САНКТ-ПЕТЕРБУРГ
МОСКВА
КРАСНОДАР

ЛАНЬ

Т. П. НИКИТИНА,
Л. В. КОРОЛЕВ

ПРОГРАММИРОВАНИЕ. ОСНОВЫ PYTHON ДЛЯ ИНЖЕНЕРОВ

Учебное пособие

ЛАНЬ
· САНКТ-ПЕТЕРБУРГ · МОСКВА · КРАСНОДАР ·
2023

УДК 004.43
ББК 22.19я73

Н 62 **Никитина Т. П.** Программирование. Основы Python для инженеров : учебное пособие для вузов / Т. П. Никитина, Л. В. Королев. — Санкт-Петербург : Лань, 2023. — 156 с. : ил. — Текст : непосредственный.

ISBN 978-5-507-45284-2

Пособие посвящено рассмотрению базовых конструкций языка Python, в частности, сначала приведены примеры простейших программ в императивном стиле программирования и примеры решения несложных задач линейной, разветвляющейся и циклической структуры, задач с последовательностями и файлами. Далее дана реализация в виде программ на Python алгоритмов методов вычислительной математики. Большое внимание уделено практике использования библиотек numpy, matplotlib, pandas и turtle, для анализа данных и их графической интерпретации.

Учебное пособие предназначено для использования в учебном процессе студентами, обучающимися по направлениям подготовки «Технологические машины и оборудование», «Химия», «Энергетическое машиностроение», «Эксплуатация транспортно-технологических машин и комплексов» и других инженерных специальностей всех форм обучения при изучении дисциплин математического и естественнонаучного цикла. Учебное пособие разработано в соответствиями с требованиями Федерального государственного образовательного стандарта.

УДК 004.43
ББК 22.19я73

Рецензент

В. А. СОКОЛОВ — доктор физико-математических наук, профессор, зав. кафедрой теоретической информатики Ярославского государственного университета им. П. Г. Демидова.

Обложка
П. И. ПОЛЯКОВА

Введение

Язык программирования Python разработал голландец Гвидо ван Россум. Python — интерпретируемый, объектно-ориентированный высокоуровневый язык программирования с динамической семантикой. Встроенные высокоуровневые структуры данных в сочетании с динамической типизацией и связыванием делают язык привлекательным для быстрой разработки приложений (RAD, Rapid Application Development). Кроме того, его можно использовать в качестве сценарного языка для связи программных компонентов.

Интерпретатор выполняет инструкции построчно: после приглашения к работе записывается строка с необходимыми действиями, после нажатия клавиши Enter, интерпретатор выдает результат. Python можно использовать как калькулятор.

Другой вариант работы — работа в среде разработки IDLE (Integrated Development and Learning Environment).

Основные понятия и инструкции Python

Структура программы

Программы на языке Python состоят из инструкций и являются обычными текстовыми файлами, которые обычно имеют расширение *.py*. Эти файлы для просмотра и редактирования можно открывать с помощью любого текстового редактора, например программы Блокнот.

Правила записи инструкций.

- Конец строки является концом инструкции.
- Вложенные инструкции объединяются в блоки по величине отступов.
- Отступ может быть любым, главное, чтобы в пределах одного вложенного блока был одинаковый отступ. Не следует использовать отступ в один пробел, так как существенно снижается наглядность и восприятие человеком блочной структуры программы. Используйте четыре пробела (знак табуляции).
- Вложенные инструкции в Python записываются по следующему шаблону: основная инструкция завершается двоеточием, вслед за которым, чаще всего после нажатия клавиши Enter, располагается вложенный блок кода с необходимым отступом:

```
for i in range (0, n):
    if a[0][i]>0: k+=1
    if a[n-1] [i]>0: k+=1
```

- Тело составной инструкции может располагаться в той же строке, что и тело основной, если тело составной инструкции не содержит составных инструкций:

```
if x > y: print(x)
```

- Можно записать несколько инструкций в одной строке, разделяя их точкой с запятой:

```
a = 1; b = 2; print (a, b)
```

- Допустимо записывать одну инструкцию на нескольких строках. Достаточно ее заключить в пару круглых, квадратных или фигурных скобок:

```
if (a == 1 and b == 2 and
c == 3 and d == 4): # Не забываем про двоеточие
print ('if занимает две строки')
```

Имена переменных

Каждый объект программы должен иметь идентификатор, задаваемый пользователем. На основе идентификатора строится имя объекта, которое позволяет обращаться как ко всему объекту, так и к отдельным его составляющим.

Синтаксические конструкции Python записывают с использованием уникальных ключевых слов, которые нельзя использовать в качестве идентификаторов.

Список ключевых слов можно получить с помощью следующих инструкций:

```
import keyword
print(keyword.kwlist)
```

Результат

```
['and', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'exec', 'finally', 'for', 'from',
'global', 'if', 'import', 'in', 'is', 'lambda', 'not', 'or',
'pass', 'print', 'raise', 'return', 'try', 'while', 'yield']
```

Переменную можно связать с объектом в любом месте блока, важно, чтобы это произошло до ее использования, иначе появится синтаксическая ошибка *NameError*. В частности, связывание имен со значениями происходит в инструкциях присваивания.

Правила записи.

- Всегда следует связывать переменную со значением до ее использования.
- Необходимо избегать глобальных переменных и передавать все необходимые данные через параметры.
- Убрать связь имени с объектом можно с помощью оператора *del*. В этом случае, если объект не имеет других ссылок на него, он будет удален. Для управления памятью в Python используется подсчет ссылок, для удаления наборов объектов с зацикленными ссылками — сборка мусора (*garbage collection*).

В каждой точке программы интерпретатор «видит» три пространства имен: локальное, глобальное и встроенное. Пространство имен связано с понятием блока кода. В Python блоком кода является то, что исполняется как единое целое, например тело цикла, функции, условной инструкции.

Локальные имена — имена, которым присвоено значение в блоке, доступны только в нем. Глобальные имена — имена, определяемые на уровне блока модуля или те, которые явно заданы как *global*. Встроенные имена — имена из специального словаря *builtins*.

Области видимости имен могут быть вложенными друг в друга, например, внутри вызванной функции видны имена, определенные в вызывающем коде. Переменные, которые используются в блоке кода, но связаны со значением вне кода, называются свободными переменными.

Константы и переменные

Данные представлены константами и переменными. Все данные в Python представляют собой объекты.

Каждый объект содержит как минимум три вида данных:

- счетчик ссылок — используется для управления памятью;
- тип;
- значение.

Python — это язык программирования с динамической типизацией, то есть в ходе выполнения программы одна и та же переменная может хранить значения различных типов. Типы данных можно разделить на встроенные в интерпретатор и не встроенные, которые можно использовать при импортировании соответствующих модулей.

В языке Python, как и в других языках программирования, например C++, различают неизменяемые (константные) и изменяемые типы данных. Основное различие при работе с ними заключается в том, что для данных с неизменяемым типом запрещены инструкции, меняющие значение объекта.

К основным встроенным типам относятся следующие.

- *None* (неопределенное значение переменной).
- Логические значения:
  - *True*;
  - *False.*
- Числа (неизменяемые типы):
  - *int* — целое число;
  - *float* — вещественное число;
  - *complex* — комплексное число.
- Списки:
  - *list* — список;
  - *tuple* — кортеж (неизменяемый тип);
  - *range* — диапазон (неизменяемый тип).
- Строки *str* (неизменяемый тип).

Арифметические константы могут быть представлены в программе своими значениями (явно):

- целые числа 4 687 –45 0;
- вещественные значения:
  - с фиксированной точкой 1.45 –3.789654 0.00453;

- с плавающей точкой 1.0E-5 –5.123e2 0.1234E3.

Операции. Присваивание. Выражение

Операция — выполнение каких-либо действий над данными (операндами). Для выполнения конкретных действий требуются специальные инструменты — операторы.

В программе на языке Python связь между данными и переменными задается с помощью знака «=». Такая операция называется присваиванием.

Простое присваивание: <имя> = <выражение>

Пример: $a = 2$

Переменная с именем a связывается со значением 2.

Каскадное присваивание: $z = a = 0$, в этом примере происходит обнуление значений переменных z и a.

Множественное присваивание:

$z, a = 0, -1$ в результате $z = 0, a = -1$.

Пример

$i = 1; x = 2.21$

$i = x = 0$

$i, x = -3,2.5**2$

Составное присваивание позволяет сократить запись:

Операция	Действие
a+=b	*a=a+b*
a–=b	*a=a–b*
a=b*	*a=a*b*
a/=b	*a=a/b*
*a**=b*	*a=a**b*
a%=b	*a=a%b*
a//=b	*a=a//b*

Пример

```
z, z1=0, –1
print (" z=", z, " z1=", z1)
z+=3
print (" z=", z)
z*=2
print (" z=", z)
z1/=2
print (" z1=", z1)
```

Результат

Выражение — правило для вычисления значения. Состоит из операндов, соединенных знаками операций. В качестве операндов могут выступать переменные, константы, указатели функций, выражения в круглых скобках.

Тип переменной в левой части оператора присваивания должен совпадать с типом значения выражения в правой части. Возможны случаи, когда выполняется автоматическое преобразование, при котором исключены какие-либо потери. Например, слева от знака присваивания стоит переменная вещественного типа, а справа — выражение целого типа. В этом случае целое автоматически преобразуется к вещественному значению и при этом исключается потеря точности.

Выражения являются составной частью операторов. Вычисление выражений осуществляется слева направо, за исключением операции возведения в степень (справа налево), с учетом наличия круглых скобок и приоритетом операций.

Приоритеты операций

Приоритеты операций по возрастанию приоритета:

Операция	Название	Приоритет
lambda	лямбда-выражение	низкий
or	логическое ИЛИ	
and	логическое И	
not x	логическое НЕ	
in, *not in*	проверка принадлежности	
is, *is not*	проверка идентичности	
<, <=, >, >=, !=, = =	сравнения	
|	побитовое ИЛИ	
^	побитовое исключающее ИЛИ	
&	побитовое И	
<<, >>	побитовые сдвиги	
+, –	сложение и вычитание	
*, /, %, //	умножение, деление, остаток	
+*x*, –*x*	унарный плюс и смена знака	
~*x*	побитовое НЕ	
**	возведение в степень	высокий

Последовательность операторов. Блок

Последовательные действия описываются следующими друг за другом строками программы. Операторы, имеющие одинаковый отступ и входящие в единую последовательность действий, образуют блок. Блок с синтаксической точки зрения рассматривается как один оператор.

Пример

```
a = 11
b = –222
print ("a=", a,"   b=", b)
a = a + b
```

```
b = a – b
a = a – b
print ("a=", a, "   b=", b)
```

Эти операторы образуют блок, меняющий местами значения *a* и *b*.

Ввод данных с клавиатуры. Функция input()

Когда функция *input*() выполняется, то поток выполнения программы останавливается в ожидании данных, которые пользователь должен ввести с клавиатуры. После ввода данных и нажатия клавиши Enter функция *input*() завершает свое выполнение и возвращает результат, который представляет собой строку символов, введенных пользователем. Если требуется получить арифметическое значение, то необходимо изменить строковый тип на числовой с помощью функций явного указания типа *int*() или *float*().

Результат, возвращаемый функцией *input*(), обычно присваивают переменной для дальнейшего использования в программе.

Пример

```
a=int(input("a=")) # целое число
b=float(input("b=")) # вещественное число
st= input () # строка
```

Вывод данных на экран. Функция print()

Для вывода результатов работы программы на экран компьютера используется функция *print*(). Самое простое обращение для вывода выглядит так: *print*(имя[,имя]...)

Пример

```
a=5.56; print(a)
b=float(input("b="))
st="Привет!!!"
print(a, b, st)
```

Форматирование вывода. Метод format()

Синтаксическая конструкция:

<спецификация>.*format*(<строка вывода>)

Возвращается строка вывода, отформатированная в соответствии со спецификацией.

Спецификаторы.

- Выравнивание:

< — выравнивание по левому краю;

> — выравнивание по правому краю;

^ — выравнивание по центру;

= — помещает результат в крайнее левое положение.

- Знаки (только для числовых значений):

+ — знак «плюс»;

– — знак «минус» (только для отрицательных значений).

- Разделители десятков (только для числовых значений):

, — в качестве разделителя тысяч;

_ — в качестве разделителя тысяч.

- Точность:

.число — количество цифр выводимых после фиксированной точки или количество символов в строке.

- Тип форматируемого объекта:

s — строка (по умолчанию);

c — преобразует целое число в символ Unicode;

d — десятичный формат;

e — формат с плавающей точкой со строчной буквой e;

E — формат с плавающей точкой с заглавной буквой E;

f — формат чисел с плавающей точкой;

F — формат чисел с плавающей точкой, верхний регистр;

g — общий формат, нижний регистр;

G — общий формат, верхний регистр;

n — формат целых чисел;

% — умножает число на 100 и использует *f* для вывода. В конце ставится %.

Пример

```
p = –24.5
t = 97.1
print('{:+f} {:+f}'.format(p, t))
print('{: f}; {: f}'.format(p**2, t*3))
print('{:<5};{:<5}; {:+f}'.format('left',' ', p))
print('{:>10}'.format('right'))
print('{:^30}'.format('по центру'))
print('{:*^30}'.format('*по центру*'))
print('{:d};{:E}'.format(123, -3.14e0))
print(24.5/97.1)
print(' Ответ в процентах: {:.2%}'.format(abs(p)/t))
```

Результат

```
–24.500000 +97.100000
 600.250000;  291.300000
left ;     ; –24.500000
    right
        по центру
**********по центру**********
123; –3.140000E+00
0.2523171987641607
```

Ответ в процентах: 25.23%

Целые числа (int)

Основные операции для целых чисел:

$x + y$	сложение
$x - y$	вычитание

x * *y*	умножение
x / *y*	деление
x // *y*	получение целой части от деления
x % *y*	остаток от деления
-*x*	смена знака числа
abs(*x*)	модуль числа
divmod(*x*,*y*)	пара (*x* // *y*, *x* % *y*)
x ** *y*	возведение в степень
pow(*x*, *y*[,*z*])	x^y по модулю (если модуль задан)

Над целыми числами можно производить битовые операции:

x | *y*	побитовое ИЛИ
x ^ *y*	побитовое ИСКЛЮЧАЮЩЕЕ ИЛИ
x & *y*	побитовое И
x << *n*	битовый сдвиг влево
x >> *n*	битовый сдвиг вправо
~*x*	инверсия битов

Пример

```
>>> print (16<<3)
128
>>> print(16>>2)
4
```

Целые числа поддерживают длинную арифметику.

