют в три карты, можете включить опцию жульничества, в результате чего игрок всегда будет проигрывать, даже если выберет нужную карту.

Программа в действии

Результат выполнения `threecardmonte.py` выглядит следующим образом:

```
Three-Card Monte, by Al Sweigart al@inventwithpython.com

Find the red lady (the Queen of Hearts)! Keep an eye on how
the cards move.

Here are the cards:

 ___  ___  ___
|J  | |Q  | |8  |
| ♦ | | ♥ | | ♣ |
|__J| |__Q| |__8|
Press Enter when you are ready to begin...
swapping left and middle...
swapping right and middle...
swapping middle and left...
swapping right and left...
swapping left and middle...
--сокращено--
<screen clears>
Which card has the Queen of Hearts? (LEFT MIDDLE RIGHT)
> middle

 ___  ___  ___
|Q  | |8  | |J  |
| ♥ | | ♣ | | ♦ |
|__Q| |__8| |__J|
You lost!
Thanks for playing, sucker!
```

Описание работы

Каждая из игральных карт в этой программе представлена с помощью кортежа `(достоинство, масть)`. Достоинство описывается строковым значением, например, `'2'`, `'10'`, `'Q'` или `'K'`, а масть представляет собой строковое значение с эмодзи червей, треф, пик или бубен. Поскольку с помощью клавиатуры ввести символы эмодзи нельзя, мы вызовем для их генерации в строках с 16-й по 19-ю функцию `chr()`. Кортеж `('9', '♦')` соответствует девятке бубен.

Вместо того чтобы выводить эти кортежи непосредственно, функция `displayCards()` в строках с 28-й по 43-ю анализирует их и выводит на экран их представления в виде ASCII-графики, как в проекте 4. Аргумент `cards` данной функции представляет собой список кортежей игральных карт, благодаря чему можно выводить несколько карт в ряд.

```
"""Три карты Монте (c) Эл Свейгарт al@inventwithpython.com
Найдите даму червей после перемешивания карт.
(В реальной жизни мошенник обычно прячет даму червей в руке, так что
вы никогда не выиграете.)
Подробнее — в статье на https://en.wikipedia.org/wiki/Three-card_Monte
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, карточная игра, игра"""

import random, time

# Задаем константы:
NUM_SWAPS = 16   # (!) Попробуйте заменить это значение на 30 или 100.
DELAY     = 0.8  # (!) Попробуйте заменить это значение на 2.0 или 0.0.

# Символы карточных мастей:
HEARTS   = chr(9829)  # Символ 9829 – '♥'
DIAMONDS = chr(9830)  # Символ 9830 – '♦'
SPADES   = chr(9824)  # Символ 9824 – '♠'
CLUBS    = chr(9827)  # Символ 9827 – '♣'
# Список кодов chr можно найти на https://inventwithpython.com/chr

# Индексы списка из трех карт:
LEFT   = 0
MIDDLE = 1
RIGHT  = 2


def displayCards(cards):
    """Отображает карты из списка cards кортежей (достоинство, масть).
    """
    rows = ['', '', '', '', '']  # Содержит текст для вывода на экран.

    for i, card in enumerate(cards):
        rank, suit = card  # card представляет собой структуру
                           # данных — кортеж.
        rows[0] += ' ___  '  # Выводим верхнюю линию карты.
        rows[1] += '|{} | '.format(rank.ljust(2))
        rows[2] += '| {} | '.format(suit)
        rows[3] += '|_{}| '.format(rank.rjust(2, '_'))

    # Построчно выводим на экран:
    for i in range(5):
        print(rows[i])


def getRandomCard():
    """Возвращает случайную карту — НЕ даму червей."""
    while True:  # Подбираем карты, пока не получим НЕ даму червей.
        rank = random.choice(list('23456789JQKA') + ['10'])
        suit = random.choice([HEARTS, DIAMONDS, SPADES, CLUBS])
```

```
51.
52.         # Возвращаем карту, если это не дама червей:
53.         if rank != 'Q' and suit != HEARTS:
54.             return (rank, suit)
55.
56.
57. print('Three-Card Monte, by Al Sweigart al@inventwithpython.com')
58. print()
59. print('Find the red lady (the Queen of Hearts)! Keep an eye on how')
60. print('the cards move.')
61. print()
62.
63. # Отображаем исходную раскладку карт:
64. cards = [('Q', HEARTS), getRandomCard(), getRandomCard()]
65. random.shuffle(cards)  # Помещаем даму червей в случайное место.
66. print('Here are the cards:')
67. displayCards(cards)
68. input('Press Enter when you are ready to begin...')
69.
70. # Отображаем на экране перетасовки карт:
71. for i in range(NUM_SWAPS):
72.     swap = random.choice(['l-m', 'm-r', 'l-r', 'm-l', 'r-m', 'r-l'])
73.
74.     if swap == 'l-m':
75.         print('swapping left and middle...')
76.         cards[LEFT], cards[MIDDLE] = cards[MIDDLE], cards[LEFT]
77.     elif swap == 'm-r':
78.         print('swapping middle and right...')
79.         cards[MIDDLE], cards[RIGHT] = cards[RIGHT], cards[MIDDLE]
80.     elif swap == 'l-r':
81.         print('swapping left and right...')
82.         cards[LEFT], cards[RIGHT] = cards[RIGHT], cards[LEFT]
83.     elif swap == 'm-l':
84.         print('swapping middle and left...')
85.         cards[MIDDLE], cards[LEFT] = cards[LEFT], cards[MIDDLE]
86.     elif swap == 'r-m':
87.         print('swapping right and middle...')
88.         cards[RIGHT], cards[MIDDLE] = cards[MIDDLE], cards[RIGHT]
89.     elif swap == 'r-l':
90.         print('swapping right and left...')
91.         cards[RIGHT], cards[LEFT] = cards[LEFT], cards[RIGHT]
92.
93.     time.sleep(DELAY)
94.
95. # Выводим несколько символов новой строки, чтобы скрыть перетасовки.
96. print('\n' * 60)
97.
98. # Просим пользователя найти даму червей:
99. while True:  # Спрашиваем, пока не будет введено LEFT, MIDDLE или RIGHT.
100.     print('Which card has the Queen of Hearts? (LEFT MIDDLE RIGHT)')
```

```
    guess = input('> ').upper()

    # Находим индекс в cards введенной пользователем позиции:
    if guess in ['LEFT', 'MIDDLE', 'RIGHT']:
        if guess == 'LEFT':
            guessIndex = 0
        elif guess == 'MIDDLE':
            guessIndex = 1
        elif guess == 'RIGHT':
            guessIndex = 2
        break

# (!) Раскомментируйте этот код, чтобы игрок всегда проигрывал:
#if cards[guessIndex] == ('Q', HEARTS):
#    # Игрок выиграл, так что перемещаем даму.
#    possibleNewIndexes = [0, 1, 2]
#    possibleNewIndexes.remove(guessIndex)  # Убираем индекс дамы.
#    newInd = random.choice(possibleNewIndexes)  # Выбираем новый индекс.
#    # Помещаем даму червей на позицию, соответствующую новому индексу:
#    cards[guessIndex], cards[newInd] = cards[newInd], cards[guessIndex]

displayCards(cards)  # Отображаем все карты.

# Проверяем, выиграл ли игрок:
if cards[guessIndex] == ('Q', HEARTS):
    print('You won!')
    print('Thanks for playing!')
else:
    print('You lost!')
    print('Thanks for playing, sucker!')
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!). Можете также сами попробовать придумать, как сделать следующее:

- воспользоваться методикой вывода символов возврата на одну позицию из проекта 57, чтобы отобразить на краткое время каждое из сообщений о перестановке карт, после чего с помощью символа \b убрать его перед выводом следующего;
- создать игру Монте на четыре карты, чтобы усложнить задачу игрока.

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `[('Q', HEARTS), getRandomCard(), getRandomCard()]` в строке 64 заменить на `[('Q', HEARTS), ('Q', HEARTS), ('Q', HEARTS)]`?
2. Что будет, если выражение `list('23456789JQKA')` в строке 49 заменить на `list('ABCDEFGHIJK')`?
3. Что будет, если удалить или закомментировать `time.sleep(DELAY)` в строке 93?

76

Крестики-нолики

Крестики-нолики — классическая игра на бумаге, играемая на поле 3 × 3. Игроки по очереди рисуют свои метки X и O, пытаясь выстроить их три в ряд. Большинство игр в крестики-нолики заканчиваются ничьей, хотя можно и перехитрить противника, если он недостаточно внимателен.

Программа в действии

Результат выполнения `tictactoe.py` выглядит следующим образом:

```
Welcome to Tic-Tac-Toe!

       | |   1 2 3
      -+-+-
       | |   4 5 6
      -+-+-
       | |   7 8 9
What is X's move? (1-9)
> 1

      X| |   1 2 3
      -+-+-
       | |   4 5 6
      -+-+-
       | |   7 8 9
What is O's move? (1-9)
--сокращено--
```

```
     X|O|X  1 2 3
     -+-+-
     X|O|O  4 5 6
     -+-+-
     O|X|X  7 8 9
The game is a tie!
Thanks for playing!
```

Описание работы

Для представления поля крестиков-ноликов в данной программе используется ассоциативный массив с ключами от `'1'` до `'9'` для отдельных клеток доски. Нумеруются клетки при этом точно так же, как и на клавиатуре телефона. Значениями в этом ассоциативном массиве служат символы `'X'` и `'O'` для оставленных игроками отметок и `' '` для пустой клетки.

```
"""Крестики-нолики, (c) Эл Свейгарт al@inventwithpython.com
Классическая настольная игра.
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: короткая, настольная игра, игра, для двух игроков"""

ALL_SPACES = ['1', '2', '3', '4', '5', '6', '7', '8', '9']
X, O, BLANK = 'X', 'O', ' '  # Константы для строковых значений.

def main():
    print('Welcome to Tic-Tac-Toe!')
    gameBoard = getBlankBoard()  # Создаем ассоциативный массив для доски
                                 # крестиков-ноликов.
    currentPlayer, nextPlayer = X, O  # Сначала ходит X, а затем O.

    while True:  # Основной цикл программы.
        # Отображаем доску на экране:
        print(getBoardStr(gameBoard))

        # Спрашиваем игрока, пока он не введет число от 1 до 9:
        move = None
        while not isValidSpace(gameBoard, move):
            print('What is {}\'s move? (1-9)'.format(currentPlayer))
            move = input('> ')
        updateBoard(gameBoard, move, currentPlayer)  # Делаем ход.

        # Проверяем, не закончилась ли игра:
        if isWinner(gameBoard, currentPlayer):  # Проверяем, кто победил.
            print(getBoardStr(gameBoard))
            print(currentPlayer + ' has won the game!')
            break
        elif isBoardFull(gameBoard):  # Проверяем на ничью.
            print(getBoardStr(gameBoard))
            print('The game is a tie!')
```

```
34.             break
35.         # Переход хода к следующему игроку:
36.         currentPlayer, nextPlayer = nextPlayer, currentPlayer
37.     print('Thanks for playing!')
38.
39.
40. def getBlankBoard():
41.     """Создаем новую пустую доску для крестиков-ноликов."""
42.     # Карта номеров клеток: 1|2|3
43.     #                       -+-+-
44.     #                       4|5|6
45.     #                       -+-+-
46.     #                       7|8|9
47.     # Ключи — числа от 1 до 9, значения — X, O и BLANK:
48.     board = {}
49.     for space in ALL_SPACES:
50.         board[space] = BLANK  # Все клетки начинаются в пустом состоянии.
51.     return board
52.
53.
54. def getBoardStr(board):
55.     """Возвращает текстовое представление доски."""
56.     return '''
57.       {}|{}|{}  1 2 3
58.       -+-+-
59.       {}|{}|{}  4 5 6
60.       -+-+-
61.       {}|{}|{}  7 8 9'''.format(board['1'], board['2'], board['3'],
62.                                 board['4'], board['5'], board['6'],
63.                                 board['7'], board['8'], board['9'])
64.
65. def isValidSpace(board, space):
66.     """Возвращает True, если space на board представляет собой
67.     допустимый номер клетки, причем эта клетка пуста."""
68.     return space in ALL_SPACES and board[space] == BLANK
69.
70.
71. def isWinner(board, player):
72.     """Возвращает True, если игрок player победил на этой доске."""
73.     # Для удобочитаемости используются более короткие названия переменных:
74.     b, p = board, player
75.     # Проверяем наличие трех отметок на одной из трех строк,
        # двух диагоналей или в одном из трех столбцов.
76.     return ((b['1'] == b['2'] == b['3'] == p) or  # Верхняя строка
77.             (b['4'] == b['5'] == b['6'] == p) or  # Средняя строка
78.             (b['7'] == b['8'] == b['9'] == p) or  # Нижняя строка
79.             (b['1'] == b['4'] == b['7'] == p) or  # Левый столбец
80.             (b['2'] == b['5'] == b['8'] == p) or  # Средний столбец
81.             (b['3'] == b['6'] == b['9'] == p) or  # Нижний столбец
82.             (b['3'] == b['5'] == b['7'] == p) or  # Диагональ
83.             (b['1'] == b['5'] == b['9'] == p))    # Диагональ
```

```
84.
85. def isBoardFull(board):
86.     """Возвращает True, если все клетки на доске заполнены."""
87.     for space in ALL_SPACES:
88.         if board[space] == BLANK:
89.             return False  # Если хоть одна клетка пуста — возвращаем False.
90.     return True  # Незаполненных клеток нет, возвращаем True.
91.
92.
93. def updateBoard(board, space, mark):
94.     """Присваиваем клетке (space) на доске (board) значение (mark)."""
95.     board[space] = mark
96.
97. if __name__ == '__main__':
98.     main()  # Вызываем main(), если этот модуль не импортируется,
99.             # а запускается.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `X, O, BLANK = 'X', 'O', ' '` в строке 7 заменить на `X, O, BLANK = = 'X', 'X', ' '`?
2. Что будет, если `board[space] = mark` в строке 95 заменить на `board[space] = X`?
3. Что будет, если `board[space] = BLANK` в строке 50 заменить на `board[space] = X`?

77

Ханойская башня

«Ханойская башня» — головоломка, в которой дано три стержня, на один из которых нанизаны диски различного размера. Цель игры — перенести стопку дисков на другой стержень. При этом можно переносить за один раз только один диск и нельзя класть больший диск на меньший. Решение данной головоломки требует определенного алгоритма. Сможете ли вы додуматься до него? (Подсказка: попробуйте сначала решить более простой вариант задачи, в котором переменная `TOTAL_DISKS` равна `3` или `4`.)

Программа в действии

Результат выполнения `towerofhanoi.py` выглядит следующим образом:

```
The Tower of Hanoi, by Al Sweigart al@inventwithpython.com

Move the tower of disks, one disk at a time, to another tower. Larger
disks cannot rest on top of a smaller disk.

More info at https://en.wikipedia.org/wiki/Tower_of_Hanoi

     ||          ||          ||
    @_1@         ||          ||
   @@_2@@        ||          ||
  @@@_3@@@       ||          ||
 @@@@_4@@@@      ||          ||
@@@@@_5@@@@@     ||          ||
      A           B           C
```

```
Enter the letters of "from" and "to" towers, or QUIT.
(e.g. AB to moves a disk from tower A to tower B.)
> ab
     ||          ||          ||
     ||          ||          ||
   @@_2@@        ||          ||
  @@@_3@@@       ||          ||
 @@@@_4@@@@      ||          ||
@@@@@_5@@@@@    @_1@         ||
      A           B           C

Enter the letters of "from" and "to" towers, or QUIT.
(e.g. AB to moves a disk from tower A to tower B.)
--сокращено--
```

Описание работы

В качестве структуры данных для представления башни в нашей программе используется список. Первое число в списке соответствует нижнему диску, а последнее — верхнему. Например, `[5, 4, 2]` соответствует следующей башне:

```
     ||
     ||
   @@_2@@
 @@@@_4@@@@
@@@@@_5@@@@@
```

Методы `append()` и `pop()` списков Python позволяют добавлять и удалять значения из конца списка соответственно. Подобно тому как выражения `someList[0]` и `someList[1]` позволяют обращаться к первому и второму элементам списка, можно использовать отрицательные индексы для обращения к значениям с конца списка с помощью выражений вида `someList[-1]` и `someList[-2]` для последнего и предпоследнего значений в списке соответственно, что очень удобно для выбора диска, находящегося на верху башни.

```
 1. """Ханойская башня, (c) Эл Свейгарт al@inventwithpython.com
 2. Головоломка с переносом столбиков
 3. Код размещен на https://nostarch.com/big-book-small-python-projects
 4. Теги: короткая, игра, головоломка"""
 5.
 6. import copy
 7. import sys
 8.
 9. TOTAL_DISKS = 5  # Чем больше дисков, тем сложнее головоломка.
10.
11. # В начале все диски находятся на башне A:
12. COMPLETE_TOWER = list(range(TOTAL_DISKS, 0, -1))
13.
```

```
14.
15. def main():
16.     print("""The Tower of Hanoi, by Al Sweigart al@inventwithpython.com
17.
18. Move the tower of disks, one disk at a time, to another tower. Larger
19. disks cannot rest on top of a smaller disk.
20.
21. More info at https://en.wikipedia.org/wiki/Tower_of_Hanoi
22. """
23.     )
24.
25.     # Задаем башни. Конец списка соответствует верху башни.
26.     towers = {'A': copy.copy(COMPLETE_TOWER), 'B': [], 'C': []}
27.
28.     while True:  # Выполняем один ход.
29.         # Отображаем башни и диски:
30.         displayTowers(towers)
31.
32.         # Просим пользователя сделать ход:
33.         fromTower, toTower = askForPlayerMove(towers)
34.
35.         # Переносим верхний диск с fromTower на toTower:
36.         disk = towers[fromTower].pop()
37.         towers[toTower].append(disk)
38.
39.         # Проверяем, не решил ли уже головоломку пользователь:
40.         if COMPLETE_TOWER in (towers['B'], towers['C']):
41.             displayTowers(towers)  # Отображаем башни последний раз.
42.             print('You have solved the puzzle! Well done!')
43.             sys.exit()
44.
45. def askForPlayerMove(towers):
46.     """Просит игрока сделать ход. Возвращает (fromTower, toTower)."""
47.
48.     while True:  # Продолжаем спрашивать игрока, пока он не введет допустимый
49.                  # ход.
50.         print('Enter the letters of "from" and "to" towers, or QUIT.')
51.         print('(e.g. AB to moves a disk from tower A to tower B.)')
52.         response = input('> ').upper().strip()
53.
54.         if response == 'QUIT':
55.             print('Thanks for playing!')
56.             sys.exit()
57.
58.         # Убеждаемся, что игрок ввел корректные буквы башен:
59.         if response not in ('AB', 'AC', 'BA', 'BC', 'CA', 'CB'):
60.             print('Enter one of AB, AC, BA, BC, CA, or CB.')
61.             continue  # Просим игрока ввести ход снова.
62.
63.         # Синтаксический сахар — более наглядные названия переменных:
64.         fromTower, toTower = response[0], response[1]
```

```
65.
66.         if len(towers[fromTower]) == 0:
67.             # Исходная башня не должна быть пустой:
68.             print('You selected a tower with no disks.')
69.             continue  # Просим игрока ввести ход снова.
70.         elif len(towers[toTower]) == 0:
71.             # На пустую целевую башню можно перенести любой диск:
72.             return fromTower, toTower
73.         elif towers[toTower][-1] < towers[fromTower][-1]:
74.             print('Can\'t put larger disks on top of smaller ones.')
75.             continue  # Просим игрока ввести ход снова.
76.         else:
77.             # Ход допустим, возвращаем выбранные башни:
78.             return fromTower, toTower
79.
80.
81. def displayTowers(towers):
82.     """Отображаем текущее состояние."""
83.
84.     # Отображаем три башни:
85.     for level in range(TOTAL_DISKS, -1, -1):
86.         for tower in (towers['A'], towers['B'], towers['C']):
87.             if level >= len(tower):
88.                 displayDisk(0)  # Отображаем пустой стержень без дисков.
89.             else:
90.                 displayDisk(tower[level])  # Отображаем диск.
91.         print()
92.
93.     # Отображаем метки башен A, B и C.
94.     emptySpace = ' ' * (TOTAL_DISKS)
95.     print('{0} A{0}{0} B{0}{0} C\n'.format(emptySpace))
96.
97.
98. def displayDisk(width):
99.     """Отображаем диск заданной ширины. Ширина 0 означает отсутствие диска."""
100.    emptySpace = ' ' * (TOTAL_DISKS - width)
101.
102.    if width == 0:
103.        # Отображаем сегмент стержня без диска:
104.        print(emptySpace + '||' + emptySpace, end='')
105.    else:
106.        # Отображаем диск:
107.        disk = '@' * width
108.        numLabel = str(width).rjust(2, '_')
109.        print(emptySpace + disk + numLabel + disk + emptySpace, end='')
110.
111.
112. # Если программа не импортируется, а запускается, производим запуск игры:
113. if __name__ == '__main__':
114.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если удалить или закомментировать строки 73, 74 и 75?
2. Что будет, если `emptySpace = ' ' * (TOTAL_DISKS - width)` в строке 100 заменить на `emptySpace = ' '`?
3. Что будет, если `width == 0` в строке 102 заменить на `width != 0`?

78

Вопросы с подвохом

Каким станет желтый камень, если его бросить в голубой пруд? Существует ли праздник 4 Июля в Англии? Как может доктор обходиться 30 дней без сна? Если вы думаете, что знаете ответы на эти вопросы, то, вероятно, ошибаетесь: 54 вопроса в этой программе были специально составлены так, чтобы ответы на них были просты, очевидны и неправильны. Поиск настоящих ответов требует определенных умственных способностей.

Не копируйте код из этой книги, чтобы не портить все удовольствие, ведь тогда вы сразу увидите ответы. Лучше скачайте эту игру с https://inventwithpython.com/trickquestions.py и запустите ее, не глядя на исходный код.

Программа в действии

Результат выполнения `trickquestions.py` выглядит следующим образом:

```
Trick Questions, by Al Sweigart al@inventwithpython.com

Can you figure out the answers to these trick questions?
(Enter QUIT to quit at any time.)