Вещественные числа (float)

Для вещественных значений основные арифметические операции те же, что и для целых значений:

+ — сложение;

– — вычитание;

* — умножение;

/ — деление;

** — возведение в степень (выполняется справа налево).

Особенности операции возведения в степень:

```
r=2**3**2
print("2**3**2=",r)
r=2.1**3.12**1.9
print("2.1**3.12**1.9=",r)
```

Результат

```
2**3**2= 512
2.1**3.12**1.9= 629.9021693067386
```

Арифметические значения могут быть операндами операций сравнения:

Операция	Описание
= =	если значения двух операндов равны, то условие становится истинным
!=	если значения двух операндов не равны, то условие становится истинным
>	если значение левого операнда больше значения правого операнда, то условие становится истинным
<	если значение левого операнда меньше значения правого операнда, то условие становится истинным
> =	если значение левого операнда больше или равно значению правого операнда, то условие становится истинным
<=	если значение левого операнда меньше или равно значению правого операнда, то условие становится истинным

Комплексные числа (complex)

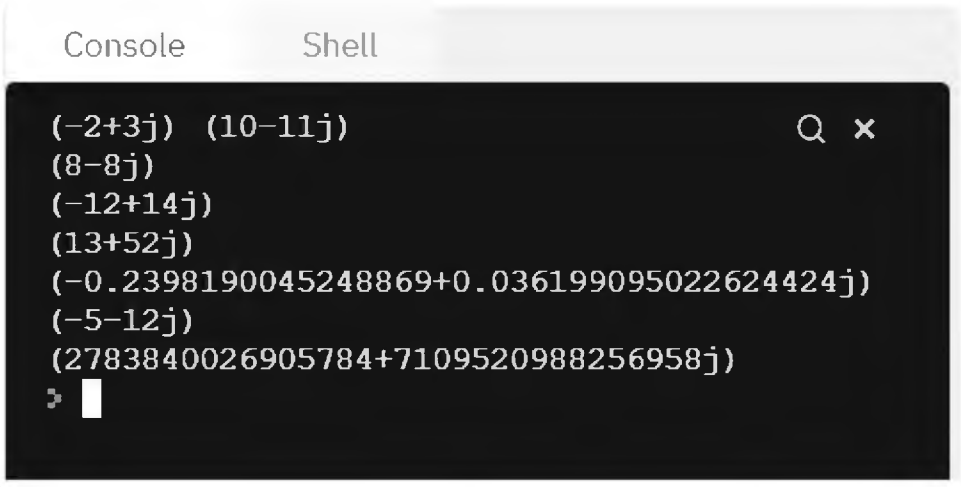
Тип данных *complex* относится к категории неизменяемых и хранит пару числовых значений, одно из которых представляет действительную часть комплексного числа, а другое — мнимую.

Над комплексными числами определены основные арифметические операции (+, –, *, /, **).

Пример

```
z=–2+3j; a=10–11j; print(z, a)
r=z + a; print(r)
r1=z – a; print(r1)
r2=z * a; print(r2)
r3=z / a; print(r3)
r4=z ** 2; print(r4)
r5=z ** a; print(r5)
```

Результат



Над комплексными числами определены следующие операции сравнения:

Операция	Результат
x == y	*True* если *x* равно *y*, иначе *False*
x != y	*True* если *x* не равно *y*, иначе *False*
x is y	*True* если *x* и *y* это один и тот же объект, иначе *False*
x is not y	*True* если *x* и *y* это не один и тот же объект, иначе *False*

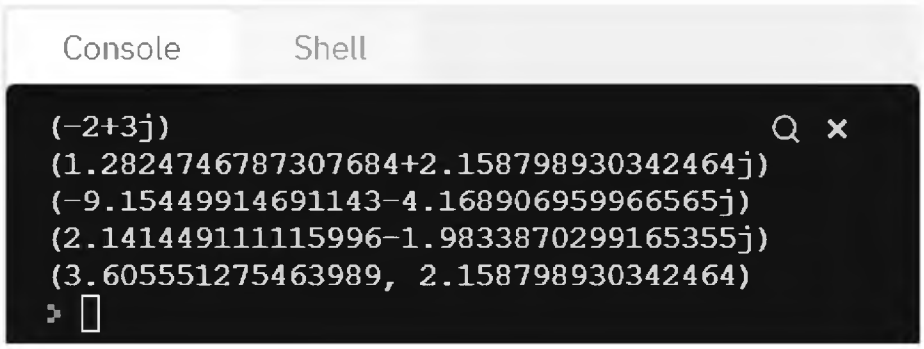
Модуль *cmath* предоставляет доступ к функциям, которые могут выполнять математические действия над комплексными числами:

Функция	**Результат**
cmath.exp(*x*)	вычисляет экспоненту комплексного числа
cmath.log(*x*), [основание])	вычисляет логарифм комплексного числа с указанным основанием, если основание не указано, то возвращается значение натурального логарифма
*cmath.log*10(*x*)	вычисляет десятичный логарифм комплексного числа
cmath.sqrt(*x*)	вычисляет квадратный корень комплексного числа
cmath.sin(*x*)	возвращает синус комплексного числа
cmath.cos(*x*)	возвращает косинус комплексного числа
cmath.tan(*x*)	возвращает тангенс комплексного числа
cmath.asin(*x*)	возвращает арксинус комплексного числа
cmath.acos(*x*)	возвращает арккосинус комплексного числа
cmath.atan(*x*)	возвращает арктангенс комплексного числа
cmath.polar(*x*)	возвращает полярные координаты комплексного числа

Пример

```
import cmath
z=–2+3j;  print(z)
r=cmath.log(z); print(r)
r1=cmath.sin(z); print(r1)
r2=cmath.acos(z); print(r2)
r3=cmath.polar(z); print(r3)
```

Результат



Логические значения (bool)

Логические константы:

— *True* — истина;

— *False* — ложь.

Результатом всех операций сравнения (==, !=, >, <, >=, <=) является логическое значение.

Основные логические операции:

- конкатенация

A and B

Истина, если оба значения *A* и *B* истинны;

- дизъюнкция

A or B

Истина, если хотя бы одно из значений *A* или *B* истинно;

- инверсия (отрицание)

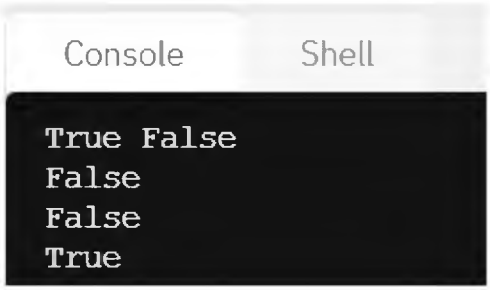
not A

Истина, если *A* ложно и наоборот.

Пример

```
a=True; b=8<2; print(a,b)
print(a and b)
print(not a)
print(a or b)
```

Результат



Строки (str)

Строка состоит из последовательности символов. Константы строки записываются в двойных или одинарных кавычках: "Привет!", *'Error'*. Константы символы записываются в одинарных кавычках: *'G'*, '$'.

Строка считывается со стандартного ввода функцией *input*().

Часто используемые действия со строками:

- Конкатенация (сложение)

```
s="Python+"
st="Пайтон"
print(s+st)
```

Результат: "*Python*+ Пайтон"

- Дублирование строки

```
st="Привет!"
print(st*3)
```

Результат: Привет!Привет!Привет!

- Длина строки

```
st="Пайтон"
print(len(st))
```

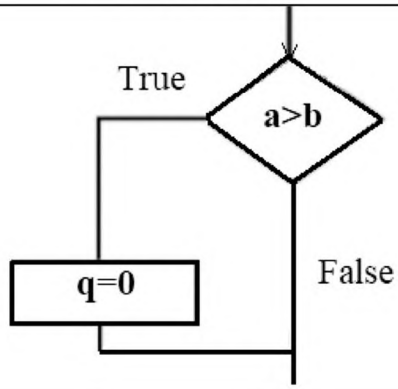
Результат: 6

- Срез — извлечение из данной строки одного символа или некоторого числа символов. Есть три формы срезов. Самая простая форма среза: взятие одного символа строки, а именно, *S*[*i*] — срез, состоящий из одного символа, который имеет номер *i* (нумерация начинается с числа 0)

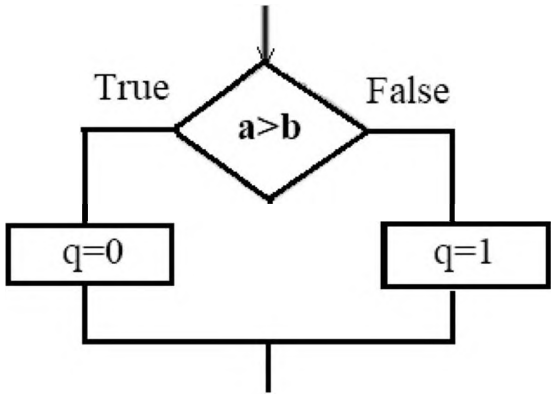
```
st="A1d23*%"
print(st[0]," ",st[3]," ",st[–2])
```

Результат: A 2 *

Оператор условия. Множественное ветвление

Оператор	Блок-схема
f < условие >: < оператор >	

Продолжение табл.

if < условие > : < оператор > *else:* < оператор >	

В язык Python встроена возможность множественного ветвления на одном уровне вложенности, которое реализуется с помощью веток *elif* (сокращение *else if*). В отличие от *else*, в заголовке *elif* обязательно должно быть логическое выражение, так же как в заголовке *if*.

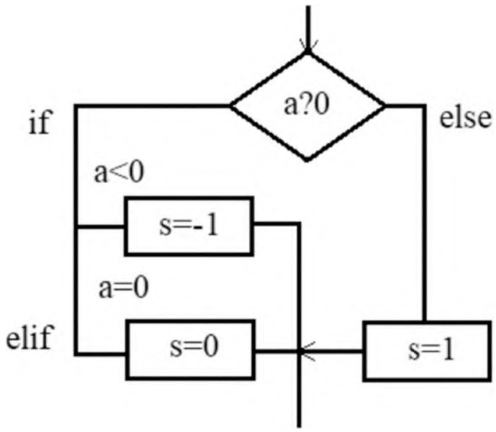
Пример оператора условия:

```
if a > b:
    q = 0
else:
    q = 1
```

Пример оператора множественного ветвления:

```
if a < 0:
    s = –1
elif a == 0:
    s = 0
else:
    s = 1
```

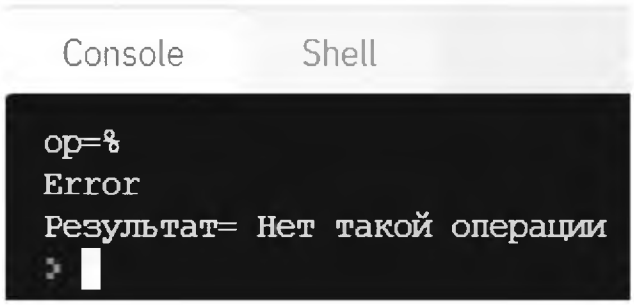
Блок-схема

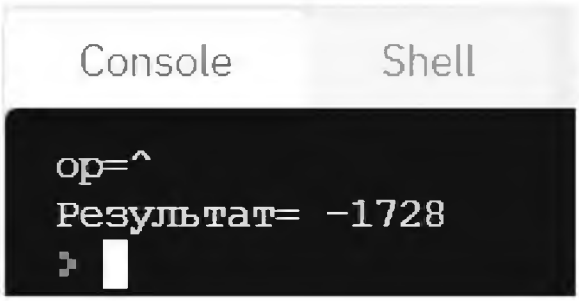


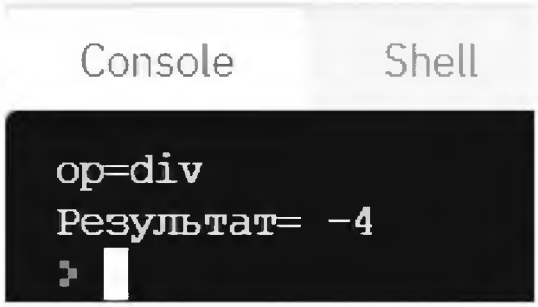
Пример простейшего калькулятора:

result = "Нет такой операции"
n = 3; *m*=–12
op=input("op=")
if op == "+": result = n+m
elif op == "-": result = n–m
elif op == "": result = n*m*
elif op == ":": result = n/m
elif op == "div": result = m//n
*elif op == "^": result = m**n*
else: print ("Error")
print("Результат=", *result*)

Результаты работы калькулятора







Цикл while

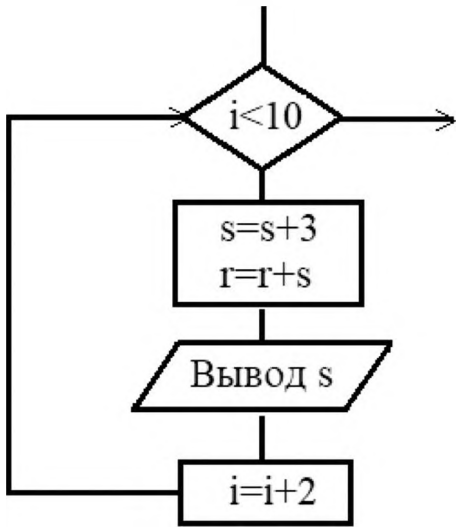
while — один из самых универсальных циклов в Python, поэтому довольно медленный. Выполняет тело цикла до тех пор, пока условие цикла истинно.

while < условие >:
    < блок инструкций >

Пример

i=2; *s*=0; *r*=1
while i < 10:
    s+=3
    r+=s
    print (*s*)
    i+=2

Блок-схема



Результат

3
6
9
12

Тело цикла выполняется четыре раза, значение переменной *i*, приводящее к завершению цикла, равно 10.

Цикл for

Цикл *for* сложнее и менее универсален, чем цикл *while*, но он выполняется гораздо быстрее. Этот цикл проходится по любому итерируемому объекту (например, строке или списку) и во время каждого прохода выполняет тело цикла.

Наиболее распространены две конструкции оператора *for*:

1) *for* <имя> *in* <список объектов>*:*
   < блок инструкций >
2) *for* <имя> *in range*(<начальное значение>, <конечное значение>, <шаг>):
   < блок инструкций >

Пример

Найти сумму цифр числа, заданного строкой знаков.

```
ta=0
for i in "654321":
    ta+=int(i)
print(ta)
```

Результат

Пример

Разделить символы исходной строки последовательностью знаков: пробел * пробел.

```
stroka = "привет"
for b in stroka:
  print(b, end=' * ')
```

Результат

Функция range()

Есть три способа вызова *range*():

- один аргумент, начальное значение = 0

range (<конечное значение>)

Пример

```
for i in range(3):
    print (i, end=' ')
```

Результат

- два аргумента

range (<начальное значение>, <конечное значение>)

Пример

```
for i in range(–4,3):
    print(i, end=' ')
```

Результат

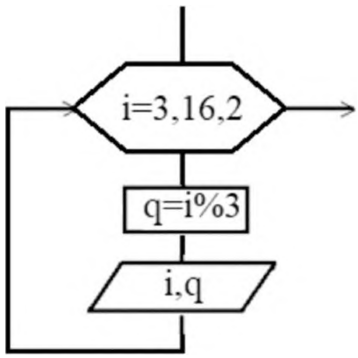
- три аргумента

range (<начальное значение>, <конечное значение>, <шаг>)

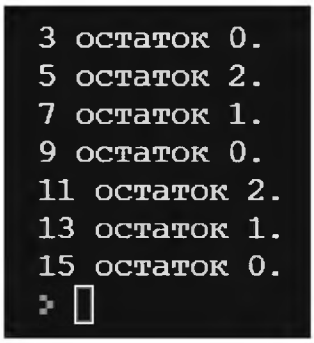
Пример

```
for i in range( 3, 16, 2 ):
    q = i % 3
    print(f"{i} остаток {int(q)}.")
```

Блок-схема



Результат



Оператор continue. Оператор break. Слово else

Оператор *continue* начинает следующий проход цикла, минуя оставшееся тело цикла *for* или *while*.