Press Enter to begin...
--сокращено--
Question: 1
```

```
Score: 0 / 54
QUESTION: A 39 year old person was born on the 22nd of February. What year is
their birthday?
  ANSWER: 1981
Incorrect! The answer is: Their birthday is on February 22nd of every year.
Press Enter for the next question...
--сокращено--
Question: 2
Score: 0 / 54
QUESTION: If there are ten apples and you take away two, how many do you have?
  ANSWER: Eight
Incorrect! The answer is: Two.
Press Enter for the next question...
--сокращено--
```

Описание работы

В переменной `QUESTIONS` содержится список ассоциативных массивов. Каждый массив соответствует одному вопросу с подвохом и содержит ключи `'question'`, `'answer'` и `'accept'`. Значения для ключей `'question'` и `'answer'` представляют собой строковые значения, соответственно отображаемые программой в момент, когда задается вопрос игроку или показывается ответ. Значение для ключа `'accept'` представляет собой список строковых значений. Любое введенное игроком значение из этого списка считается правильным, что позволяет ему вводить в качестве ответа текст в свободной форме. Программа достаточно точно определяет, правильный ли ответ получила.

```
  1. """Вопросы с подвохом, (c) Эл Свейгарт al@inventwithpython.com
  2. Викторина из нескольких вопросов с подвохом.
  3. Код размещен на https://nostarch.com/big-book-small-python-projects
  4. Теги: большая, юмор"""
  5.
  6. import random, sys
  7.
  8. # QUESTIONS представляет собой список ассоциативных массивов, каждый
  9. # из которых содержит вопрос с подвохом и ответ на него. В таком
 10. # ассоциативном массиве есть ключи 'question' (с текстом вопроса),
 11. # 'answer' (с текстом ответа) и 'accept' (со списком строковых
 12. # значений, которые программа считает правильными ответами на вопрос).
 13. # (!) Придумайте собственные вопросы с подвохом и добавьте сюда:
 14. QUESTIONS = [
 15.  {'question': "How many times can you take 2 apples from a pile of 10 apples?",
 16.   'answer': "Once. Then you have a pile of 8 apples.",
 17.   'accept': ['once', 'one', '1']},
 18.  {'question': 'What begins with "e" and ends with "e" but only has one
      ➥ letter in it?',
 19.   'answer': "An envelope.",
 20.   'accept': ['envelope']},
```

```
21.  {'question': "Is it possible to draw a square with three sides?",
22.   'answer': "Yes. All squares have three sides. They also have a fourth side.",
23.   'accept': ['yes']},
24.  {'question': "How many times can a piece of paper be folded in half by hand
     ➥ without unfolding?",
25.   'answer': "Once. Then you are folding it in quarters.",
26.   'accept': ['one', '1', 'once']},
27.  {'question': "What does a towel get as it dries?",
28.   'answer': "Wet.",
29.   'accept': ['wet']},
30.  {'question': "What does a towel get as it dries?",
31.   'answer': "Drier.",
32.   'accept': ['drier', 'dry']},
33.  {'question': "Imagine you are in a haunted house full of evil ghosts. What do
     ➥ you have to do to stay safe?",
34.   'answer': "Nothing. You're only imagining it.",
35.   'accept': ['nothing', 'stop']},
36.  {'question': "A taxi driver is going the wrong way down a one-way street. She
     ➥ passes ten cops but doesn't get a ticket. Why not?",
37.   'answer': "She was walking.",
38.   'accept': ['walk']},
39.  {'question': "What does a yellow stone thrown into a blue pond become?",
40.   'answer': "Wet.",
41.   'accept': ['wet']},
42.  {'question': "How many miles does must a cyclist bike to get to training?",
43.   'answer': "None. They're training as soon as they get on the bike.",
44.   'accept': ['none', 'zero', '0']},
45.  {'question': "What building do people want to leave as soon as they enter?",
46.   'answer': "An airport.",
47.   'accept': ['airport', 'bus', 'port', 'train', 'station', 'stop']},
48.  {'question': "If you're in the middle of a square house facing the west side
     ➥ with the south side to your left and the north side to your right, which
     ➥ side of the house are you next to?",
49.   'answer': "None. You're in the middle.",
50.   'accept': ['none', 'middle', 'not', 'any']},
51.  {'question': "How much dirt is in a hole 3 meters wide, 3 meters long,
     ➥ and 3 meters deep?",
52.   'answer': "There is no dirt in a hole.",
53.   'accept': ['no', 'none', 'zero']},
54.  {'question': "A girl mails a letter from America to Japan. How many miles did
     ➥ the stamp move?",
55.   'answer': "Zero. The stamp was in the same place on the envelope the
      ➥ whole time.",
56.   'accept': ['zero', '0', 'none', 'no']},
57.  {'question': "What was the highest mountain on Earth the day before Mount
     ➥ Everest was discovered?",
58.   'answer': "Mount Everest was still the highest mountain of Earth the day
      ➥ before it was discovered.",
59.   'accept': ['everest']},
60.  {'question': "How many fingers do most people have on their two hands?",
61.   'answer': "Eight. They also have two thumbs.",
```

```
 62.   'accept': ['eight', '8']},
 63.  {'question': "The 4th of July is a holiday in America. Do they have
      ➥ a 4th of July in England?",
 64.   'answer': "Yes. All countries have a 4th of July on their calendar.",
 65.   'accept': ['yes']},
 66.  {'question': "Which letter of the alphabet makes honey?",
 67.   'answer': "None. A bee is an insect, not a letter.",
 68.   'accept': ['no', 'none', 'not']},
 69.  {'question': "How can a doctor go 30 days without sleep?",
 70.   'answer': "By sleeping at night.",
 71.   'accept': ['night', 'evening']},
 72.  {'question': "How many months have 28 days?",
 73.   'answer': "12. All months have 28 days. Some have more days as well.",
 74.   'accept': ['12', 'twelve', 'all']},
 75.  {'question': "How many two cent stamps are in a dozen?",
 76.   'answer': "A dozen.",
 77.   'accept': ['12', 'twelve', 'dozen']},
 78.  {'question': "Why is it illegal for a person living in North Dakota to be
      ➥ buried in South Dakota?",
 79.   'answer': "Because it is illegal to bury someone alive.",
 80.   'accept': ['alive', 'living', 'live']},
 81.  {'question': "How many heads does a two-headed coin have?",
 82.   'answer': "Zero. Coins are just circular pieces of metal. They don't have
       ➥ heads.",
 83.   'accept': ['zero', 'none', 'no', '0']},
 84.  {'question': "What kind of vehicle has four wheels and flies?",
 85.   'answer': "A garbage truck.",
 86.   'accept': ['garbage', 'dump', 'trash']},
 87.  {'question': "What kind of vehicle has four wheels and flies?",
 88.   'answer': "An airplane.",
 89.   'accept': ['airplane', 'plane']},
 90.  {'question': "What five-letter word becomes shorter by adding two letters?",
 91.   'answer': "Short.",
 92.   'accept': ['short']},
 93.  {'question': "Gwen's mother has five daughters. Four are named Haha, Hehe,
      ➥ Hihi, and Hoho. What's the fifth daughter's name?",
 94.   'answer': "Gwen.",
 95.   'accept': ['gwen']},
 96.  {'question': "How long is a fence if there are three fence posts each one
      ➥ meter apart?",
 97.   'answer': "Two meters long.",
 98.   'accept': ['2', 'two']},
 99.  {'question': "How many legs does a dog have if you count its tail as a leg?",
100.   'answer': "Four. Calling a tail a leg doesn't make it one.",
101.   'accept': ['four', '4']},
102.  {'question': "How much more are 1976 pennies worth compared to 1975 pennies?",
103.   'answer': "One cent.",
104.   'accept': ['1', 'one']},
105.  {'question': "What two things can you never eat for breakfast?",
106.   'answer': "Lunch and dinner.",
107.   'accept': ['lunch', 'dinner', 'supper']},
```

```
108.  {'question': "How many birthdays does the average person have?",
109.   'answer': "One. You're only born once.",
110.   'accept': ['one', '1', 'once' 'born']},
111.  {'question': "Where was the United States Declaration of Independence
      ➥ signed?",
112.   'answer': "It was signed at the bottom.",
113.   'accept': ['bottom']},
114.  {'question': "A person puts two walnuts in their pocket but only has one
        thing in their pocket five minutes later. What is it?",
115.   'answer': "A hole.",
116.   'accept': ['hole']},
117.  {'question': "What did the sculptor make that no one could see?",
118.   'answer': "Noise.",
119.   'accept': ['noise']},
120.  {'question': "If you drop a raw egg on a concrete floor, will it crack?",
121.   'answer': "No. Concrete is very hard to crack.",
122.   'accept': ['no']},
123.  {'question': "If it takes ten people ten hours to build a fence, how many
      ➥ hours does it take five people to build it?",
124.   'answer': "Zero. It's already built.",
125.   'accept': ['zero', 'no', '0', 'already', 'built']},
126.  {'question': "Which is heavier, 100 pounds of rocks or 100 pounds of
      ➥ feathers?",
127.   'answer': "Neither. They weigh the same.",
128.   'accept': ['neither', 'none', 'no', 'same', 'even', 'balance']},
129.  {'question': "What do you have to do to survive being bitten by a poisonous
      ➥ snake?",
130.   'answer': "Nothing. Only venomous snakes are deadly.",
131.   'accept': ['nothing', 'anything']},
132.  {'question': "What three consecutive days don't include Sunday, Wednesday, or
        Friday?",
133.   'answer': "Yesterday, today, and tomorrow.",
134.   'accept': ['yesterday', 'today', 'tomorrow']},
135.  {'question': "If there are ten apples and you take away two, how many do you
      ➥ have?",
136.   'answer': "Two.",
137.   'accept': ['2', 'two']},
138.  {'question': "A 39 year old person was born on the 22nd of February. What
      ➥ year is their birthday?",
139.   'answer': "Their birthday is on February 22nd of every year.",
140.   'accept': ['every', 'each']},
141.  {'question': "How far can you walk in the woods?",
142.   'answer': "Halfway. Then you are walking out of the woods.",
143.   'accept': ['half', '1/2']},
144.  {'question': "Can a man marry his widow's sister?",
145.   'answer': "No, because he's dead.",
146.   'accept': ['no']},
147.  {'question': "What do you get if you divide one hundred by half?",
148.   'answer': "One hundred divided by half is two hundred. One hundred divided
       ➥ by two is fifty.",
149.   'accept': ['two', '200']},
```