Оператор *break* досрочно прерывает цикл.

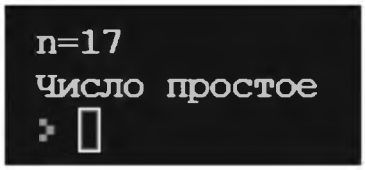
Слово *else*, примененное в цикле *for* или *while*, проверяет, был ли произведен выход из цикла инструкцией *break* или цикл был выполнен полностью. Блок инструкций внутри *else* выполнится только в том случае, если выход из цикла произошел без помощи *break*.

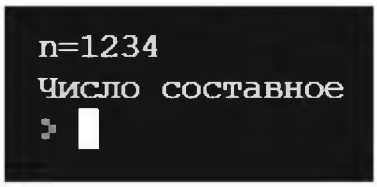
Пример

Дано натуральное число. Требуется определить, является ли число простым.

```
n = int(input("n="))
bol = True
for i in range (2, n):
    if n % i == 0:
# если исходное число делится на какое-либо отличное от 1 и самого себя
        bol = False
# останавливаем цикл если встретили делитель числа
        break
if bol == True:
    print('Число простое')
else:
    print('Число составное')
```

Результаты



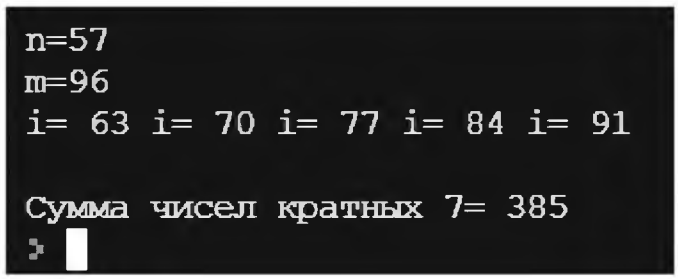


Пример

Дана последовательность натуральных чисел от n до m–1. Найти сумму чисел из этой последовательности кратных 7.

```
n = int(input("n="))
m = int(input("m="))
s=0
for i in range(n, m):
  if i % 7 != 0:
     continue
  print ("i=", i, end=' ')
  s+=i
print('\n')
print ("Сумма чисел кратных 7=", s)
```

Результат



Пример

Дана последовательность натуральных чисел, от *n* до *m*–1. Считать сумму чисел последовательности, пока она не превышает 1000.

```
n = int(input("n="))
m = int(input("m="))
s=0
for i in range(n, m):
  s+=i
  if s>1000:
    break
    print ("i=", i, end=' ')
else:
  print('Продолжаем расчет без s>1000')
print("s=", s)
```

Результаты

```
n=20
m=30
Продолжаем расчет без s>1000
s= 245
```

```
n=20
m=100
s= 1035
```

Функции

Функции — изолированный блок инструкций языка, обращение к которому в процессе выполнения программы может быть многократным. Предполагается, что функция в точку вызова возвращает не только управление, но и значение.

Описание функции. Программист может определять собственные функции двумя способами: с помощью оператора *def* или *lambda*-функции (анонимные функции).

Функции def

Описание:

def <Имя функции> ([Параметры]):
  <Блок>
  [*return* <Значение>]

def — ключевое слово, с которого начинается заголовок функции. Имя функции может быть любым, но желательно осмысленным. Один или несколько параметров записываются через запятую в круглых скобках. Даже если параметров у функции нет, круглые скобки указываются обязательно. Далее идет двоеточие, обозначающее окончание заголовка функции (аналогично с условиями и циклами). После заголовка с новой строки и с отступом следуют инструкции тела функции.

Тело функции — блок, состоящий из инструкций. Заканчивается функция инструкцией, перед которой находится меньшее количество пробелов, и которая принадлежит блоку внешнему по отношению к рассматриваемому.

В функции чаще всего присутствует инструкция *return*, которая завершает работу функции, возвращает управление и значение в точку вызова.

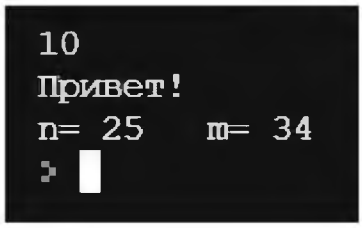
Вызов функции состоит из имени функции и списка фактических параметров, заключенного в круглые скобки. Вызов функции может быть операндом выражения.

Пример

```
def max (x, y):
   z=x if x>y else y
   return z
def newfunc (n,x):
   d=n//x
```

```
    o=n%x
    return (d, o)

print(max(1, 10))
print(max("Привет!","Hellow" ))
n,m = newfunc(1234, 48)
print("n=", n," m=", m)
```

Результат



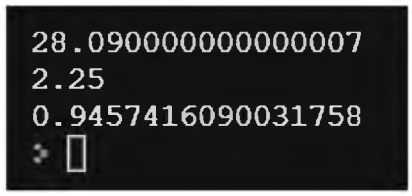
Из результата работы программы следует, что строка Привет! «больше» строки *Hellow*. Если добавить строку *print* (*ord*("П"), (*ord*("*H*"))), увидим коды символов П и *H*: 1055 72, и еще учесть, что длина строки Привет! равна 7, а длина строки *Hellow* — 6, то полученный ответ становится очевидным.

Функция может принимать произвольное количество аргументов или не иметь их вовсе. Так же распространены функции с произвольным числом аргументов, функции с позиционными и именованными аргументами, обязательными и необязательными.

Пример

```
def func(a, b, c=2):       # c — необязательный аргумент
    return (a + b)**c
print (func(1.1, 4.2))     # c = 2
print (func(a=2, b=–0.5))        # c = 2
print (func(a=1.3,b=–0.5, c=0.25))
```

Результат



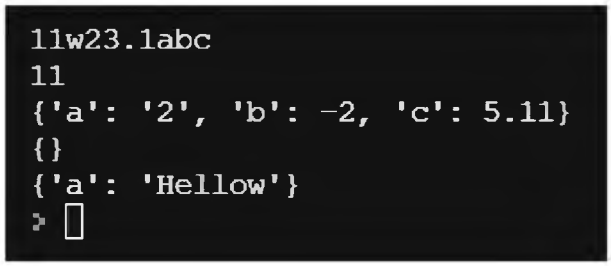
Функция может принимать переменное количество позиционных аргументов, тогда в списке формальных параметров в описании функции, перед именем формального параметра ставится «*».

Функция может принимать произвольное число именованных аргументов, тогда перед именем ставится «**».

Пример

```
def fun1(*ar):
    s=""
    # конкатенация заданного при вызове количества аргументов
for i in ar:
        s+=str(i)
    return s
def fun2(**kar):
    return kar
print(fun1(11, "w2", 3.1, 'abc'))      # три параметра
print(fun1(11))                        # один параметр
print(fun2(a="2", b=-2, c=5.11))   # три параметра
print(fun2())                          # без параметров
print(fun2(a="Hellow"))                # один параметр
```

Результат



Параметром функции может быть имя другой функции.

Пример

Составить программу табулирования заданных функций.

Табулирование функции — вычисление значений функции при изменении аргумента от некоторого начального значения *xn* до некоторого конечного значения *xk* с шагом *h*. Последовательность значений аргумента образует арифметическую прогрессию. При задании исходных значений необходимо, чтобы *xk* было больше, чем *xn* при положительном шаге *h*. Для отрицательного шага *h* должно быть справедливо соотношение *xn* > *xk*.

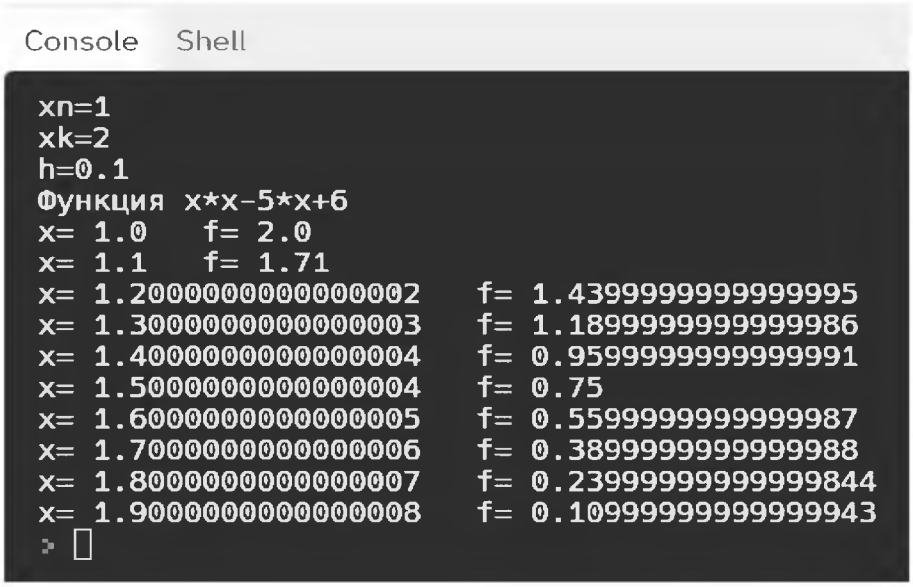
Построим таблицу значений функции *f*(*x*)=*x***x*–5**x*+6.

Результатом работы является таблица значений, в каждой строке которой сначала выводится значение аргумента, а потом значение функции.

Программа

```
import math
def fun1(x): # функция f(x)
   return x*x–5*x+6
def Tabf(x0,x1,dx,f): # реализация табулирования функции f
 x=x0
 while x<=x1 :
   z=f(x)
   print("x=", x, " f=",z)
   x+=dx
xn=float(input("xn="))
xk=float(input("xk="))
h=float(input("h="))
print("Функция x*x–5*x+6")
Tabf(xn,xk,h,fun1)
```

Результат

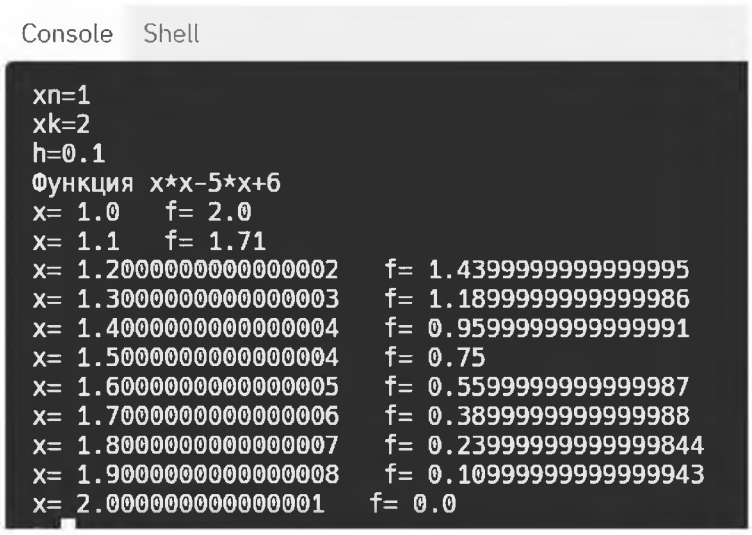


Если внимательно посмотрим на результат, то увидим, что программа работает некорректно, так как отсутствует значение функции при $x=2$. Очевидно, что это происходит из-за условия окончания цикла. Запишем условие окончания цикла $x < xk$ + малое положительное число, например $1E{-}10$.

Программа

```
import math
def fun1(x): # функция f(x)
    return x*x-5*x+6
def Tabf(x0,x1,dx,f):
  x=x0
  while x<x1+1E-10   # добавили 1E-10
    z=f(x)
    print("x=", x, "  f=", z)
    x+=dx
xn=float(input("xn="))
xk=float(input("xk="))
h=float(input("h="))
print("Функция x*x-5*x+6")
Tabf(xn,xk,h,fun1)
```

Результат



Последняя строка появилась и в ней значение x больше 2. Делаем вывод: работая с вещественными значениями надо всегда помнить, что вещественные значения имеют ограниченную точность и имеет место вычислительная погрешность.

Правильно работает и следующая программа:

```
import math
def fun1(x):
    return x*x–5*x+6
def Tabf(x0,x1,dx,f):
  x=x0
  while x<x1:
    z=f(x)
    print("x=", x, " f=",z)
    x+=dx
xn=float(input("xn="))
xk=float(input("xk="))
h=float(input("h="))
print("Функция x*x–5*x+6")
Tabf(xn,xk,h,fun1)
print("x=", xk, " f=",fun1(xk))
```

Пример программы для нескольких функций

Построить таблицу значений функций.

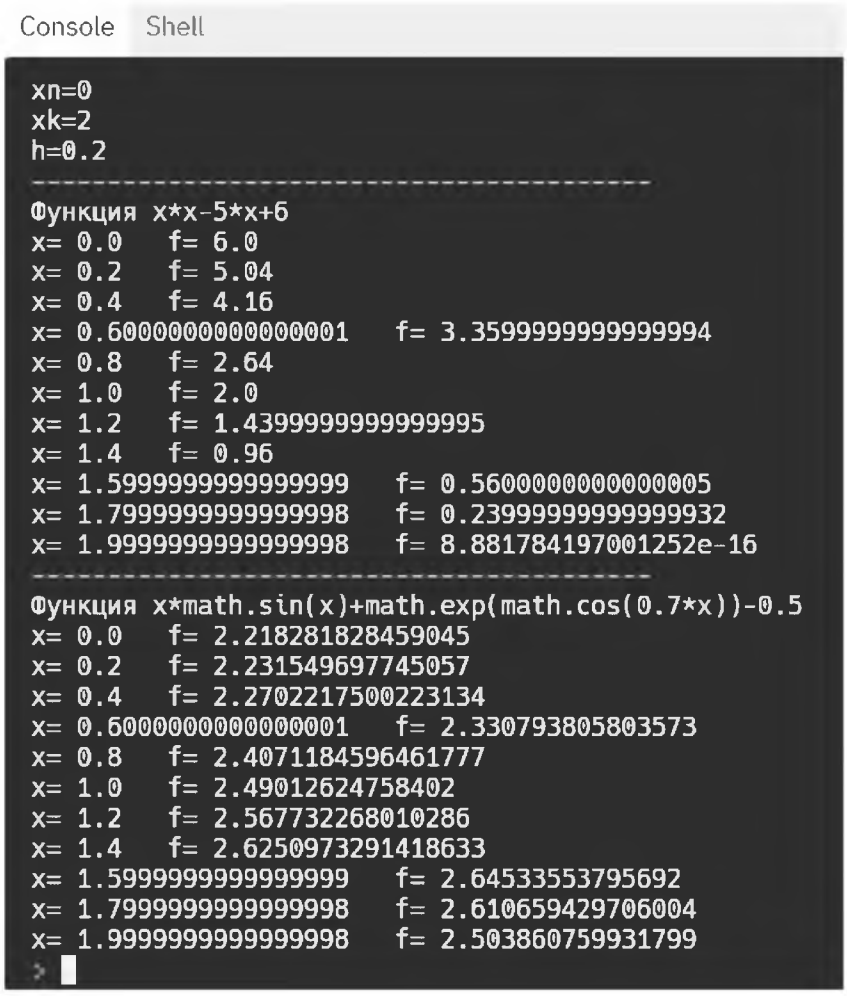
*fun*1(*x*)=*x***x*-5**x*+6.