```
150.  {'question': "What do you call someone who always knows where their
      ➥ spouse is?",
151.   'answer': "A widow or widower.",
152.   'accept': ['widow', 'widower']},
153.  {'question': "How can someone take a photo but not be a photographer?",
154.   'answer': "They can be a thief.",
155.   'accept': ['thief', 'steal', 'take', 'literal']},
156.  {'question': "An electric train leaves the windy city of Chicago at 4pm on
      ➥ a Monday heading south at 100 kilometers per hour. Which way does the smoke
      ➥ blow from the smokestack?",
157.   'answer': "Electric trains don't have smokestacks.",
158.   'accept': ["don't", "doesn't", 'not', 'no', 'none']},
159.  {'question': 'What is the only word that rhymes with "orange"?',
160.   'answer': "Orange.",
161.   'accept': ['orange']},
162.  {'question': "Who is the U.S. President if the U.S. Vice President dies?",
163.   'answer': "The current U.S. President.",
164.   'accept': ['president', 'current', 'already']},
165.  {'question': "A doctor gives you three pills with instructions to take one
      ➥ every half-hour. How long will the pills last?",
166.   'answer': "One hour.",
167.   'accept': ['1', 'one']},
168.  {'question': "Where is there an ocean with no water?",
169.   'answer': "On a map.",
170.   'accept': ['map']},
171.  {'question': "What is the size of a rhino but weighs nothing?",
172.   'answer': "A rhino's shadow.",
173.   'accept': ['shadow']},
174.  {'question': "The clerk at a butcher shop is exactly 177 centimeters tall.
        What do they weigh?",
175.   'answer': "The clerk weighs meat.",
176.   'accept': ['meat']}]
177.
178. CORRECT_TEXT = ['Correct!', 'That is right.', "You're right.",
179.                  'You got it.', 'Righto!']
180. INCORRECT_TEXT = ['Incorrect!', "Nope, that isn't it.", 'Nope.',
181.                    'Not quite.', 'You missed it.']
182.
183. print('''Trick Questions, by Al Sweigart al@inventwithpython.com
184.
185. Can you figure out the answers to these trick questions?
186. (Enter QUIT to quit at any time.)
187. ''')
188.
189. input('Press Enter to begin...')
190.
191. random.shuffle(QUESTIONS)
192. score = 0
193.
194. for questionNumber, qa in enumerate(QUESTIONS):  # Основной цикл программы.
195.     print('\n' * 40)  # Очищаем экран.
```

```
196.     print('Question:', questionNumber + 1)
197.     print('Score:', score, '/', len(QUESTIONS))
198.     print('QUESTION:', qa['question'])
199.     response = input('  ANSWER: ').lower()
200.
201.     if response == 'quit':
202.         print('Thanks for playing!')
203.         sys.exit()
204.
205.     correct = False
206.     for acceptanceWord in qa['accept']:
207.         if acceptanceWord in response:
208.             correct = True
209.
210.     if correct:
211.         text = random.choice(CORRECT_TEXT)
212.         print(text, qa['answer'])
213.         score += 1
214.     else:
215.         text = random.choice(INCORRECT_TEXT)
216.         print(text, 'The answer is:', qa['answer'])
217.     response = input('Press Enter for the next question...').lower()
218.
219.     if response == 'quit':
220.         print('Thanks for playing!')
221.         sys.exit()
222.
223. print("That's all the questions. Thanks for playing!")
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Идеи касательно возможных небольших изменений вы найдете в комментариях, помеченных (!).

Исследование программы

Это очень простая программа, так что вариантов ее модификации не так уж много. Вместо этого подумайте, где еще может пригодиться программа вида «вопрос — ответ».

79

Игра «2048»

Веб-разработчик Габриэле Чирулли (Gabriele Cirulli) создал игру «2048» за одни выходные. В ее основе лежит игра «1024» от Veewo Studios, созданная, в свою очередь, под впечатлением от игры Threes! от команды разработчиков Sirvo. В игре «2048» нужно объединять числа на доске 4 × 4, таким образом убирая их с экрана. Две двойки объединяются в четверку, две четверки объединяются в восьмерку и т. д. При каждом объединении на игровую доску добавляется новая двойка. Цель: достичь 2048, прежде чем вся доска заполнится.

Программа в действии

Результат выполнения `twentyfortyeight.py` выглядит следующим образом:

```
Twenty Forty-Eight, by Al Sweigart al@inventwithpython.com
--сокращено--
+-----+-----+-----+-----+
|     |     |     |     |
|     |     |  2  |  16 |
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|     |  16 |  4  |  2  |
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|  2  |     |  4  |  32 |
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|     |     |     |  2  |
|     |     |     |     |
+-----+-----+-----+-----+

Score: 80
Enter move: (WASD or Q to quit)
--сокращено--
```

Описание работы

Поведение этой программы реализуется с помощью структур данных для столбцов, представленных списками из четырех строковых значений: `BLANK` (строковое значение, состоящее из одного пробела), `'2'`, `'4'`, `'8'` и т. д. Первое значение в данном списке соответствует низу столбца, а последнее — верху. Объединяющиеся в столбце числа всегда сдвигаются вниз, а игрок может сдвигать «плитки» вверх, вниз, влево или вправо. Можете представить себе, что сила тяжести тянет «плитки» в этих направлениях. Например, на рис. 79.1 показана доска с «плитками», сдвигающимися вправо. Создаем четыре списка для столбцов:

- `['2', '4', '8', ' ']`
- `[' ', ' ', ' ', '4']`
- `[' ', ' ', ' ', '2']`
- `[' ', ' ', ' ', ' ']`

Функция `combineTilesInColumn()` получает на входе один список для столбца и возвращает другой, в котором все совпадающие числа объединены и сдвинуты вниз.

За создание списков для столбцов в соответствующем направлении и обновление игровой доски на основе возвращаемого списка отвечает вызывающий функцию `combineTilesInColumn()` код.

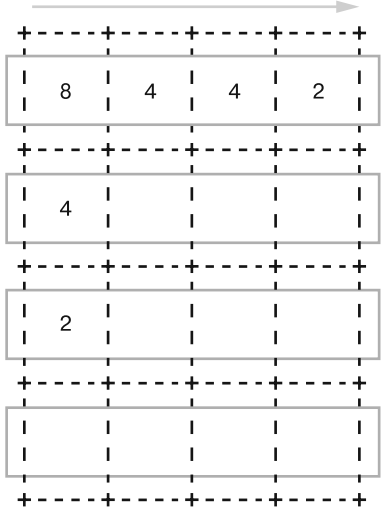


Рис. 79.1. Столбцы (выделены) в случае сдвига вправо

```
"""Игра «2048», (c) Эл Свейгарт al@inventwithpython.com
Игра, в которой при сдвиге "плиток" объединяются числа, растущие
в геометрической прогрессии. На основе игры "2048" Габриэле Чирулли —
клона игры "1024" от Veewo Studios, которая является клоном игры Threes!
Подробнее — в статье https://ru.wikipedia.org/wiki/2048_(игра)
Код размещен на https://nostarch.com/big-book-small-python-projects
Теги: большая, игра, головоломка"""

import random, sys

# Задаем константы:
BLANK = ''  # Значение, соответствующее пустой клетке на доске.


def main():
    print('''Twenty Forty-Eight, by Al Sweigart al@inventwithpython.com

Slide all the tiles on the board in one of four directions. Tiles with
like numbers will combine into larger-numbered tiles. A new 2 tile is
added to the board on each move. You win if you can create a 2048 tile.
You lose if the board fills up the tiles before then.''')
    input('Press Enter to begin...')