*fun*2(*x*)= $x * \sin(x) + e^{\cos(0,7*x)} - 0,5$.

Программа

```
import math
def fun1(x):
  return x*x–5*x+6
def fun2(x):
  return (x*math.sin(x)+math.exp(math.cos(0.7*x))–0.5)
def Tabf(x0,x1,dx,f):
    x=x0
    while x<x1+1E–10 :
      z=f(x)
      print("x=", x, "  f=",z)
      x+=dx
xn=float(input("xn="))
xk=float(input("xk="))
h=float(input("h="))
print("----------------------------------------")
print("Функция x*x–5*x+6")
Tabf(xn,xk,h,fun1)
print("----------------------------------------")
print("Функция x*math.sin(x)+math.exp(math.cos(0.7*x))–0.5")
Tabf(xn,xk,h,fun2)
```

Результат



Анонимные функции. Инструкция lambda

Анонимные функции создаются с помощью инструкции *lambda*. Анонимные функции не имеют имени, а содержат только выражение, однако выполняются они быстрее. *lambda*-функции, в отличие от обычных функций, не используют инструкцию *return* для передачи значения в точку вызова.

Описание:

lambda [<Параметр>[,<Параметр>]...]: <Выражение>.

У этих функций нет имени, именно поэтому их называют анонимными. При вызове анонимная функция возвращает ссылку на объект-функцию. Вызывается анонимная функция указанием круглых скобок, внутри которых указаны передаваемые параметры.

Пример

```
fun=lambda x, a=–5,y=6: x*x + a*x + y
a=float(input("a= "))
b=float(input("b= "))
c=float(input("c= "))
print("fun(a)=",fun(a) )
print("fun(a,b)=",fun(a,b))
print("fun(a,b,c)=",fun(a,b,c))
print("********************")
print(fun(a,b))
for x in range(0,8,2):
    print('{:s}{:.2f}   {:s}{:.2f}'.format('x=', x/2,'f=',fun(x/2)))
print('********************')
for x in range(-6,6,2):
    print('{0:<3} {1:<8} {2:<3} {3:<8}'. format('x=',x/2, 'f=', fun(x/2)))
```

Результат

```
a= 1.00
b= 2.00
c= 0.10
fun(a)= 2.0
fun(a,b)= 9.0
fun(a,b,c)= 3.1
********************
9.0
x=0.00   f=6.00
x=1.00   f=2.00
x=2.00   f=0.00
x=3.00   f=0.00
********************
x=  -3.0     f=  30.0
x=  -2.0     f=  20.0
x=  -1.0     f=  12.0
x=  0.0      f=  6.0
x=  1.0      f=  2.0
x=  2.0      f=  0.0
>
```

Функции генераторы. Инструкция yield

Функция-генератор — функция, которая может возвращать при каждом обращении одно значение из заданной последовательности.

Приостановить выполнение функции позволяет инструкция *yield*. Она при каждом обращении (на каждой итерации) возвращает следующее значение. Обычные функции возвращают всю последовательность сразу, а функция-генератор только одно значение, что повышает эффективность программы при работе с большими последовательностями данных.

Пример

```
def range1(m,n):
    while m <= m+n:
        yield m # Генерирует значение (n)
        m += 1
k=int(input("k="))
c = range1(0, k) # c=0.0
for i in range(1, k+2):
    z=next(c)/10 # следующее значение
    print(z,"   ", z*z*z-5*z*z+8*z-2.1)
```

Результат

```
k=10
0.0     -2.1
0.1     -1.3490000000000002
0.2     -0.692
0.3     -0.12300000000000022
0.4     0.3640000000000003
0.5     0.7749999999999999
0.6     1.116
0.7     1.3929999999999998
0.8     1.612
0.9     1.7790000000000004
1.0     1.9
>
```

Рекурсивные функции

Рекурсия — процесс повторения инструкций самоподобным образом. Рекурсивным называется любой объект, который частично определяется через себя. В рекурсивном определении должно присутствовать граничное условие, при выходе на которое дальнейшая инициация рекурсивных обращений прекращается.

Рекурсивные версии большинства подпрограмм могут выполняться немного медленнее, чем их итеративные эквиваленты, поскольку к необходимым действиям добавляются вызовы функций. Но в большинстве случаев это не имеет значения. Много рекурсивных вызовов в функции может привести к переполнению памяти.

Основным преимуществом применения рекурсивных функций является более простой код программы по сравнению с итеративными эквивалентами.

Рекурсия бывает прямая и косвенная.

Если вызов функции является инструкцией тела вызываемой функции, то такая рекурсия называется прямой, если через другие функции, то рекурсия называется косвенной или взаимной.

Пример (прямая рекурсия)

Вычисление факториала числа *n*.

```
def fak(n):
    if n==0:
        return 1
    else:
        return n*fak(n–1)
print(" n=",5," Факториал=", fak(5))
print(" n=",8," Факториал=", fak(8))
```

Результат

```
n= 5  Факториал= 120
n= 8  Факториал= 40320
```

Пример (прямая рекурсия)

Вычисления *n*-го числа ряда Фибоначчи.

Если нулевой элемент последовательности равен 0, первый — 1, а каждый последующий равен сумме двух предыдущих, то ряд Фибоначчи будет следующим: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, ...

```
def fib(n):
    if n==0 or n==1:
        return n
    else:
        return fib(n–1)+fib(n–2)
print(" n=",5," число Фибоначчи=", fib(5–1), ' # нумерация с 1')
print(" n=",10," число Фибоначчи=", fib(10–1),' # нумерация с 1')
print(" n=",5," число Фибоначчи=", fib(5),' # нумерация с 0')
print(" n=",10," число Фибоначчи=", fib(10),' # нумерация с 0')
```

Результат

```
n= 5  число Фибоначчи= 3  # нумерация с 1
n= 10  число Фибоначчи= 34  # нумерация с 1
n= 5  число Фибоначчи= 5  # нумерация с 0
n= 10  число Фибоначчи= 55  # нумерация с 0
```

Пример (косвенная рекурсия)

```
def f(n):
    print("f: n="," ",n,"**",end=" ")
    if n > 0: g(n – 1)
```

```
def g(n):
    print("g: n="," ",n, "&",end=" ")
    if n > 1: f(n – 2)
f(8)
```

Результат

```
f: n=  8 ** g: n=  7 & f: n=  5 ** g: n=  4 & f: n=  2 ** g: n=  1 &
```

Файлы. Работа с файлами

Длительное хранение данных можно осуществить двумя способами:

- запись в файл — используется для хранения информации относительно небольшого объема;
- хранение в базе данных.

Под файлом понимают способ хранения данных на внешнем устройстве.

В Python работа с файлами осуществляется с помощью функций, в которых запрограммированы все необходимые действия. Они позволяют работать с различными устройствами: коммуникационными каналами, дисками, принтерами, клавиатурой и т. д. Эти устройства сильно отличаются друг от друга, однако файловая система преобразует их в единое абстрактное логическое представление, называемое потоком (абстрактный канал связи, создаваемый в программе для обмена данными). С физической точки зрения файл – именованная совокупность данных, находящаяся на внешнем устройстве и имеющая определенные атрибуты (характеристики).

Файл, рассматриваемый как последовательность строк символов, называется текстовым. Его можно создавать и редактировать с помощью любого текстового редактора.

Работа с файлами включает следующие действия:

- открытие файла;
- чтение из существующего файла или запись в файл;

- закрытие файла.

Открытие файла

Открытие файла выполняется с помощью функции *open*().

Синтаксическая конструкция с обязательными параметрами:

open(<имя файла>,<режим открытия>,<кодировка>)

Имя файла задается в виде строки знаков и может быть относительным или с указанием полного пути. Если файл находится в текущем каталоге, то достаточно указать только имя файла.

Режимы открытия файла:

'r' — открытие на чтение (является значением по умолчанию);

'w' — открытие на запись, содержимое файла удаляется, если файл не существует, то создается новый;

'b' — открытие в двоичном режиме;

't' — открытие в текстовом режиме (является значением по умолчанию);

'+' — открытие на чтение и запись.

Режимы могут быть объединены, например,*'rb*' — чтение в двоичном режиме (по умолчанию режим *'rt'*).

Параметр <кодировка> используется только при открытии для чтения текстового файла (по умолчанию кодировка UTF-8).

Методы для работы с файлами

Метод	Описание
file.close()	закрывает открытый файл
file.next()	возвращает следующую строку файла
file.read(*n*)	чтение первых *n* символов файла
file.readline(*n*)	читает одну строку с номером *n*
file.readlines()	читает и возвращает список всех строк в файле
file.seek(смещение[, позиция])	устанавливает указатель в позицию, имеющую смещение относительно позиции
file.tell()	возвращает текущую позицию в файле
file.write(строка)	добавляет строку *str* в файл

Продолжение табл.

file.writelines (последовательность)	добавляет последовательность строк в файл

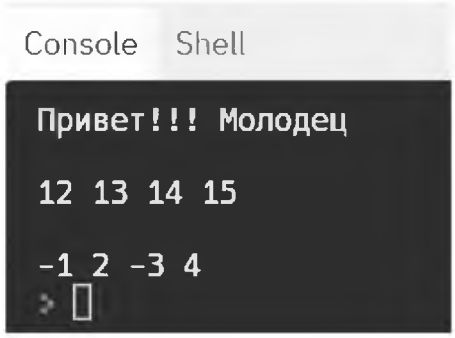
Примеры

Чтение из файла *test.txt*, который загружен в текущий каталог, и вывод на экран

```
for line in open("test.txt"):
    print (line)
```

Содержимое файла *test.txt*
Привет!!! Молодец
12 13 14 15
–1 2 –3 4

Результат



В текущей папке создать файл с именем «*out.txt*», содержащий значения квадратных корней для чисел от 10 до 20. До запуска программы файла «*out.txt*» в каталоге нет.

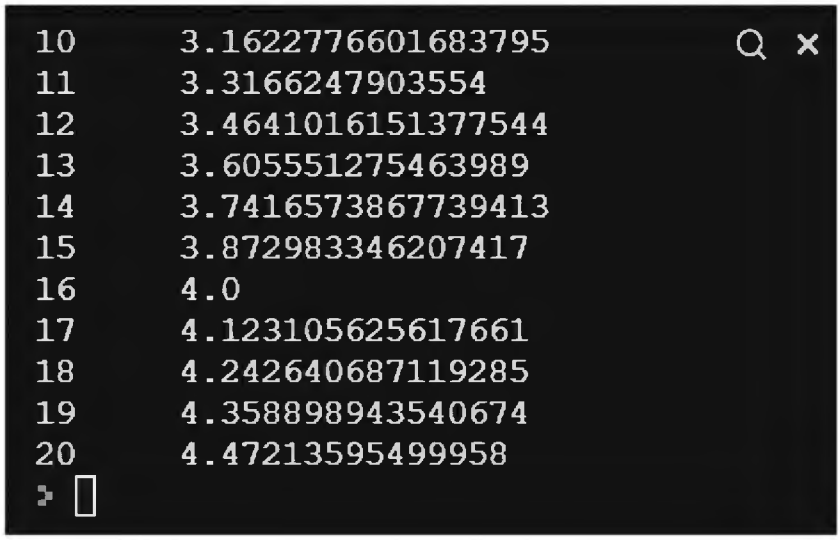
Программа

```
f = open("out.txt","w") # Открывает файл для записи
a=9
b=20
p=1
```

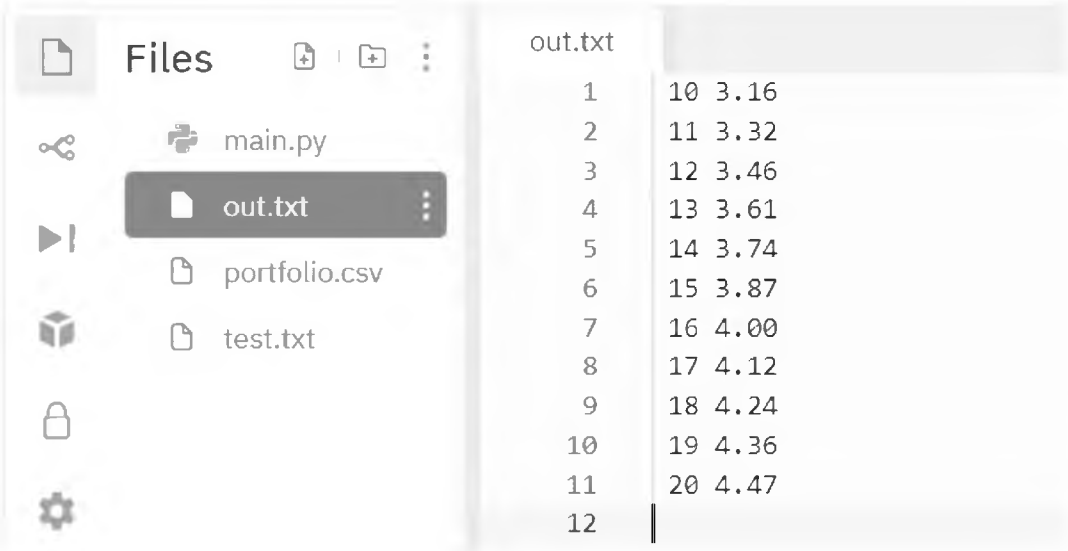
```
r=0.5
while a < b:
    p = (a+1)**r
    f.write("%3d %0.2f\n" % (a+1,p))
    a += 1
    print( a,"   ",p)
f.close()
```

Результат

Вывод на экран:

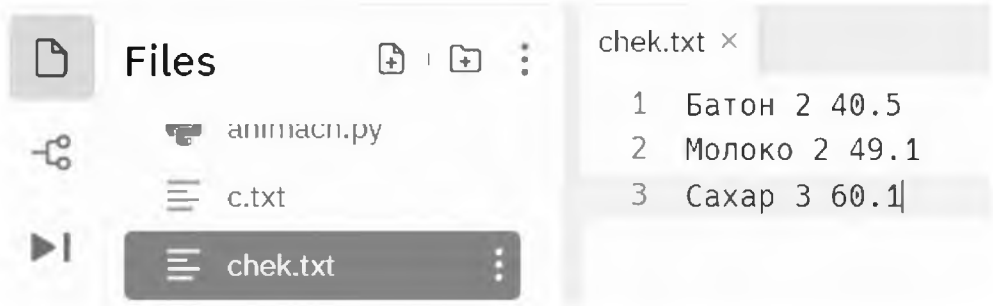


Файл *"out.txt"*



Программа предназначена для формирования стоимости и печати чека покупки. Сведения о продажах читаются из файла *chek.txt*. Структура записи: <Товар> <Количество> <Цена> (значения отделяются друг от друга пробелом).

Содержимое файла:



Программа

```
filename= "chek.txt"
ch = []
total = 0.0
rr=[]
print ("{0:7} {1:10} {2:5}".format("Товар","Кол-во",  "Цена"))
for line in open(filename):
  fields= line.split(" ") # Преобразует строку в список
  n = fields[0] # Извлекает и преобразует отдельные значения полей
  s = int(fields[1])
  p = float(fields[2])
  print("{0:7} {1:3} {2:11}".format(n,s,p))
  st = (n,s,p)
  ch.append(st)
print("----Чек----")
for n, s, p in ch:
    r=float(s * p)
    rr.append(r)
    print(n,"  ",r)
    total += s * p
print("Сумма=",total)
```

Результат

Исключения

Для обработки особых ситуаций (таких как деление на ноль или попытка чтения из несуществующего файла) применяется механизм исключений. Перехватывать исключения можно с помощью блоков *try ... except*, которые имеют следующий синтаксис:

try:
<операторы> # исключения отслеживаются и обрабатываются
except <имя исключения 1>*:* <обработка ситуации>

except <имя исключения *N* >*:* <обработка ситуации>
else:
<обработка ситуации>
finally:
<обработка ситуации>

Эта конструкция должна содержать хотя бы один блок *except*, а блоки *else* и *finally* являются необязательными. Блок *else_* выполняется только если в операторах блока *try* не возникает исключительных ситуаций. Если блок *finally* присутствует, он выполняется всегда и в последнюю очередь.

Пример инструкции *try-except:*

```
try:
   res = int(open('a.txt').read()) / int(open('c.txt').read())
   print (res)
except IOError:
   print ("Ошибка ввода-вывода")
except ZeroDivisionError:
   print ("Деление на 0")
except KeyboardInterrupt:
   print ("Прерывание с клавиатуры")
except:
   print ("Ошибка")
try:
   res = int(open('a.txt').read()) / int(open('c.txt').read())
   print res
except IOError:
   print "Ошибка ввода-вывода"
except ZeroDivisionError:
   print "Деление на 0"
except KeyboardInterrupt:
   print "Прерывание с клавиатуры"
except:
   print "Ошибка"
```

В этом примере берут числа из двух файлов и делят одно на другое. В результате этих действий могут возникать различные исключения, имена тех, которые из них могут быть обработаны, указываются в инструкциях *except*. Последняя инструкция *except* в этом примере отслеживает все другие исключения, которые не были упомянуты ранее.

Исключения рассматриваются как тип данных в Python. Исключения необходимы для того, чтобы сообщать об ошибках, возникаю-

щих при выполнении программы и приводящие к невозможности ее дальнейшей корректной работы.

- *ArithmeticError* — арифметическая ошибка
  - *FloatingPointError* — порождается при неудачном выполнении операции с плавающей запятой.
  - *OverflowError* — возникает, когда результат арифметической операции слишком велик для представления.
  - *ZeroDivisionError* — деление на ноль.
- *MemoryError* — недостаточно памяти.
- *NameError* — не найдено переменной с таким именем.
- *RuntimeError* — возникает, когда исключение не попадает ни под одну из других категорий.
- *ValueError* — функция получает аргумент правильного типа, но не корректного значения.

Пример (калькулятор)

```
try:
  n=float(input("  n="))
  m=float(input("  m="))
  result=0.0
  for op in ["*","+",":","div","^","/"]:
    #print("n",op,"m =", "  ")
    if op == "+": result = n+m
    elif op == "–": result = n–m
    elif op == "*": result = n*m
    elif op == ":": result = n/m
    elif op == "div": result = n//m
    elif op == "^": result = m**n
    else: raise RuntimeError
    print("n",op,"m =", result)
    print ("------------------------------")
except RuntimeError:
     print ("n",op,"m =","Калькулятор не знает этой операции")
```

```
except ZeroDivisionError:
    print ("Деление на 0")
except KeyboardInterrupt:
    print (" Прерывание с клавиатуры") #<Ctrl>+<c>
except :
    print ("Ошибка ввода")
```

Результаты

```
n=7.1
m= Прерывание с клавиатуры
>
```

Вместо ввода значения переменной *m*, была нажата комбинация *<Ctrl>+<c>*.

```
  n=6.2
  m=2.1
n * m = 13.020000000000001
------------------------------
n + m = 8.3
------------------------------
n : m = 2.9523809523809526
------------------------------
n div m = 2.0
------------------------------
n ^ m = 99.48546343115241
------------------------------
n / m = Калькулятор не знает этой операции
>
```

Понятие модуля

Модуль оформляется в виде отдельного файла с исходным кодом. В модуль включаются функции и классы, имеющие определенную область действия. Например, модули, реализующие различные математические функции, модули для работы с таймером, модули, генерирующие псевдослучайные значения, модули для построения графических объектов и т. п.

Подключение модуля к программе на Python осуществляется с помощью инструкции *import*.

Первый вариант:

import < имя модуля >

Пример

import math

или

import < имя модуля > *as* < новое имя модуля >

Пример

import math as pas

Второй вариант:

from < имя модуля > *import* < имена объектов модуля >

Пример

from sys import argv, environ
*from string import **

С помощью первого варианта с текущей областью видимости связывается только имя, ссылающееся на объект модуля, а при использовании второй — указанные имена (или все имена, если применена «*») объектов модуля связываются с текущей областью видимости. При импорте можно изменить имя, с которым объект будет связан, с помощью *as*.

В первом случае в программе известно только имя самого модуля, поэтому при обращении к функции, входящей в модуль, указывается имя, состоящее из имени модуля, точки и имени самой функции *math.sin*(*x*).

Во втором случае имена функций используются так, как если бы они были определены в текущем модуле, то есть достаточно указать только имя самой функции *sin*(*x*).

Повторный импорт модуля происходит гораздо быстрее, так как модули кэшируются интерпретатором. Загруженный модуль можно загрузить еще раз (например, если модуль изменился на диске) с помощью функции *reload*():.

Пример

```
import mymodule
. . .
reload(mymodule)
```

Генерация псевдослучайных чисел. Модуль random

Python использует в качестве основного генератора алгоритм вихрь Мерсе́нна. Вихрь Мерсенна (*англ. Mersenne twister*, *MT*) — генератор псевдослучайных чисел (ГПСЧ), разработанный в 1997 г. японскими учёными Макото Мацумото и Такудзи Нисимура. Вихрь Мерсенна основывается на свойствах простых чисел Мерсенна и обеспечивает быструю генерацию высококачественных по критерию случайности псевдослучайных чисел. Это один из наиболее широко протестированных генераторов случайных чисел, однако он не подходит для криптографии.

В модуль *random* включены следующие функции:

— *random.random* — возвращает псевдослучайное число от 0.0 до 1.0;

— *random.uniform*(*a*, *b*) — возвращает случайное число *N* с плавающей точкой таким образом, чтобы *a* <= *N* <= *b* для *a* <= *b* и *b* <= *N* <= *a* для *b* < *a*;

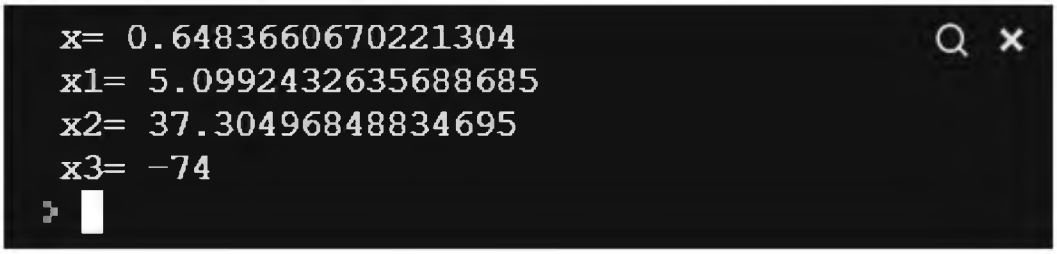
— *random.triangular*(*a*, *b*, *m*) – возвращает случайное число с плавающей точкой *N*, так, что *a* <= *N* <= *b* и с указанным *m* между этими границами. Границы *a* и *b* по умолчанию равны 0 и 1;

— *random.randint*(*a*, *b*) – возвращает случайное целое число *N* так, чтобы *a* <= *N* <= *b*.

Пример

```
import random
x=random.random()
print(" x=",x)
x1=random.uniform(2.5, 15.6)
print(" x1=",x1)
x2=random.triangular(20, 60, 35)
print(" x2=", x2)
x3=random.randint(–100, 101)
print(" x3=", x3)
```

Результат



Типы коллекций

Последовательности — один из типов данных, которые используются в программах на Python для реализации массивов (в частности, для моделирования векторов и матриц). К последовательностям относятся также строки знаков.

Последовательности поддерживают инструкции:

- проверки на вхождение *in*;
- функцию определения размера *len*();
- оператор извлечения срезов [];

- возможность выполнения итераций.

К типичным последовательностям относятся: списки, кортежи, словари.

Списки. Функция list()

Списки в Python — упорядоченные изменяемые последовательности объектов произвольных типов (почти как массив, но типы могут отличаться).

Их можно задавать с помощью литералов, записываемых в квадратных скобках, или посредством списковых включений. Пустой список создается с помощью пары пустых квадратных скобок [], а список, состоящий из одного или более элементов, может быть создан с помощью последовательности элементов, разделенных запятыми и заключенных в квадратные скобки.

Пример

```
lst1 = [1, 2, 3]
lst2 = [x**2 for x in range(10) if x % 2 == 1]
lst3 = list("abcde")
print(lst1)
print(lst2)
print(lst3)
```

Результат

```
[1, 2, 3]
[1, 9, 25, 49, 81]
['a', 'b', 'c', 'd', 'e']
```

Функция *list*() — без аргументов возвращает пустой список; с аргументом типа *list* возвращает копию аргумента; в случае если ар-

гумент имеет другой тип (как в предыдущем примере *str*), то выполняется преобразование его в объект типа *list*.

Взятие элемента по индексу

В языке Python нумерация элементов списка (последовательности) начинается с нуля. При попытке доступа к несуществующему индексу возникает исключение *IndexError*.

Пример

```
z=[2, 4, –5,"ok"]
print ("z[0]=",z[0])
print ("z[2]=",z[2])
print ("z[3]=",z[3])
print ("z[-2]=",z[–2])
print ("z[-1]=",z[–1])
```

Результат

В данном примере переменная *z* является списком, однако взять элемент по индексу можно и у других типов: строк, кортежей. В Python также поддерживаются отрицательные индексы, при этом нумерация идёт с конца *z*[–1] и *z* [–2].

Методы списков:

Метод	Описание
list.append(*x*)	добавляет элемент в конец списка
list.extend(*L*)	расширяет список, добавляя в конец все элементы списка
list.insert(*i*, *x*)	вставляет на *i*-й элемент значение *x*

Продолжение табл.

list.remove(*x*)	удаляет первый элемент в списке, имеющий значение
list.pop([*i*])	удаляет *i*-й элемент и возвращает его. Если индекс не указан, удаляется последний элемент
list.index(*x*, [*n*[, *m*]])	возвращает положение первого элемента от *n* до *m* со значением *x*
list.count(*x*)	возвращает количество элементов со значением *x*
list.sort([*key* =функция])	сортирует список на основе функции
list.reverse()	разворачивает список
list.copy()	копия списка
list.clear()	очищает список

Пример

```
a = [66.25, 333, 333, 1, 1234.5]
print(" a=",a)
print(a.count(333), a.count(66.25), a.count('x'))
a.insert(2, -1)
print(" a1=",a)
a.sort()
print(" a2=",a)
a.reverse()
print(" a3=",a)
b=a
b.reverse()
print(" b1=",b)
b.pop(-4)
print(" b2=",b)
b.extend(["ok","go"])
print(" b3=",b)
```

Результат

```
 a= [66.25, 333, 333, 1, 1234.5]
2 1 0
 a1= [66.25, 333, -1, 333, 1, 1234.5]
 a2= [-1, 1, 66.25, 333, 333, 1234.5]
 a3= [1234.5, 333, 333, 66.25, 1, -1]
 b1= [-1, 1, 66.25, 333, 333, 1234.5]
 b2= [-1, 1, 333, 333, 1234.5]
 b3= [-1, 1, 333, 333, 1234.5, 'ok', 'go']
>
```

Функции range() и списки

Функция *range*() уже упоминалась при рассмотрении цикла *for*. Эта функция принимает от одного до трех аргументов. Если аргумент всего один, она генерирует список чисел от 0 до заданного числа минус 1. Если аргументов два, то список начинается с числа, указанного первым аргументом. Если аргументов три — третий аргумент задает шаг списка.

Пример

```
z=list(range(10))
z1=list(range(4, 10))
z2=list(range(2, 10, 3))
print ("z=",z)
print ("z1=",z1)
print ("z2=",z2)
```

Результат

```
z= [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
z1= [4, 5, 6, 7, 8, 9]
z2= [2, 5, 8]
>
```

Кортежи. Функция tuple()

Кортеж представляет собой неизменяемый список. Используется для представления константной последовательности (разнородных) объектов. Литерал кортежа обычно записывается в круглых скобках, но можно, если не возникает неоднозначности, писать и без них.

Преимущества кортежей:

- являясь константной последовательностью, они защищены от случайных изменений (защита от «дурака»);
- для размещения кортежей в памяти требуется меньше места, чем для списков.

Пример

```
a = (1, –2, 3, –4, 5, –6, 7, –8, –9, 0)
b = [1, –2, 3, –4, 5, –6, 7, –8, –9, 0]
print("Для a=", a.__sizeof__())
print("Для b=", b.__sizeof__())
```

Результат

```
Для a= 104
Для b= 120
>
```

Создание кортежей:

- создать пустой кортеж можно с помощью функции *tuple*();
- создать кортеж можно с помощью круглых скобок и обязательной запятой;
- создать кортеж можно с помощью с помощью функции *tuple*(аргумент).

Пример

```
a = tuple() # функцией tuple()
a1 = ('s', ) # запятая обязательно!
```

*a*2 = *tuple*('Свобода — это осознанная необходимость!')
print("*a*= ",*a*)
print("*a*1= ",*a*1)
print("*a*2= ",*a*2)

Результат

```
a=  ()
a1=  ('s',)
a2=  ('С', 'в', 'о', 'б', 'о', 'д', 'а', ' ',
'–', ' ', 'э', 'т', 'о', ' ', 'о', 'с', 'о', '
з', 'н', 'а', 'н', 'н', 'а', 'я', ' ', 'н', 'е
', 'о', 'б', 'х', 'о', 'д', 'и', 'м', 'о', 'с'
, 'т', 'ь', '!')
```

Доступ к элементам кортежа осуществляется так же, как к элементам списка, — через указание индексов и срезов.