```

```
    gameBoard = getNewBoard()

    while True:  # Основной цикл программы.
        drawBoard(gameBoard)
        print('Score:', getScore(gameBoard))
        playerMove = askForPlayerMove()
        gameBoard = makeMove(gameBoard, playerMove)
        addTwoToBoard(gameBoard)

        if isFull(gameBoard):
            drawBoard(gameBoard)
            print('Game Over - Thanks for playing!')
            sys.exit()


def getNewBoard():
    """Возвращает новую структуру данных для доски,
    которая представляет собой ассоциативный массив, ключи которого —
    кортежи (x, y), а значения находятся в соответствующих клетках.
    Эти значения равны либо числам — степеням двойки, либо BLANK.
    Система координат выглядит вот так:
        X0 1 2 3
       Y+-+-+-+-+
       0| | | | |
        +-+-+-+-+
       1| | | | |
        +-+-+-+-+
       2| | | | |
        +-+-+-+-+
       3| | | | |
        +-+-+-+-+"""

    newBoard = {}  # Содержит возвращаемую структуру данных доски.
    # Проходим по всем клеткам доски, устанавливая "плиткам" значение BLANK:
    for x in range(4):
        for y in range(4):
            newBoard[(x, y)] = BLANK

    # Выбираем две случайные клетки для двух начальных двоек:
    startingTwosPlaced = 0  # Число выбранных изначально клеток.
    while startingTwosPlaced < 2:  # Повторяем, чтобы продублировать клетки.
        randomSpace = (random.randint(0, 3), random.randint(0, 3))
        # Проверяем, что случайно выбранная клетка еще не занята:
        if newBoard[randomSpace] == BLANK:
            newBoard[randomSpace] = 2
            startingTwosPlaced = startingTwosPlaced + 1

    return newBoard


def drawBoard(board):
```

```
    """Отрисовываем структуру данных для доски на экране."""

    # Проходим по всем возможным клеткам слева направо, сверху вниз
    # и создаем список меток всех клеток.
    labels = []  # Список строковых значений для числа/BLANK данной "плитки".
    for y in range(4):
        for x in range(4):
            tile = board[(x, y)]  # Получаем значение "плитки" в этой клетке.
            # Убеждаемся, что длина метки равна 5:
            labelForThisTile = str(tile).center(5)
            labels.append(labelForThisTile)

    # {} заменяются метками соответствующих "плиток":
    print("""
+-----+-----+-----+-----+
|     |     |     |     |
|{}|{}|{}|{}|
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|{}|{}|{}|{}|
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|{}|{}|{}|{}|
|     |     |     |     |
+-----+-----+-----+-----+
|     |     |     |     |
|{}|{}|{}|{}|
|     |     |     |     |
+-----+-----+-----+-----+
""".format(*labels))


def getScore(board):
    """Возвращает сумму всех "плиток" в структуре данных для доски."""
    score = 0
    # Проходим в цикле по всем клеткам, прибавляя "плитки" к score:
    for x in range(4):
        for y in range(4):
            # Прибавляем к score только непустые "плитки":
            if board[(x, y)] != BLANK:
                score = score + board[(x, y)]
    return score


def combineTilesInColumn(column):
    """column представляет собой список из четырех "плиток". Индекс 0
    соответствует низу столбца column, а сила тяжести тянет "плитки" вниз,
    в случае равных значений они объединяются. Например, combineTilesInCo-
    lumn([2, BLANK, 2, BLANK]) возвращает [4, BLANK, BLANK, BLANK]."""
```

```
126.
127.     # Копируем в combinedTiles только числа (не BLANK) из column
128.     combinedTiles = []  # Список непустых "плиток" в column.
129.     for i in range(4):
130.         if column[i] != BLANK:
131.             combinedTiles.append(column[i])
132.     # Продолжаем присоединять к списку значения BLANK, пока не получится
133.     # четыре "плитки":
134.     while len(combinedTiles) < 4:
135.         combinedTiles.append(BLANK)
136.
137.     # Объединяем числа, если число сверху — такое же, и удваиваем.
138.     for i in range(3):  # Пропускаем индекс 3: это верхняя клетка.
139.         if combinedTiles[i] == combinedTiles[i + 1]:
140.             combinedTiles[i] *= 2  # Удваиваем число на "плитке".
141.             # Сдвигаем расположенные вверху "плитки" на одну клетку:
142.             for aboveIndex in range(i + 1, 3):
143.                 combinedTiles[aboveIndex] = combinedTiles[aboveIndex + 1]
144.             combinedTiles[3] = BLANK  # Самая верхняя клетка — всегда пустая.
145.     return combinedTiles
146.
147.
148. def makeMove(board, move):
149.     """Производит ход на доске.
150.
151.     Аргумент move — 'W', 'A', 'S' или 'D'. Функция возвращает
152.     получившуюся структуру данных для доски (board)."""
153.
154.     # board разбивается на четыре столбца, различные
155.     # в зависимости от направления хода:
156.     if move == 'W':
157.         allColumnsSpaces = [[(0, 0), (0, 1), (0, 2), (0, 3)],
158.                             [(1, 0), (1, 1), (1, 2), (1, 3)],
159.                             [(2, 0), (2, 1), (2, 2), (2, 3)],
160.                             [(3, 0), (3, 1), (3, 2), (3, 3)]]
161.     elif move == 'A':
162.         allColumnsSpaces = [[(0, 0), (1, 0), (2, 0), (3, 0)],
163.                             [(0, 1), (1, 1), (2, 1), (3, 1)],
164.                             [(0, 2), (1, 2), (2, 2), (3, 2)],
165.                             [(0, 3), (1, 3), (2, 3), (3, 3)]]
166.     elif move == 'S':
167.         allColumnsSpaces = [[(0, 3), (0, 2), (0, 1), (0, 0)],
168.                             [(1, 3), (1, 2), (1, 1), (1, 0)],
169.                             [(2, 3), (2, 2), (2, 1), (2, 0)],
170.                             [(3, 3), (3, 2), (3, 1), (3, 0)]]
171.     elif move == 'D':
172.         allColumnsSpaces = [[(3, 0), (2, 0), (1, 0), (0, 0)],
173.                             [(3, 1), (2, 1), (1, 1), (0, 1)],
174.                             [(3, 2), (2, 2), (1, 2), (0, 2)],
175.                             [(3, 3), (2, 3), (1, 3), (0, 3)]]
176.     # Структура данных board после выполнения хода:
```

```
177.     boardAfterMove = {}
178.     for columnSpaces in allColumnsSpaces:  # Проходим в цикле по всем
179.                                            # четырем столбцам.
180.         # Получаем "плитки" для этого столбца (первая "плитка"
181.         # соответствует "низу" столбца):
182.         firstTileSpace = columnSpaces[0]
183.         secondTileSpace = columnSpaces[1]
184.         thirdTileSpace = columnSpaces[2]
185.         fourthTileSpace = columnSpaces[3]
186.
187.         firstTile = board[firstTileSpace]
188.         secondTile = board[secondTileSpace]
189.         thirdTile = board[thirdTileSpace]
190.         fourthTile = board[fourthTileSpace]
191.
192.         # Формируем столбец и объединяем "плитки" в нем:
193.         column = [firstTile, secondTile, thirdTile, fourthTile]
194.         combinedTilesColumn = combineTilesInColumn(column)
195.
196.         # Формируем новую структуру данных для доски
             # с объединенными "плитками":
197.         boardAfterMove[firstTileSpace] = combinedTilesColumn[0]
198.         boardAfterMove[secondTileSpace] = combinedTilesColumn[1]
199.         boardAfterMove[thirdTileSpace] = combinedTilesColumn[2]
200.         boardAfterMove[fourthTileSpace] = combinedTilesColumn[3]
201.
202.     return boardAfterMove
203.
204.
205. def askForPlayerMove():
206.     """Просим игрока указать направление следующего хода (или выйти).
207.
208.     Проверяем ход на допустимость: 'W', 'A', 'S' или 'D'."""
209.     print('Enter move: (WASD or Q to quit)')
210.     while True:  # Выполняем цикл, пока не будет введен допустимый ход.
211.         move = input('> ').upper()
212.         if move == 'Q':
213.             # Завершаем программу:
214.             print('Thanks for playing!')
215.             sys.exit()
216.
217.         # Возвращаем допустимый ход или выполняем еще итерацию
             # и спрашиваем снова:
218.         if move in ('W', 'A', 'S', 'D'):
219.             return move
220.         else:
221.             print('Enter one of "W", "A", "S", "D", or "Q".')
222.
223.
224. def addTwoToBoard(board):
225.     """Добавляет на доску две случайные новые "плитки"."""
```

```
226.     while True:
227.         randomSpace = (random.randint(0, 3), random.randint(0, 3))
228.         if board[randomSpace] == BLANK:
229.             board[randomSpace] = 2
230.             return  # Выполняем возврат из функции после обнаружения
231.                     # непустой "плитки".
232.
233. def isFull(board):
234.     """Возвращает True, если в структуре данных для доски нет пустых клеток."""
235.     # Проходим в цикле по всем клеткам доски:
236.     for x in range(4):
237.         for y in range(4):
238.             # Если клетка пуста, возвращаем False:
239.             if board[(x, y)] == BLANK:
240.                 return False
241.     return True  # Пустых клеток нет, возвращаем True.
242.
243.
244. # Если программа не импортируется, а запускается, выполняем запуск игры:
245. if __name__ == '__main__':
246.     try:
247.         main()
248.     except KeyboardInterrupt:
249.         sys.exit()  # Если нажато Ctrl+C — завершаем программу.
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `return score` в строке 118 заменить на `return 9999`?
2. Что будет, если выражение `board[randomSpace] = 2` в строке 229 заменить на `board[randomSpace] = 256`?

80

Шифр Виженера

Шифр Виженера, ошибочно приписываемый криптографу XIX века Блезу де Виженеру (Blaise de Vigenère) (другие исследователи независимо от Виженера открыли его раньше), в течение нескольких столетий не поддавался взлому. По существу, он представляет собой шифр Цезаря, только с состоящим из нескольких частей ключом. Так называемый *ключ Виженера* представляет собой слово или даже случайную последовательность букв. Каждой букве соответствует число, показывающее, на сколько необходимо сдвинуть эту букву в сообщении: A соответствует сдвигу буквы в сообщении на 0 букв, B — на 1, C — на 2 и т. д.

Например, если ключ Виженера — слово CAT, C означает сдвиг на 2 буквы, A — на 0, а T — на 19. Первая буква сообщения сдвигается на 2 буквы, вторая не сдвигается, а третья — сдвигается на 19. Для четвертой буквы просто используется ключ для первой, равный 2.

Именно использование нескольких ключей шифра Цезаря обеспечивает криптостойкость шифра Виженера. Возможное количество комбинаций слишком велико для прямого перебора. В то же время шифр Виженера устойчив к частотному анализу — слабой стороне шифра простой подстановки. Многие столетия шифр Виженера был последним словом в криптографии.

Вы увидите, что программы для шифра Виженера и шифра Цезаря во многом похожи. Подробнее о шифре Виженера можно прочитать в статье «Википедии»: https://

ru.wikipedia.org/wiki/Шифр_Виженера. Если вам хотелось бы узнать больше о шифрах и методах их взлома — можете прочитать мою книгу *Cracking Codes with Python*.

Программа в действии

Результат выполнения `vigenere.py` выглядит следующим образом:

```
Vigenère Cipher, by Al Sweigart al@inventwithpython.com
The Vigenère cipher is a polyalphabetic substitution cipher that was
powerful enough to remain unbroken for centuries.
Do you want to (e)ncrypt or (d)ecrypt?
> e
Please specify the key to use.
It can be a word or any combination of letters:
> PIZZA
Enter the message to encrypt.
> Meet me by the rose bushes tonight.
Encrypted message:
Bmds mt jx sht znre qcrgeh bnmivps.
Full encrypted text copied to clipboard.
```

Описание работы

Поскольку процессы шифрования и расшифровки очень похожи, и за тот и за другой отвечает функция `translateMessage()`. Функции `encryptMessage()` и `decryptMessage()` — всего лишь *адаптеры* для `translateMessage()`. Другими словами, это функции, адаптирующие свои аргументы, передающие их в другую функцию, а затем возвращающие то, что вернула эта другая функция. Эти функции-адаптеры вызываются в нашей программе примерно так, как вызывались `encryptMessage()` и `decryptMessage()` в проекте 66. Можете импортировать эти проекты в качестве модулей в других программах и использовать их код шифрования/расшифровки, не прибегая к копированию/вставке кода непосредственно в новую программу.