Пример

a = (1, 2, 3, 4, 5)
print(" *a*[0]=",*a*[0])
print(" *a*[1:3]=",*a*[1:3])

Результат

```
a[0]= 1
a[1:3]= (2, 3)
```

Работа с кортежами во многом совпадает с работой со списками. В частности, определены все операции над списками, не изменяющие список (сложение, умножение на число, методы *index*() и *count*()).

С помощью очень простой инструкции можно поменять кортежи местами.

Пример

```
a = tuple("Good morning!")
b=(10, 20, 30, 40, –50, –70)
print ("Исходные значения a=", a)
print ("                b=",b)
print(" Размер a=", a.__sizeof__())
print(" Размер b=", b.__sizeof__())
a, b = b, a
print ("После перестановки a=", a)
print ("                b=",b)
print(" Размер a=", a.__sizeof__())
print(" Размер b=", b.__sizeof__())
```

Результат

```
Исходные значения a= ('G', 'o', 'o', 'd', ' ', 'm', 'o', 'r', 'n', 'i', 'n', 'g', '!')
                    b= (10, 20, 30, 40, –50, –70)
Размер a = 128
Размер b = 72
После перестановки a= (10, 20, 30, 40, –50, –70)
                     b= ('G', 'o', 'o', 'd', ' ', 'm', 'o', 'r', 'n', 'i', 'n', 'g', '!')
Размер a= 72
Размер b= 128
```

Удалить отдельные элементы из кортежа невозможно, но можно удалить кортеж целиком с помощью инструкции *del*.

Пример

```
b=(10, 20, 30, 40, –50, –70)
print(" b=",b)
del b
```

В Python реализована возможность взаимного преобразования списков и кортежей.

Пример

```
lst = [1, 2, 3, 4, 5]
print(type(lst))
print("  lst=",lst)
tpl = tuple(lst)
print(type(tpl))
print("  tpl= ",tpl)
```

Результат

```
<class 'list'>
   lst= [1, 2, 3, 4, 5]
<class 'tuple'>
  tpl=  (1, 2, 3, 4, 5)
```

Кортежи могут содержать списки. Список изменяем, кортеж нет. Это правило действует и при вложении списков. Элементы вложенных списков можно изменять.

Пример

```
n = (–3.5, "1,5", ["spicok", 1, "a"])
print("1.  n=",n)
n[2][2]="new"
print("2.  n=",n)
```

Результат

```
1.  n= (-3.5, '1,5', ['spicok', 1, 'a'])
2.  n= (-3.5, '1,5', ['spicok', 1, 'new'])
```

Словари. Функция dict()

Словари — неупорядоченные коллекции произвольных объектов с доступом по ключу. Их иногда ещё называют ассоциативными массивами или хеш-таблицами.

Чтобы работать со словарём, его нужно создать. Сделать это можно несколькими способами:

- с помощью литерала;
- с помощью функции *dict*();
- с помощью метода *fromkeys*();
- с помощью генераторов словарей.

Пример

```
d = {}
print(" d1=",d)
d ={'dict': 1, 'dictionary': 2}
print(" d2=",d)
d = dict([(1, 1), (2, 4)])
print(" d3=",d)
d = dict.fromkeys(['a', 'b'])
print(" d4=",d)
d = {a: a ** 2 for a in range(7)}
print(" d5=",d)
```

Результат

```
d1= {}
d2= {'dict': 1, 'dictionary': 2}
d3= {1: 1, 2: 4}
d4= {'a': None, 'b': None}
d5= {0: 0, 1: 1, 2: 4, 3: 9, 4: 16, 5: 25, 6:
36}
```

Присвоение по новому ключу расширяет словарь, присвоение по существующему ключу изменяет значение с этим ключом, а попытка извлечения несуществующего ключа порождает исключение.

Пример

```
d = {1: 2, 2: 4, 3: 9}
print(" d[1]= ", d[2])
d[4] = –4 ** 2
print(" d=", d)
s={'a':"qw", 'b':"rt"}
print(" s=", s)
s['b']="NNN"
print(" s=", s)
```

Результат

```
d[1]=  4
 d= {1: 2, 2: 4, 3: 9, 4: -16}
 s= {'a': 'qw', 'b': 'rt'}
 s= {'a': 'qw', 'b': 'NNN'}
>
```

Методы словарей:

Метод	Описание
dict.clear()	очищает словарь
dict.copy()	возвращает копию словаря
dict.fromkeys(*seq*[, *value*])	создает словарь с ключами из *seq* и значением *value* (по умолчанию *None*)
dict.get(*key*[, *default*])	возвращает значение ключа
dict.items()	возвращает пары (ключ, значение)
dict.keys()	возвращает ключи в словаре
dict.pop(*key*[, *default*])	удаляет ключ и возвращает значение
dict.popitem()	удаляет и возвращает пару (ключ, значение)
dict.setdefault(*key*[, *default*])	возвращает значение ключа

Продолжение табл.

dict.update([*other*])	обновляет словарь, добавляя пары (ключ, значение) из *other*. Существующие ключи перезаписываются
dict.values()	возвращает значения в словаре

Пример

```
z={'a':123,'b':45}
print(" z=",z)
sil = input('ссылка: ')
name = input('название: ')
z.update([(sil, name)])
print(" z1=",z)
t=z.items()
print(" z2=",t)
z.pop('b')
print(" z3=",z)
p={'1': 555,'2':999}
z.update(p)
print(" z4=",z)
```

Результат

```
 z= {'a': 123, 'b': 45}
ссылка: uu
название: Привет!
 z1= {'a': 123, 'b': 45, 'uu': 'Привет!'}
 z2= dict_items([('a', 123), ('b', 45), ('uu',
 'Привет!')])
 z3= {'a': 123, 'uu': 'Привет!'}
 z4= {'a': 123, 'uu': 'Привет!', '1': 555, '2'
: 999}
>
```

Примеры решения задач для освоения основных инструкций Python

Линейные программы

Задача 1

Вычислить значения Y и F для заданных значений x, a, b.

$$Y = (x - a) \cdot arcctg(a + x) - \sqrt[3]{|x + a|} + x \cdot \ln(a + x)$$

$$F = \sqrt{b + x + a \cdot x^2} - e^{-a+x} + a \cdot x \cdot \ln(x + a)$$

$a = 2$, $b = 0{,}5$, при $x = 3$ и $x = 1{,}5$.

Решение задачи

Формализация задачи

Дано: x, a, b.

Найти: Y и F.

Программа

```
import math
x=float(input("  x="))
a=2; b=0.5
Y=(x-a)/math.atan(1/(a+x))–math.exp(1.0/3*math.log(abs(x+a)))
+x*math.log(a+x)
F=math.sqrt(a*x*x+b+x)–math.exp(-a+x)+a*x+math.log(a+x)
print ("  Y=",Y)
print ("  F=",F)
```

Результат

для $x = 3$

x = 3

Y= 8.184307869995674

F= 9.527965331722907,

для $x = 1{,}5$

x = 1.5

Y= –1.4357745845414875

F= 6.195742065579127

Задача 2

Смешаны *V*1 литр воды температуры *T*1 с *V*2 литрами воды температуры *T*2. Написать программу вычисления объема и температуры воды после смешения.

Решение задачи

Формализация задачи

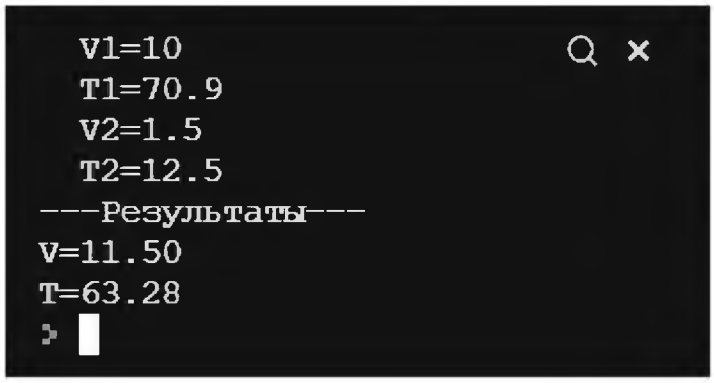
Дано: *V*1, *T*1, *V*2, *T*2.

Найти: *V* и *T*.

Программа

```
V1=float(input("  V1="))
T1=float(input("  T1="))
V2=float(input("  V2="))
T2=float(input("  T2="))
V=V1+V2
T=(V1*T1+V2*T2)/V
print ("---Результат---")
print ('{:s}{:.2f}'.format("V=",V))
print ('{:s}{:.2f}'.format("T=",T))
```

Результат



Ветвления

Задача 3

Заданы три целых числа *a*, *b* и *c*. Найти наибольшее из них кратное трем. Если среди *a*, *b* и *c* нет значений кратных трем, то вывести *NO* и найти минимальное среди значений *a* + *b*, *b* + *c* и *a* + *c*.

Решение задачи

Формализация задачи

Дано: *a*, *b* и *c*.

Найти: *m*.

Программа

```
a=int(input(" a="))
b=int(input(" b="))
c=int(input(" c="))
m=0;
bl=True
l=la=lb=lc=0
if a%3==0: la=1
if b%3==0: lb=1
if c%3==0: lc=1
if la+lb+lc>=1:
 if la==1:
  if bl: m=a; bl=False
 if lb==1:
  if bl:
   m=b;
   bl=False
  if b>m: m=b
 if lc==1:
   if bl:  m=c;
   if c>m: m=c
 else:
```

```
  l=1
  m=a+b
  if b+c<m: m=b+c
  if a+c<m: m=a+c
print ("---Результат---")
if l==1: print ("No")
print ('{:.2f}'.format(m))
```

Результаты

Выводит максимальное среди кратных трем.

Кратных значений два — *a* и *c*:

```
 a=21
 b=161
 c=333
---Результаты---
333.00
```

Кратных значений два — *a* и *b*:

```
 a=-15
 b=33
 c=62
---Результаты---
33.00
```

Кратных значений три — *a*, *b* и *c*:

```
 a=-12
 b=-45
 c=-3
---Результаты---
-3.00
```

Значений кратных трем нет, выводит минимальное среди значений $a + b$, $b + c$ и $a + c$:

Циклы

Задача 4

Построить таблицу значений альтернативно заданной функции $f(d)$:

$$f = \begin{cases} \dfrac{ln\sqrt{-d}}{\sqrt{-d}} & \text{при } d < 0 \\ \dfrac{1}{b+c} & \text{при } d = 0 \\ \dfrac{arctg\left(\dfrac{b+c}{\sqrt{d}}\right)}{\sqrt{d}} & \text{при } d > 0 \end{cases}$$

d изменяется от начального значения dn до конечного dk с шагом dh. Значения b и c заданы.

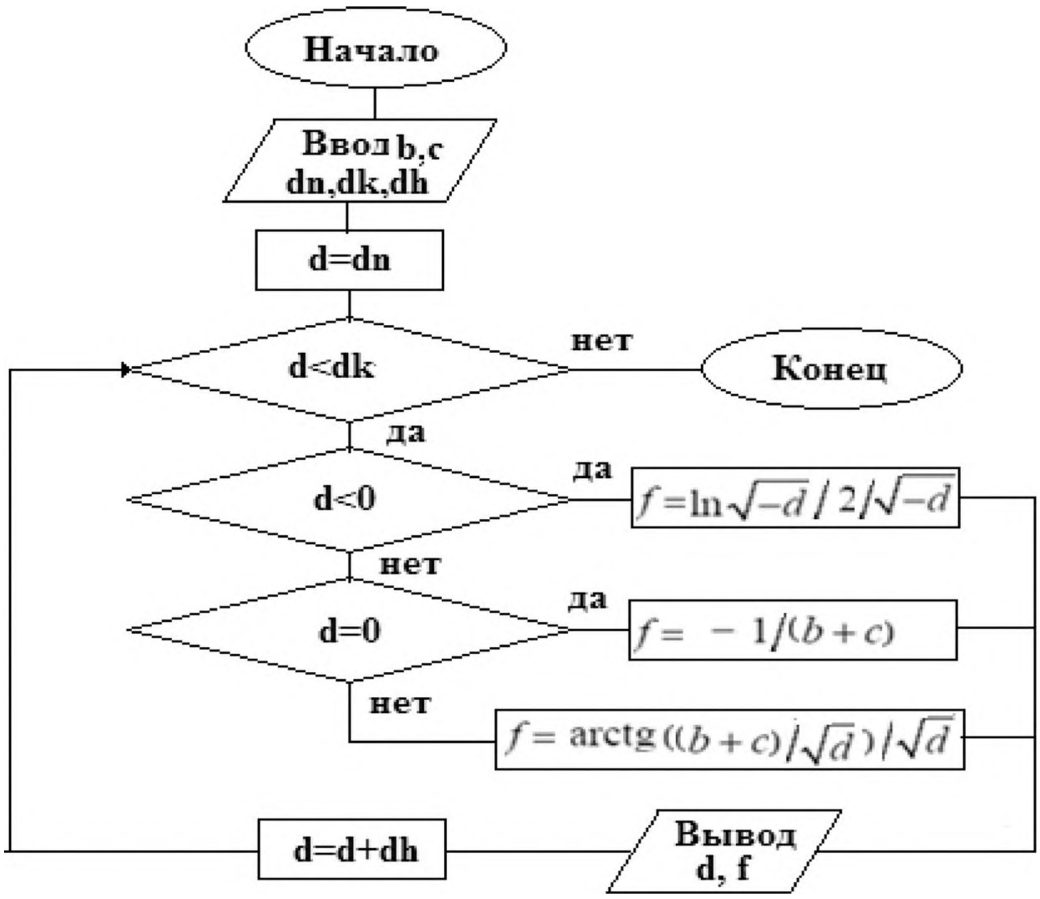
Решение задачи

Формализация задачи

Дано: $b = 1.7$, $c = 0.4$, dn, dk, dh.

Найти: f.

Блок-схема



На первый взгляд кажется, что блок-схема абсолютно правильная. Однако написав программу и выполнив расчет, например, при следующих исходных данных: $dn = -1$, $dk = 2$, $dh = 0.2$, увидим неверный результат для значения 0. Эта ситуация возникает из-за того, что количество знаков во внутреннем представлении вещественных чисел ограничено и присутствует вычислительная ошибка (было рассмотрено в примере табулирования функции). Правильно эта программа работает только для целых значений аргумента функции. Один из возможных вариантов решения проблемы реализован в тексте следующей программы.

Программа

```
import (input(" dn="))
dk=float(input(" dk="))
```

```
dh=float(input(" dh="))
d=dn; b=1.7; c=0.4
print ("---Результат---")
while d<dk+dh*0.1:
 if d<–0.1*dh: f=math.log(–d)/(2*math.sqrt(–d))
 else:
  if ((d>–0.1*dh) and (d<0.1*dh)): f=–1/(b+c)
  else: f=math.atan(1/(b+c))/math.sqrt(d)/math.sqrt(d)
 print ('{0: .2f}   {1: 0.3}'.format(d,f))
 d+=dh
```

Результат

```
 dn=-1
 dk=2
 dh=0.2
---Результаты---
-1.00    0.0
-0.80   -0.125
-0.60   -0.33
-0.40   -0.724
-0.20   -1.8
-0.00   -0.476
 0.20    2.22
 0.40    1.11
 0.60    0.741
 0.80    0.556
 1.00    0.444
 1.20    0.37
 1.40    0.317
 1.60    0.278
 1.80    0.247
 2.00    0.222
```

Задача 5

Для x, изменяющегося в интервале от $x0$ до xk с шагом h, вычислить значения бесконечной суммы:

$$S(x)=\sum_{n=0}^{\infty}\frac{(-1)^n(2x)^n}{(2n)!}$$

с точностью ε=0.00001 и функции $y(x)=\cos(\sqrt{2x})$.