```
 1. """Шифр Виженера, (c) Эл Свейгарт al@inventwithpython.com
 2. Шифр Виженера — многоалфавитный шифр подстановки, настолько
 3. эффективный, что его не могли взломать многие столетия.
 4. Подробнее — в статье https://ru.wikipedia.org/wiki/Шифр_Виженера
 5. Код размещен на https://nostarch.com/big-book-small-python-projects
 6. Теги: короткая, криптография, математическая"""
 7.
 8. try:
 9.     import pyperclip  # pyperclip копирует текст в буфер обмена.
10. except ImportError:
11.     pass  # Если pyperclip не установлена, ничего не делаем. Не проблема.
```

```
12.
13. # Все возможные символы для шифрования/дешифровки:
14. LETTERS = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
15.
16.
17. def main():
18.     print('''Vigenère Cipher, by Al Sweigart al@inventwithpython.com
19. The Vigenère cipher is a polyalphabetic substitution cipher that was
20. powerful enough to remain unbroken for centuries.''')
21.
22.     # Спрашиваем у пользователя, хочет он шифровать или расшифровывать:
23.     while True:  # Повторяем вопрос, пока пользователь не введет e или d.
24.         print('Do you want to (e)ncrypt or (d)ecrypt?')
25.         response = input('> ').lower()
26.         if response.startswith('e'):
27.             myMode = 'encrypt'
28.             break
29.         elif response.startswith('d'):
30.             myMode = 'decrypt'
31.             break
32.         print('Please enter the letter e or d.')
33.
34.     # Просим пользователя ввести ключ шифрования:
35.     while True: # Повторяем вопрос, пока пользователь не введет допустимый ключ
36.         print('Please specify the key to use.')
37.         print('It can be a word or any combination of letters:')
38.         response = input('> ').upper()
39.         if response.isalpha():
40.             myKey = response
41.             break
42.
43.     # Просим пользователя ввести сообщение для шифрования/расшифровки:
44.     print('Enter the message to {}.'.format(myMode))
45.     myMessage = input('> ')
46.
47.     # Производим шифрование/расшифровку:
48.     if myMode == 'encrypt':
49.         translated = encryptMessage(myMessage, myKey)
50.     elif myMode == 'decrypt':
51.         translated = decryptMessage(myMessage, myKey)
52.
53.     print('%sed message:' % (myMode.title()))
54.     print(translated)
55.
56.     try:
57.         pyperclip.copy(translated)
58.         print('Full %sed text copied to clipboard.' % (myMode))
59.     except:
60.         pass  # Если pyperclip не установлена, ничего не делаем.
61.
```

```
 62.
 63. def encryptMessage(message, key):
 64.     """Шифрует сообщение message в соответствии с ключом key."""
 65.     return translateMessage(message, key, 'encrypt')
 66.
 67.
 68. def decryptMessage(message, key):
 69.     """Расшифровывает сообщение message в соответствии с ключом key."""
 70.     return translateMessage(message, key, 'decrypt')
 71.
 72.
 73. def translateMessage(message, key, mode):
 74.     """Зашифровывает или расшифровывает сообщение в соответствии с ключом."""
 75.     translated = []  # Для хранения строкового значения
 76.                      # зашифрованного/расшифрованного сообщения.
 77.     keyIndex = 0
 78.     key = key.upper()
 79.
 80.     for symbol in message:  # Проходим в цикле по всем символам сообщения.
 81.         num = LETTERS.find(symbol.upper())
 82.         if num != -1:  # -1 означает, что symbol.upper() не входит в LETTERS.
 83.             if mode == 'encrypt':
 84.                 # Прибавляем при шифровании:
 85.                 num += LETTERS.find(key[keyIndex])
 86.             elif mode == 'decrypt':
 87.                 # Вычитаем при расшифровке:
 88.                 num -= LETTERS.find(key[keyIndex])
 89.
 90.             num %= len(LETTERS)  # Учитываем возможный переход по кругу.
 91.
 92.             #  Добавляем зашифрованный/расшифрованный символ в translated.
 93.             if symbol.isupper():
 94.                 translated.append(LETTERS[num])
 95.             elif symbol.islower():
 96.                 translated.append(LETTERS[num].lower())
 97.
 98.             keyIndex += 1  # Переходим к следующей букве ключа.
 99.             if keyIndex == len(key):
100.                 keyIndex = 0
101.         else:
102.             # Просто добавляем символ без шифрования/расшифровки:
103.             translated.append(symbol)
104.
105.     return ''.join(translated)
106.
107.
108. # Если программа не импортируется, а запускается, производим запуск:
109. if __name__ == '__main__':
110.     main()
```

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что происходит при шифровании с ключом `'A'`?
2. Какую ошибку вызовет удаление или комментирование `myKey = response` в строке 40?

81

Головоломка с ведрами воды

В этой головоломке, рассчитанной на одного игрока, необходимо с помощью трех ведер (на три, пять и восемь литров) получить в одном из ведер ровно четыре литра воды. Ведра можно только выливать, полностью наполнять или переливать из одного в другое. Например, можете наполнить пятилитровое ведро и вылить его содержимое в трехлитровое, в результате чего у вас окажется полное трехлитровое ведро и два литра воды в пятилитровом.

Немного усилий — и вы решите эту головоломку. Но как решить ее с помощью минимального количества ходов?

Программа в действии

Результат выполнения `waterbucket.py` выглядит следующим образом:

```
Water Bucket Puzzle, by Al Sweigart al@inventwithpython.com

Try to get 4L of water into one of these
buckets:

8|      |
7|      |
6|      |
5|      |  5|      |
4|      |  4|      |
3|      |  3|      |  3|      |
2|      |  2|      |  2|      |
1|      |  1|      |  1|      |
 +------+   +------+   +------+
    8L         5L         3L

You can:
  (F)ill the bucket
  (E)mpty the bucket
  (P)our one bucket into another
  (Q)uit
> f
Select a bucket 8, 5, 3, or QUIT:
> 5

Try to get 4L of water into one of these
buckets:

8|      |
7|      |
6|      |
5|      |  5|WWWWWW|
4|      |  4|WWWWWW|
3|      |  3|WWWWWW|  3|      |
2|      |  2|WWWWWW|  2|      |
1|      |  1|WWWWWW|  1|      |
 +------+   +------+   +------+
    8L         5L         3L
--сокращено--
```

Описание работы

В переменной `waterInBucket` содержится ассоциативный массив, описывающий состояние ведер воды. Ключами этого ассоциативного массива служат строковые

значения `'8'`, `'5'` и `'3'`, соответствующие ведрам, а значения — целочисленные (и описывают количество литров воды в соответствующем ведре).

В строках с 48-й по 59-ю этот ассоциативный массив используется для визуализации ведер и воды на экране. Содержащий значения либо `'WWWWWW'` (представляющие воду), либо `'  '` (представляющие воздух) список `waterDisplay` передается в строковый метод `format()`. Первые восемь строковых значений в списке `waterDisplay` служат для заполнения восьмилитрового ведра, следующие пять — пятилитрового, а последние три — трехлитрового.

```
  1. """Головоломка с ведрами воды, (c) Эл Свейгарт al@inventwithpython.com
  2. Головоломка с переливанием воды.
  3. Подробнее — в статье https://en.wikipedia.org/wiki/Water_pouring_puzzle
  4. Код размещен на https://nostarch.com/big-book-small-python-projects
  5. Теги: """
  6.
  7. import sys
  8.
  9. print('Water Bucket Puzzle, by Al Sweigart al@inventwithpython.com')
 10.
 11. GOAL = 4  # Точный объем воды, необходимый для победы.
 12. steps = 0  # Содержит количество ходов, выполненных игроком для решения
 13.            # головоломки.
 14.
 15. # Объем воды в каждом из ведер:
 16. waterInBucket = {'8': 0, '5': 0, '3': 0}
 17.
 18. while True:  # Основной цикл игры.
 19.     # Отображаем текущее состояние ведер:
 20.     print()
 21.     print('Try to get ' + str(GOAL) + 'L of water into one of these')
 22.     print('buckets:')
 23.
 24.     waterDisplay = []  # Содержит строковые значения для воды и пустого места.
 25.
 26.     # Получаем строковые значения для восьмилитрового ведра:
 27.     for i in range(1, 9):
 28.         if waterInBucket['8'] < i:
 29.             waterDisplay.append('      ')  # Добавляем пустое место.
 30.         else:
 31.             waterDisplay.append('WWWWWW')  # Добавляем воду.
 32.
 33.     # Получаем строковые значения для пятилитрового ведра:
 34.     for i in range(1, 6):
 35.         if waterInBucket['5'] < i:
 36.             waterDisplay.append('      ')  # Добавляем пустое место.
 37.         else:
 38.             waterDisplay.append('WWWWWW')  # Добавляем воду.
 39.
 40.     # Получаем строковые значения для трехлитрового ведра:
```

```
    for i in range(1, 4):
        if waterInBucket['3'] < i:
            waterDisplay.append('      ')  # Добавляем пустое место.
        else:
            waterDisplay.append('WWWWWW')  # Добавляем воду.

    # Отображаем на экране ведра, каждое со своим объемом воды:
    print('''
8|{7}|
7|{6}|
6|{5}|
5|{4}|  5|{12}|
4|{3}|  4|{11}|
3|{2}|  3|{10}|  3|{15}|
2|{1}|  2|{9}|  2|{14}|
1|{0}|  1|{8}|  1|{13}|
 +------+   +------+   +------+
    8L         5L         3L
'''.format(*waterDisplay))

    # Проверяем, не содержится ли в каком-то из ведер нужное количество воды:
    for waterAmount in waterInBucket.values():
        if waterAmount == GOAL:
            print('Good job! You solved it in', steps, 'steps!')
            sys.exit()

    # Спрашиваем у игрока, какое действие он хочет произвести с ведром:
    print('You can:')
    print('  (F)ill the bucket')
    print('  (E)mpty the bucket')
    print('  (P)our one bucket into another')
    print('  (Q)uit')
    while True:  # Спрашиваем, пока пользователь не укажет допустимое
                 # действие.
        move = input('> ').upper()
        if move == 'QUIT' or move == 'Q':
            print('Thanks for playing!')
            sys.exit()

        if move in ('F', 'E', 'P'):
            break  # Игрок выбрал допустимое действие.
        print('Enter F, E, P, or Q')

    # Предлагаем игроку выбрать ведро:
    while True:  # Спрашиваем, пока не будет указано допустимое ведро.
        print('Select a bucket 8, 5, 3, or QUIT:')
        srcBucket = input('> ').upper()

        if srcBucket == 'QUIT':
            print('Thanks for playing!')
            sys.exit()
```