Решение задачи

Формализация задачи

Дано: *x*0, *xk*, *h*.

Найти*: S*, *y*.

Данную задачу решаем с использованием рекуррентной формулы для члена ряда *S*.

Вывод рекуррентной формулы для расчета текущего члена ряда.

Формула общего члена ряда

$$u_i = \frac{-1^i \cdot (2 \cdot x)^i}{(2 \cdot i)!}$$

$$i = 0 \qquad u_0 = \frac{-1^0 \cdot (2 \cdot x)^0}{(2 \cdot 0)!} = 1$$

$$i = 1 \qquad u_1 = \frac{-1^1 \cdot (2 \cdot x)^1}{(2 \cdot 1)!} = -x$$

……..

$$i = n \qquad u_n = \frac{-1^n \cdot (2 \cdot x)^n}{(2 \cdot n)!}$$

$$i = n + 1 \qquad u_{n+1} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1}}{(2 \cdot (n+1))!} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1}}{(2 \cdot n + 2)!}$$

Найдем отношение

$$\frac{u_{n+1}}{u_n} = \frac{-1^{n+1} \cdot (2 \cdot x)^{n+1} \cdot (2 \cdot n)!}{(2 \cdot n + 2)! \cdot -1^n \cdot (2 \cdot x)^n} = -\frac{2 \cdot x}{(2 \cdot n + 1) \cdot (2 \cdot n + 2)} =$$

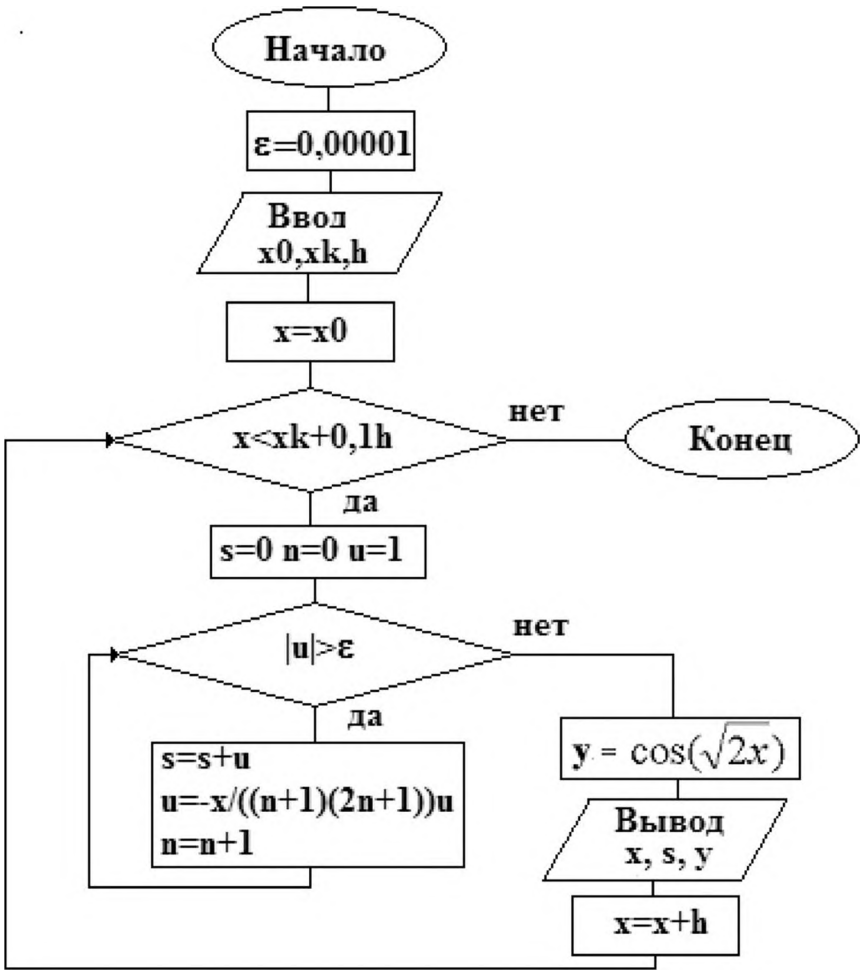
$$= -\frac{x}{(n+1) \cdot (2 \cdot n + 1)}$$

Выразим *un*+1 и получим рекуррентную формулу

$$u_{n+1} = -\frac{x}{(n+1) \cdot (2 \cdot n + 1)} \cdot \quad u_n$$

Эта формула позволяет вычислить любой текущий член ряда кроме u_0.

Блок-схема

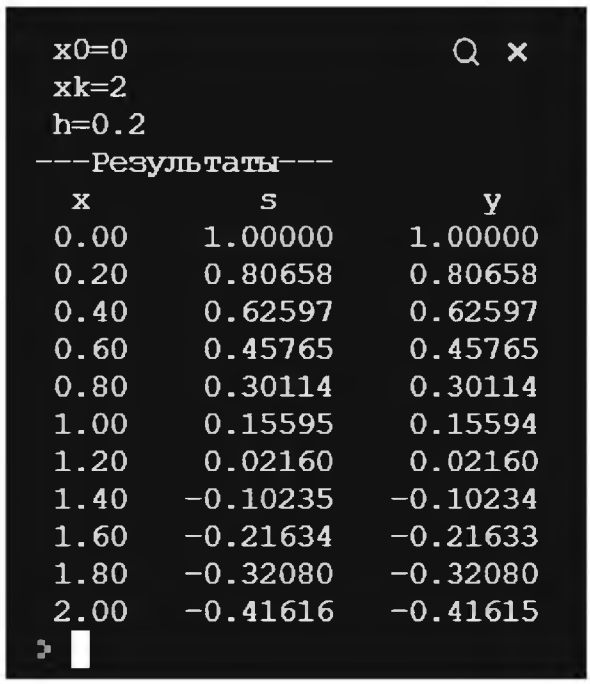


Программа

```
import math
x0=float(input(" x0="))
xk=float(input(" xk="))
h=float(input(" h="))
x=x0; e=1e–5
print ("---Результат---")
print (" x        s          y")
while x<xk+h*0.1:
  s=0; n=0;  u=1
  y=math.cos(math.sqrt(2*x))
  while abs(u)>e:
    s+=u
```

```
    u*=-x/((n+1)*(2*n+1))
    n+=1
  print ('{0: .2f}    {1: 0.5f}    {2: 0.5f}'.format(x,s, y))
  x+=h
```

Результат



```
x0=0
xk=2
h=0.2
---Результаты---
  x          s          y
0.00    1.00000    1.00000
0.20    0.80658    0.80658
0.40    0.62597    0.62597
0.60    0.45765    0.45765
0.80    0.30114    0.30114
1.00    0.15595    0.15594
1.20    0.02160    0.02160
1.40   -0.10235   -0.10234
1.60   -0.21634   -0.21633
1.80   -0.32080   -0.32080
2.00   -0.41616   -0.41615
>
```

Значения s и y должны быть приблизительно одинаковыми в данном случае с точностью 0.00001.

Последовательности (задачи с векторами и матрицами)

Задача 6

Для вектора a размерностью n вычислить вектор b по формуле:

$$b_i = \sum_{j=1}^{i} a_j$$

Найти максимальный элемент вектора b.

Векторы в программе моделируем списками.

Решение задачи

Формализация задачи

Дано: *n*, *a*.

Найти: *b* и *mb*.

Программа

```
import random
n=int(input(" n="))
a=[random.uniform(–49, 51) for i in range(0,n)]
print (" Массив a:")
for x in a:
  print ('{0:.2f} '.format(x),end=' ')
print(" ")
x=0
b=[]
for i in range (0,n):
  x+=a[i]
  b.append(x) # добавляем новый элемент в список
print ("---Результат---")
print (" Массив b:")
for x in b:
  print ('{0:.2f} '.format(x),end=' ')
print(" ")
mb=max(b)
print ("max b=  {0:.2f}".format(mb))
```

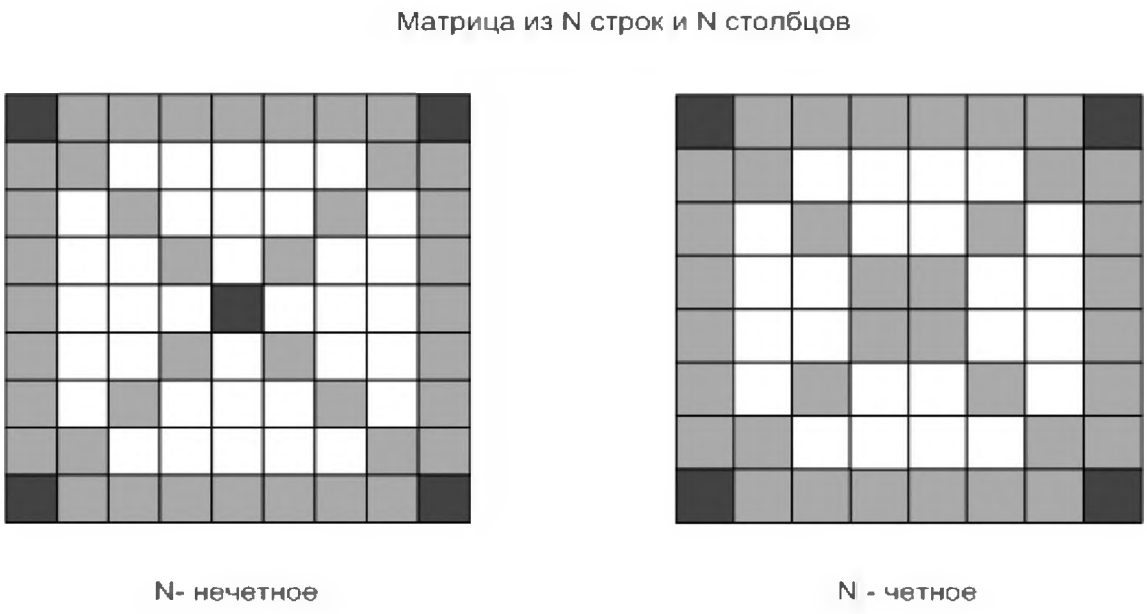
Результат

```
 n=7
 Массив a:
27.33  -24.23  49.29  -12.11  -42.27  -40.36  22.02
---Результаты---
 Массив b:
27.33  3.10  52.39  40.28  -1.99  -42.36  -20.34
max b=  52.39
>
```

Массив *a* заполняем псевдослучайными значениями, лежащими в пределах от –49 до 51.

Задача 7

Вычислить количество положительных элементов квадратной матрицы, расположенных по ее периметру и на диагоналях.



Векторы и матрицы в программе моделируем списками.

Решение задачи

Формализация задачи

Дано: n, массив a_{n*n}

Найти: k.

Программа

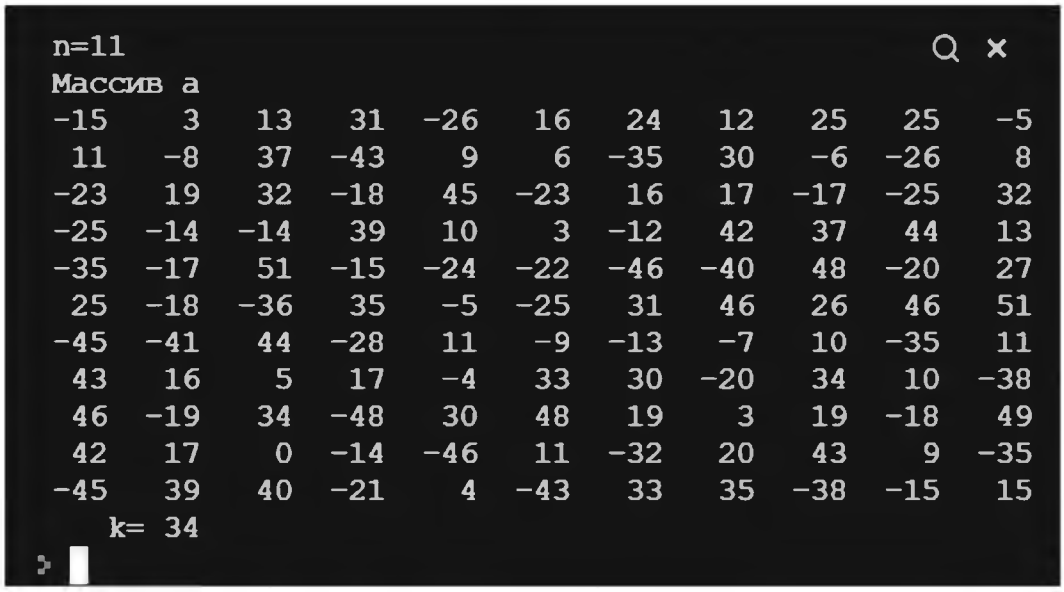
```
import random
n=int(input(" n="))
a=[]
for i in range(0, n):
  x=[random.randint (-49, 51) for j in range(0,n)]
  a.append(x)
print(" Массив a")
for i in range(0,n):
```

```
  for j in range(0,n):
    if j<n-1: print ('{:4}' .format (a[i][j]), end=' ')
    else: print ('{: 4}' .format (a[i][j]))
k=0
for i in range(0,n):
  if a[0][i]>0: k+=1
  if a[n-1][i]>0: k+=1
for i in range(1,n–1):
  if a[i][0]>0: k+=1
  if a[i][n-1]>0: k+=1
for i in range(1,n–1):
  if a[i][i]>0: k+=1
  if a[i][n-2-i+1]>0: k+=1
if (n%2==1) and (a[n//2][n//2]): k–=1
print("    k=",k)
```

Результаты

```
n=7
Массив a
 51  -32  -35   11  -31  -22   26
-36   20   27   35   -1   42   33
-19  -16   30  -15  -30  -12    2
 -4   -3   -2   23   45   11    2
 30  -34   18  -18    0  -11  -14
 34  -45   45  -39  -20   51   42
-10  -11   27   -5   14  -45   36
   k= 18
>
```

```
n=6
Массив a
 48   -3  -34  -48   17   30
 28   36  -29  -18  -27   10
  2   13  -45  -45  -36   28
 46   18   12   41  -45  -29
-31   17   42   23   49    4
-43   50  -11   44  -43   28
   k= 17
>
```



Строки

Задача 8

Дана строка знаков, содержащая текст из слов, разделенных одним пробелом, и целое число *m*. Разбить исходный текст на строки длиной *m*, переносы слов запрещены, до требуемой длины результирующие строки дополняются символами «'–'».