```
 92.
 93.         if srcBucket in ('8', '5', '3'):
 94.             break  # Игрок выбрал допустимое ведро.
 95.
 96.     # Производим выбранное действие:
 97.     if move == 'F':
 98.         # Устанавливаем количество воды в максимальное значение.
 99.         srcBucketSize = int(srcBucket)
100.         waterInBucket[srcBucket] = srcBucketSize
101.         steps += 1
102.
103.     elif move == 'E':
104.         waterInBucket[srcBucket] = 0  # Устанавливаем количество воды
105.                                       # в нулевое значение.
106.         steps += 1
107.     elif move == 'P':
108.         # Предлагаем игроку выбрать ведро, в которое наливать воду:
109.         while True:  # Спрашиваем, пока не будет указано допустимое ведро.
110.             print('Select a bucket to pour into: 8, 5, or 3')
111.             dstBucket = input('> ').upper()
112.             if dstBucket in ('8', '5', '3'):
113.                 break  # Игрок выбрал допустимое ведро.
114.
115.         # Определяем наливаемый объем воды:
116.         dstBucketSize = int(dstBucket)
117.         emptySpaceInDstBucket = dstBucketSize - waterInBucket[dstBucket]
118.         waterInSrcBucket = waterInBucket[srcBucket]
119.         amountToPour = min(emptySpaceInDstBucket, waterInSrcBucket)
120.
121.         # Выливаем воду из ведра:
122.         waterInBucket[srcBucket] -= amountToPour
123.
124.         # Наливаем вылитую воду в другое ведро:
125.         waterInBucket[dstBucket] += amountToPour
126.         steps += 1
127.
128.     elif move == 'C':
129.         pass  # Если игрок выбрал Cancel, ничего не делаем.
```

Когда вы введете исходный код и запустите его несколько раз, попробуйте поэкспериментировать с внесением в него изменений. Можете также сами попробовать придумать, как сделать следующее:

- разнообразить игру, добавив настройки для изменения размеров трех ведер и целевого объема воды;
- добавить «подсказку», при которой программа исследует количество воды в ведрах и подбирает рекомендуемый следующий шаг. Если программа не может вычислить следующий шаг, то просто выводит надпись «Не знаю, что вы можете сделать дальше. Может, просто начать игру сначала?».

Исследование программы

Попробуйте найти ответы на следующие вопросы. Поэкспериментируйте с изменениями кода и запустите программу снова, чтобы увидеть, как они повлияют на ее работу.

1. Что будет, если `waterInBucket[srcBucket] = 0` в строке 104 заменить на `waterInBucket[srcBucket] = 1`?
2. Что будет, если `{'8': 0, '5': 0, '3': 0}` в строке 16 заменить на `{'8': 0, '5': 4, '3': 0}`?
3. Что будет, если `{'8': 0, '5': 0, '3': 0}` в строке 16 заменить на `{'8': 9, '5': 0, '3': 0}`?

A

Указатель тегов

Проекты в этой книге снабжены тегами, описывающими тип соответствующей программы. Первый тег описывает ее размер: крошечная (от 1 до 63 строк), короткая (от 64 до 127 строк), большая (от 128 до 255 строк) и очень большая (256 строк и более). Ниже представлен список проектов по тегам размера:

- **крошечная:** #3 Сообщение в виде битовой карты, #7 Взлом шифра Цезаря, #12 Гипотеза Коллатца, #14 Обратный отсчет, #15 Глубокая пещера, #16 Ромбы, #19 Цифровые часы, #20 Цифровой поток, #24 Разложение на множители, #25 Быстрый стрелок, #31 Угадай число, #32 Простак, #35 Гексагональная сетка, #40 П0г0в0рим (leetspeak), #42 Магический хрустальный шар, #46 Моделирование статистики за миллион бросков игральных костей, #49 Таблица умножения, #50 Девяносто девять бутылок, #52 Счет в различных системах счисления, #56 Простые числа, #57 Индикатор хода выполнения, #58 Радуга, #60 Камень, ножницы, бумага (беспроигрышная версия), #61 Шифр ROT13, #65 Ковер из «Сияния», #67 Синусовидное сообщение, #72 Губкорегистр, #74 Преобразование текста в речь;
- **короткая:** #1 Бейглз, #2 Парадокс дней рождения, #5 Отскакивающий от краев логотип DVD, #6 Шифр Цезаря, #8 Генерация календарей, #10 Чо-хан, #13 Игра «Жизнь» Конвея, #18 Выбрасыватель игральных костей, #21 Визуализация ДНК, #26 Фибоначчи, #29 Моделирование лесного пожара, #51 ДевяНосто деевяять буутылок, #53 Периодическая таблица элементов,

#54 Поросячья латынь, #55 Лотерея Powerball, #59 Камень, ножницы, бумага, #64 Семисегментный модуль индикации, #66 Простой шифр подстановки, #69 Бега улиток, #71 Повторение музыки, #76 Крестики-нолики, #77 Ханойская башня, #80 Шифр Виженера;

- **большая:** #4 Блек-джек, #9 Морковка в коробке, #11 Генератор заголовков-приманок, #17 Арифметика с игральными костями, #22 Утята, #23 Гравировщик, #28 Заливка, #30 Четыре в ряд, #33 Мини-игра со взломом, #34 «Виселица» и «гильотина», #36 Песочные часы, #37 Голодные роботы, #39 Муравей Лэнгтона, #41 Под счастливой звездой, #43 Манкала, #44 Бегущий в лабиринте 2D, #47 Генератор картин в стиле Мондриана, #48 Парадокс Монти Холла, #62 Вращающийся куб, #63 Царская игра Ура, #68 Игра в 15, #70 Соробан — японский абак, #73Головоломка судоку, #75 Три карты Монте, #78 Вопросы с подвохом, #79 Игра «2048», #81 Головоломка с ведрами воды;
- **очень большая:** #27 Аквариум, #38 «Я обвиняю!», #45 Бегущий в лабиринте 3D.

Остальные теги описывают различные особенности программы:

- **bext:** #5 Отскакивающий от краев логотип DVD, #27 Аквариум, #28 Заливка, #29 Моделирование лесного пожара, #36 Песочные часы, #39 Муравей Лэнгтона, #47 Генератор картин в стиле Мондриана, #58 Радуга;
- **головоломка:** #1 Бейглз, #33 Мини-игра со взломом, #34 «Виселица» и «гильотина», #38 «Я обвиняю!», #68 Игра в 15, #73 Головоломка судоку, #77 Ханойская башня, #79 2048, #81 Головоломка с ведрами воды;
- **графика:** #3 Сообщение в виде битовой карты, #5 Отскакивающий от краев логотип DVD, #13 Игра «Жизнь» Конвея, #14 Обратный отсчет, #15 Глубокая пещера, #16 Ромбы, #19 Цифровые часы, #20 Цифровой поток, #21 Визуализация ДНК, #22 Утята, #23 Гравировщик, #27 Аквариум, #33 Мини-игра со взломом, #35 Гексагональная сетка, #36 Песочные часы, #39 Муравей Лэнгтона, #45 Бегущий в лабиринте 3D, #47 Генератор картин в стиле Мондриана, #58 Радуга, #62 Вращающийся куб, #65 Ковер из «Сияния»,#67 Синусовидное сообщение, #69 Бега улиток, #70 Соробан — японский абак;
- **для двух игроков:** #9 Морковка в коробке, #30 Четыре в ряд, #43 Манкала, #63 Царская игра Ура, #76 Крестики-нолики;
- **для начинающих:** #6 Шифр Цезаря, #7 Взлом шифра Цезаря, #9 Морковка в коробке, #10 Чо-хан, #11 Генератор заголовков-приманок, #12 Гипотеза Коллатца, #15 Глубокая пещера, #16 Ромбы, #20 Цифровой поток, #24 Разложение на множители, #25 Быстрый стрелок, #31 Угадай число, #32 Простак, #35 Гексагональная сетка, #40 П0г0в0рим (leetspeak), #42 Магический хрустальный шар, #46 Моделирование статистики за миллион бросков

игральных костей, #49 Таблица умножения, #50 Девяносто девять бутылок, #58 Радуга, #65 Ковер из «Сияния», #69 Бега улиток, #71 Повторение музыки, #72 Губкорегистр, #74 Преобразование текста в речь;

- **игра:** #1 Бейглз, #4 Блек-джек, #9 Морковка в коробке, #10 Чо-хан, #17 Арифметика с игральными костями, #25 Быстрый стрелок, #28 Заливка, #30 Четыре в ряд, #31 Угадай число, #33 Мини-игра со взломом, #34 «Виселица» и «гильотина», #37 Голодные роботы, #38 «Я обвиняю!», #41 Под счастливой звездой, #43 Манкала, #44 Бегущий в лабиринте 2D, #45 Бегущий в лабиринте 3D, #48 Парадокс Монти Холла, #59 Камень, ножницы, бумага, #60 Камень, ножницы, бумага (беспроигрышная версия), #63 Царская игра Ура, #68 Игра в 15, #69 Бега улиток, #71 Повторение музыки, #73 Головоломка судоку, #75 Три карты Монте, #76 Крестики-нолики, #77 Ханойская башня, #79 2048, #81 Головоломка с ведрами воды;
- **имитационное моделирование:** #2 Парадокс дней рождения, #13 Игра «Жизнь» Конвея, #18 Выбрасыватель игральных костей, #29 Моделирование лесного пожара, #36 Песочные часы, #39 Муравей Лэнгтона, #46 Моделирование статистики за миллион бросков игральных костей, #48 Парадокс Монти Холла, #55 Лотерея Powerball, #70 Соробан — японский абак;
- **карточная игра:** #4 Блек-джек, #75 Три карты Монте;
- **криптография:** #6 Шифр Цезаря, #7 Взлом шифра Цезаря, #61 Шифр ROT13, #66 Простой шифр подстановки, #80 Шифр Виженера;
- **лабиринт:** #44 Бегущий в лабиринте 2D, #45 Бегущий в лабиринте 3D;
- **математическая:** #2 Парадокс дней рождения, #6 Шифр Цезаря, #7 Взлом шифра Цезаря, #12 Гипотеза Коллатца, #17 Арифметика с игральными костями, #24 Разложение на множители, #26 Фибоначчи, #46 Моделирование статистики за миллион бросков игральных костей, #48 Парадокс Монти Холла, #49 Таблица умножения, #52 Счет в различных системах счисления, #56 Простые числа, #62 Вращающийся куб, #66 Простой шифр подстановки, #70 Соробан — японский абак, #80 Шифр Виженера, #81 Головоломка с ведрами воды;
- **многопользовательская:** #41 Под счастливой звездой, #69 Бега улиток;
- **модуль:** #57 Индикатор хода выполнения, #64 Семисегментный модуль индикации;
- **настольная игра:** #30 Четыре в ряд, #43 Манкала, #63 Царская игра Ура, #76 Крестики-нолики;
- **научная:** #21 Визуализация ДНК, #53 Периодическая таблица элементов;
- **объектно-ориентированная:** #22 Утята, #73 Головоломка судоку;

- **прокрутка:** #15 Глубокая пещера, #20 Цифровой поток, #21 Визуализация ДНК, #22 Утята, #50 Девяносто девять бутылок, #51 ДевяНосто деевяять буутылок, #56 Простые числа, #58 Радуга;
- **слова:** #11 Генератор заголовков-приманок, #34 «Виселица» и «Гильотина», #40 П0г0в0рим (leetspeak), #51 ДевяНосто деевяять буутылок, #54 Поросячья латынь, #72 Губкорегистр;
- **юмор:** #11 Генератор заголовков-приманок, #32 Простак, #38 «Я обвиняю!», #42 Магический хрустальный шар, #55 Лотерея Powerball, #60 Камень, ножницы, бумага (беспроигрышная версия), #78 Вопросы с подвохом.

Б

Таблица кодов символов

Функция `print()` позволяет с легкостью вывести на экран любой символ, который только можно набрать на клавиатуре. Однако существует много других символов, которые может понадобиться отобразить на экране: символы мастей карт (червы, бубны, трефы, пики); линии; затененные прямоугольники; стрелки; музыкальные ноты и т. д. Строковые значения для этих символов можно получить, передав их числовой код, называемый кодом символа Unicode, функции `chr()`. Текст хранится в компьютерах в виде последовательности чисел, и каждому символу соответствует свое число. В этом приложении вы найдете список подобных кодов символов.

Использование функций chr() и ord()

Встроенная функция `chr()` языка Python принимает аргумент типа `integer` и возвращает строковое представление соответствующего этому числу символа. Функция `ord()` производит противоположное действие: принимает строковый аргумент, содержащий отдельный символ, и возвращает соответствующее данному символу число — его код в соответствии со стандартом Unicode.

Например, введите в интерактивной командной оболочке следующие команды:

```
>>> chr(65)
'A'
>>> ord('A')
```

```
65
>>> chr(66)
'B'
>>> chr(9829)
'♥'
```

Не все числа являются допустимыми кодами символов, подходящих для выведения на экран. Спектр символов, которые можно отобразить в окне терминала, куда выводятся текстовые результаты работы программ, ограничен. Выводимый программой символ должен также поддерживаться шрифтом, используемым окном терминала. Вместо любых символов, которые оно не может отобразить, окно терминала выводит на экран замещающий символ, ❶.

Диапазон отображаемых окнами терминала Windows символов еще более ограничен. Данный набор символов известен под названием Windows Glyph List 4, он приведен в этом приложении и в статье «Википедии»: https://ru.wikipedia.org/wiki/Windows_Glyph_List_4.

Коды символов часто указываются в виде шестнадцатеричных чисел, а не привычных нам десятичных. В шестнадцатеричных числах, помимо десятичных цифр от `0` до `9`, содержатся еще и буквы от `A` до `F`. Шестнадцатеричные числа часто записываются с префиксом `0x`, который и означает, что следующее за ним число — шестнадцатеричное.

Преобразовать десятичное целое в строковое значение соответствующего шестнадцатеричного числа можно с помощью функции `hex()`. Преобразовать строковое представление шестнадцатеричного числа в десятичное целое можно с помощью функции `int()`, передав ей в качестве второго аргумента `16`. Например, введите в интерактивной командной оболочке следующее:

```
>>> hex(9)
'0x9'
>>> hex(10)
'0xa'
>>> hex(15)
'0xf'
>>> hex(16)
'0x10'
>>> hex(17)
'0x11'
>>> int('0x11', 16)
17
>>> int('11', 16)
17
```

При вызове функции `chr()` необходимо передавать в качестве аргумента десятичное целое, а не шестнадцатеричное строковое значение.

Таблица кодов символов

Ниже приведены все коды символов Unicode из набора, известного как Windows Glyph List 4, то есть символов, поддерживаемых программой терминала Windows, CMD. И macOS, и Linux способны отображать больше символов, чем представлено в данном списке, но ради совместимости ваших программ на языке Python я рекомендую ограничиться символами из этой таблицы.

32 <пробел>	56 8	80 P	104 h	162 ¢
33 !	57 9	81 Q	105 i	163 £
34 «	58 :	82 R	106 j	164 ¤
35 #	59 ;	83 S	107 k	165 ¥
36 $	60 <	84 T	108 l	166 ¦
37 %	61 =	85 U	109 m	167 §
38 &	62 >	86 V	110 n	168 ¨
39 '	63 ?	87 W	111 o	169 ©
40 (	64 @	88 X	112 p	170 ª
41)	65 A	89 Y	113 q	171 «
42 *	66 B	90 Z	114 r	172 ¬
43 +	67 C	91 [	115 s	173 -
44 ,	68 D	92 \	116 t	174 ®
45 -	69 E	93]	117 u	175 ¯
46 .	70 F	94 ^	118 v	176 °
47 /	71 G	95 _	119 w	177 ±
48 0	72 H	96 `	120 x	178 ²
49 1	73 I	97 a	121 y	179 ³
50 2	74 J	98 b	122 z	180 ´
51 3	75 K	99 c	123 {	181 µ
52 4	76 L	100 d	124 |	182 ¶
53 5	77 M	101 e	125 }	183 ·
54 6	78 N	102 f	126 ~	184 ¸
55 7	79 O	103 g	161 ¡	185 ¹

186 º
187 »
188 ¼
189 ½
190 ¾
191 ¿
192 À
193 Á
194 Â
195 Ã
196 Ä
197 Å
198 Æ
199 Ç
200 È
201 É
202 Ê
203 Ë
204 Ì
205 Í
206 Î
207 Ï
209 Ñ
210 Ò
211 Ó
212 Ô
213 Õ
214 Ö
215 ×
216 Ø
217 Ù
218 Ú
219 Û
220 Ü
221 Ý
223 ß
224 à
225 á
226 â
227 ã
228 ä
229 å
230 æ
231 ç
232 è
233 é
234 ê
235 ë
236 ì
237 í
238 î
239 ï
241 ñ
242 ò
243 ó
244 ô
245 õ
246 ö
247 ÷
248 ø
249 ù
250 ú
251 û
252 ü
253 ý
255 ÿ
256 Ā
257 ā
258 Ă
259 ă
260 Ą
261 ą
262 Ć
263 ć
268 Č
269 č
270 Ď
271 ď
272 Đ
273 đ
274 Ē
275 ē
278 Ė
279 ė
280 Ę
281 ę
282 Ě
283 ě
286 Ğ
287 ğ
290 Ģ
291 ģ
298 Ī
299 ī
302 Į
303 į
304 İ
305 ı
310 Ķ
311 ķ
313 Ĺ
314 Í
315 Ļ
316 ļ
317 Ľ
318 ľ
321 Ł
322 ł
323 Ń
324 ń
325 Ņ
326 ņ
327 Ň
328 ň
332 Ō
333 ō
336 Ő
337 ő
338 Œ
339 œ
340 Ŕ
341 ŕ
342 Ŗ
343 ŗ
344 Ř
345 ř
346 Ś
347 ś
350 Ş
351 ş
352 Š
353 š
354 Ţ
355 ţ
356 Ť
357 ť
362 Ū
363 ū
366 Ů
367 ů
368 Ű
369 ű
370 Ų
371 ų
376 Ÿ

377 Ź
378 ź
379 Ż
380 ż
381 Ž
382 ž
402 *ƒ*
710 ˆ
711 ˇ
728 ˘
729 ˙
731 ˛
732 ˜
733 ˝
900 ΄
901 ΅
902 Ά
904 Έ
905 Ή
906 Ί
908 Ό
910 Ύ
911 Ώ
912 ΐ
913 Α
914 Β
915 Γ
916 Δ
917 Ε
918 Ζ
919 Η
920 Θ
921 Ι
922 Κ
923 Λ
924 Μ
925 Ν
926 Ξ
927 Ο
928 Π
929 Ρ
931 Σ
932 Τ
933 Υ
934 Φ
935 Χ
936 Ψ
937 Ω
938 Ϊ
939 Ϋ
940 ά
941 έ
942 ή
943 ί
944 ΰ
945 α
946 β
947 γ
948 δ
949 ε
950 ζ
951 η
952 θ
953 ι
954 κ
955 λ
956 μ
957 ν
958 ξ
959 ο
960 π
961 ρ
962 ς
963 σ
964 τ
965 υ
966 φ
967 χ
968 ψ
969 ω
970 ϊ
971 ϋ
972 ό
973 ύ
974 ώ
1025 Ё
1026 Ђ
1027 Ѓ
1028 Є
1029 S
1030 I
1031 Ї
1032 J
1033 Љ
1034 Њ
1035 Ћ
1036 Ќ
1038 Ў
1039 Џ
1040 А
1041 Б
1042 В
1043 Г
1044 Д
1045 Е
1046 Ж
1047 З
1048 И
1049 Й
1050 К
1051 Л
1052 М
1053 Н
1054 О
1055 П
1056 Р
1057 С
1058 Т
1059 У
1060 Ф
1061 Х
1062 Ц
1063 Ч
1064 Ш
1065 Щ
1066 Ъ
1067 Ы
1068 Ь
1069 Э
1070 Ю
1071 Я
1072 а
1073 б
1074 в
1075 г
1076 д
1077 е
1078 ж
1079 з
1080 и
1081 й
1082 к
1083 л
1084 м
1085 н

1086 о
1087 п
1088 р
1089 с
1090 т
1091 у
1092 ф
1093 х
1094 ц
1095 ч
1096 ш
1097 щ
1098 ъ
1099 ы
1100 ь
1101 э
1102 ю
1103 я
1105 ё
1106 ђ
1107 ѓ
1108 є
1109 s
1110 і
1111 ї
1112 j
1113 љ
1114 њ
1115 ћ
1116 ќ
1118 ў
1119 џ
1168 Ґ
1169 ґ
8211 –
8212 —
8213 ―
8216 ‘
8217 ’
8218 ‚
8220 “
8221 ”
8222 „
8224 †
8225 ‡
8226 •
8230 ...
8240 ‰
8249 ‹
8250 ›
8319 ⁿ
8359 Pts
8364 €
8470 №
8482 ™
8729 ∙
8730 √
8734 ∞
8745 ∩
8776 ≈
8801 ≡
8804 ≤
8805 ≥
8976 ⌐
8992 ⌠
8993 ⌡
9472 ─
9474 │
9484 ┌
9488 ┐
9492 └
9496 ┘
9500 ├
9508 ┤
9516 ┬
9524 ┴
9532 ┼
9552 ═
9553 ║
9554 ╒
9555 ╓
9556 ╔
9557 ╕
9558 ╖
9559 ╗
9560 ╘
9561 ╙
9562 ╚
9563 ╛
9564 ╜
9565 ╝
9566 ╞
9567 ╟
9568 ╠
9569 ╡
9570 ╢
9571 ╣
9572 ╤
9573 ╥
9574 ╦
9575 ╧
9576 ╨
9577 ╩
9578 ╪
9579 ╫
9580 ╬
9600 ▀
9604 ▄
9608 █
9612 ▌
9616 ▐
9617 ░
9618 ▒
9619 ▓
9632 ■
9633 □
9642 ▪
9643 ▫
9644 ▬
9650 ▲
9658 ►
9660 ▼
9668 ◄
9674 ◊
9675 ○
9679 ●
9688 ◘
9689 ◙
9702 ◦
9786 ☺
9787 ☻
9788 ☼
9792 ♀
9794 ♂
9824 ♠
9827 ♣
9829 ♥
9830 ♦
9834 ♪
9835 ♫

Эл Свейгарт

Большая книга проектов Python

Перевел с английского *И. Пальти*

Заведующая редакцией	*Ю. Сергиенко*
Ведущий редактор	*Н. Гринчик*
Литературный редактор	*Н. Хлебина*
Художественный редактор	*В. Мостипан*
Корректоры	*М. Молчанова, Е. Павлович*
Верстка	*Л. Егорова*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 02.2022. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 — Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 10.12.21. Формат 70×100/16. Бумага офсетная. Усл. п. л. 34,830. Тираж 1500. Заказ 0000.