Решение задачи

Формализация задачи

Дано: *s* — строка знаков, длиной не более 255 символов, *m* — длина результирующих строк, *m* должно быть больше самого длинного слова в заданной строке.

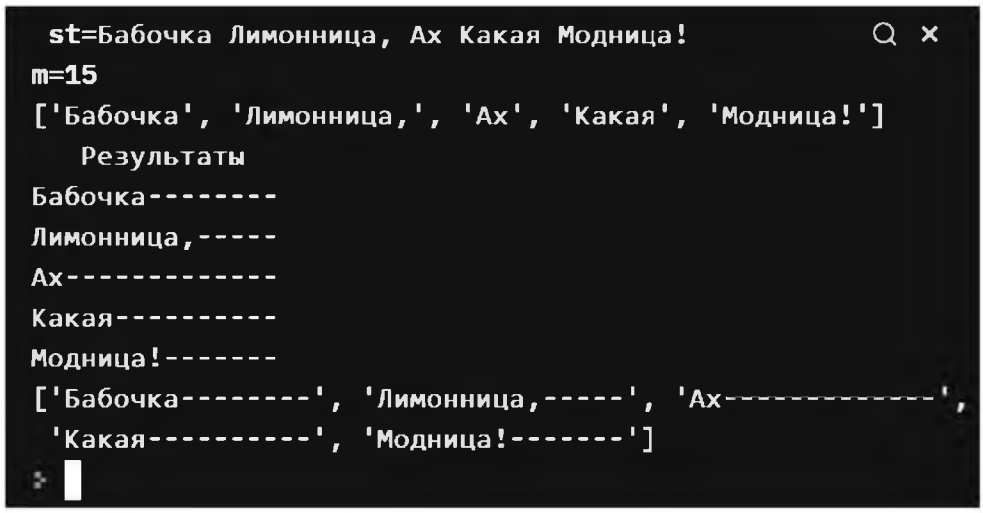
Найти: строки заданной длины.

Программа

```
st=input("st=")
m=int(input("m="))
sp=st.split()
print ("  Результат")
stroki=[]
for s in sp:
  l=len(s)
```

```
 for j in range(0,m –l):
   s+='–'
 stroki.append(s)
 print (s)
#print (stroki)
```

Результат



Задача 9

Табулирование функции $f = 1+x$ с использованием цикла *for* и строк знаков.

Решение задачи

Формализация задачи

Дано: $x0$, xk, h в виде строк

Найти: f как вещественные значения.

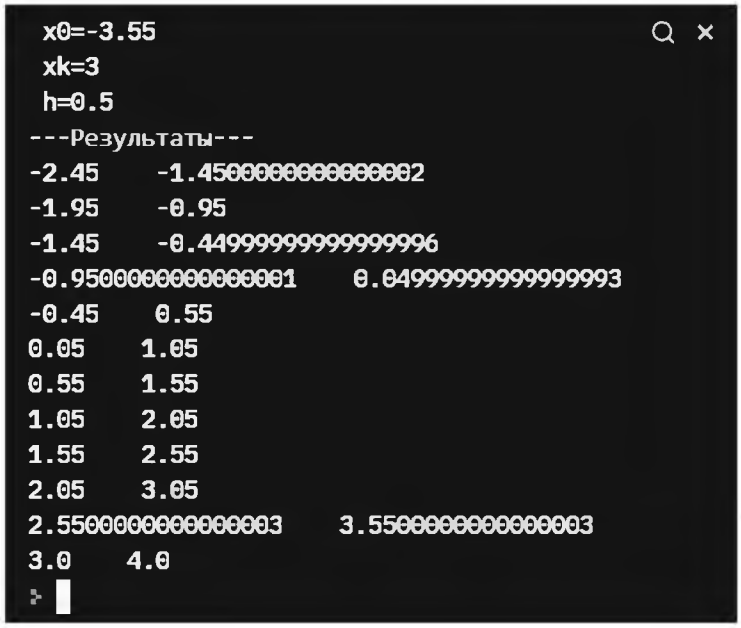
Программа

```
import math
sx0=input(" x0=")
sxk=input(" xk=")
sh= input(" h=")
p1=[]; p2=[]; p3=[]
p3=sh.partition('.')
p2=sxk.partition('.')
p1=sx0.partition('.')
```

```
l1=len(p1[2])
l2=len(p2[2])
l3=len(p3[2])
lr=max(l1,l2,l3)
if l1==0: x0=int(p1[0])*(10**lr)
else: x0=int(p1[0])*(10**lr)+int(p1[2])*10**(lr–l1)
if l2==0: xk=int(p2[0])*10**lr
else: xk=int(p2[0])*10**lr+int(p2[2])*10**(lr–l2)
if l3==0: h=int(p3[0])*10**lr
else: h=int(p3[0])*10**lr+int(p3[2])*10**(lr–l3)
print ("---Результат---")
for i in  range(x0,xk,h):
  x=i*10**(–lr)
  f=1+x
  print (x," ",f)
x=float(sxk)
f=1+x
print (x," ",f)
```

Результаты

```
 x0=-2
 xk=2
 h=0.25
---Результаты---
-2.0    -1.0
-1.75    -0.75
-1.5    -0.5
-1.25    -0.25
-1.0    0.0
-0.75    0.25
-0.5    0.5
-0.25    0.75
0.0    1.0
0.25    1.25
0.5    1.5
0.75    1.75
1.0    2.0
1.25    2.25
1.5    2.5
1.75    2.75
2.0    2.0
:
```



Функции

Задача 10

Даны натуральные числа *a*, *b*, *c*. Найти *NOD* (*a*, *b*, *c*) (наибольший общий делитель). Нахождение наибольшего общего делителя двух чисел оформить как функцию.

Решение задачи

Формализация задачи

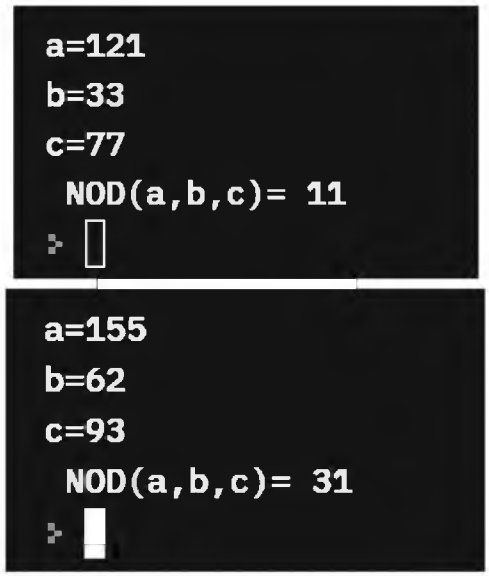
Дано: *a*, *b*, *c*.

Найти: наибольший общий делитель *a*, *b*, *c*.

Программа

```
def NOD(i,j):
  a,b=i,j
  while a!=b:
    if a>b: a=a–b
    else: b=b–a
  return a
a=int(input("a="))
b=int(input("b="))
c=int(input("c="))
print (" NOD(a,b,c)=", NOD(NOD(a,b),c)) # рекурсия
```

Результаты



Задача 11

Написать программу, содержащую функцию для чтения с клавиатуры и вывода на экран значений элементов вектора и матрицы.

Решение задачи

Программа

```
def vvod (n, a,b):
  n=int(input("n="))
  a = [[0]*n for i in range(n)]
  b=[0 for i in range(n)]
 for i in range(0,n):
   for j in range (0,n):
     print("a[{0:2},{1:2}]=".format(i,j),end=' ')
     a[i][j]=float(input())
   print("b[{0:2}]=".format(i),end=' ')
   b[i]=float(input())
  return n, a,b

a=[[]]
b=[]
n=0
```

```
n,a,b=vvod (n, a, b)
print (n)
print(a)
print (b)
```

Результат

```
n=2
a[ 0, 0]= 11
a[ 0, 1]= 12
b[ 0]= 321
a[ 1, 0]= 21
a[ 1, 1]= 22
b[ 1]= –98
2
[[11.0, 12.0], [21.0, 22.0]]
[321.0, –98.0]
```

Работа с файлами Excel. Модуль pandas. DataFrame

Для чтения и записи файлов *Excel* можно использовать различные возможности Python. Рассмотрим, как это делается с помощью модулей *pandas* и объектов *DataFrame*. Более подробное описание приведено по ссылке: *https://pythonru.com/uroki/chtenie-i-zapis-fajlov-excel-xlsx-v-python*.

Задача 12

Создать файл *tab.xlsx* с результатом табулирования функции:

f(*x*)=(*x*-*a*)/*math.atan*(1/(*a*+*x*))-*math.exp*(1/3**math.log*(*abs*(*x*+*a*))) +*x***math.log*(*a*+*x*) от *xn* до *xk* с шагом *h*.

Программа

```
import pandas as pd
import math
xn=float(input("xn="))
```

```
xk=float(input("xk="))
h=float(input("h="))
a=float(input("a="))
x=xn
ax=[]
bf=[]
while x<=xk :
  ax.append(x)
  z=(x-a)/math.atan(1/(a+x))-math.exp(1/3*math.log(abs(x+a)))
+x*math.log(a+x)
  bf.append(z)
  x+=h
df = pd.DataFrame({'x': [x for x in ax],'f(x)':[f for f in bf]})
print (df)
df.to_excel('tab.xlsx',sheet_name='Табулирование', index=False)
```

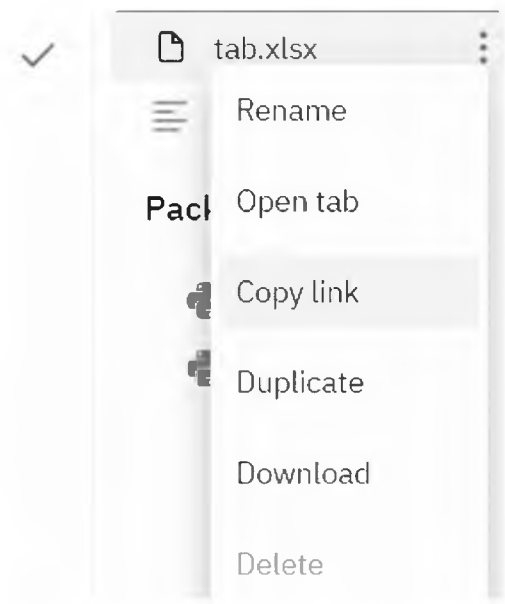
Результат

Вывод на экран

```
xk=1
h=0.2
a=1.5
                x      f(x)
0  -1.000000e+00 -2.358606
1  -8.000000e-01 -2.998222
2  -6.000000e-01 -3.408296
3  -4.000000e-01 -3.645575
4  -2.000000e-01 -3.736532
5  -5.551115e-17 -3.695723
6   2.000000e-01 -3.532235
7   4.000000e-01 -3.252306
8   6.000000e-01 -2.860532
9   8.000000e-01 -2.360466
10  1.000000e+00 -1.754957
```

Файл *tab.xlsx*

Файл *tab.xlsx* предварительно с помощью команды *Download*



Необходимо загрузить из текущего каталога в папку Загрузки и только после копировать в другую папку и/или открывать для дальнейшей работы.

Результат представлен в таблице *Excel.*

	A	B	C	D	E	F	G
1	x	f(x)					
2	-1	-0,84147					
3	-0,8	-0,71736					
4	-0,6	-0,56464					
5	-0,4	-0,38942					
6	-0,2	-0,19867					
7	-5,6E-17	-5,6E-17					
8	0,2	0,198669					
9	0,4	0,389418					
10	0,6	0,564642					
11	0,8	0,717356					
12	1	0,841471					
13							
14							
15							
16							

Табулирование

Готово 100%

Задача 13

Дана таблица, содержащая медицинские данные об учащихся.

№	Ф. И. О.	Рост, см	Вес, кг	Идеальный вес, кг	Группа здоровья
1	Петров А. И.	170	55		
2	Иванов Р. С.	168	67		
3	Кузнецов В. Ю.	174	63		
4	Сидоров Л. И.	172	57		
5	Коромыслов М. В.	181	90		
6	Чернов П. А.	173	68		
7	Кудрявцев Н. О.	174	59		
8	Покатин Р. В.	178	64		
9	Бакурев И. И.	180	71		
10	Семенов Л. П	176	91		

Заполните столбцы «Идеальный вес» и «Группа здоровья».

Идеальный вес = Рост – 110.

Данные о группе здоровья представьте римскими цифрами:

I группа — (Вес – Идеальный вес) / Вес < –0.10;

II группа — (Вес – Идеальный вес) / Вес < 0.1;

III группа — (Вес – Идеальный вес) / Вес > 0.1.

Программа

```
import pandas as pd
tabs = pd.read_excel ('Сведения.xlsx')
print (tabs)
k=len(tabs.index)
print("-----------------------------------------------------------------")
for i in range (1,k):
  tabs.loc[i,'Идеальный вес,кг']=tabs.loc[i,'Рост,см']-110
  u=(tabs.loc[i,'Вес,кг']-tabs.loc[i,'Идеальный вес,кг'])/ tabs.loc[i,'Вес,кг']
```

```
    if u<–0.1:
      tabs.loc[i,'Группа здоровья']="I"
    else:
      if u<0.1:
        tabs.loc[i,'Группа здоровья']="II"
      else: tabs.loc[i,'Группа здоровья']="III"
print (tabs)
```

Результат

Сначала выводится исходная таблица с незаполненными столбцами «Идеальный вес», кг, и «Группа здоровья».

```
      №           Ф.И.О.  Рост,см  Вес,кг  Идеальный вес,кг  Группа здоровья
0   NaN              NaN      NaN     NaN               NaN              NaN
1   1.0     Петров А.И.     170.0    55.0               NaN              NaN
2   2.0     Иванов Р.С.     168.0    67.0               NaN              NaN
3   3.0   Кузнецов В.Ю.     174.0    63.0               NaN              NaN
4   4.0     Сидоров Л.И.    172.0    57.0               NaN              NaN
5   5.0  Коромыслов М.В.    181.0    90.0               NaN              NaN
6   6.0     Чернов П.А.     173.0    68.0               NaN              NaN
7   7.0   Кудрявцев Н.О.    174.0    59.0               NaN              NaN
8   8.0     Покатин Р.В.    178.0    64.0               NaN              NaN
9   9.0     Бакурев И.И.    180.0    71.0               NaN              NaN
10  10.0     Семенов Л.П    176.0    91.0               NaN              NaN
------------------------------------------------------------------------------
      №           Ф.И.О.  Рост,см  Вес,кг  Идеальный вес,кг Группа здоровья
0   NaN              NaN      NaN     NaN               NaN              NaN
1   1.0     Петров А.И.     170.0    55.0              60.0               II
2   2.0     Иванов Р.С.     168.0    67.0              58.0              III
3   3.0   Кузнецов В.Ю.     174.0    63.0              64.0               II
4   4.0     Сидоров Л.И.    172.0    57.0              62.0               II
5   5.0  Коромыслов М.В.    181.0    90.0              71.0              III
6   6.0     Чернов П.А.     173.0    68.0              63.0               II
7   7.0   Кудрявцев Н.О.    174.0    59.0              64.0               II
8   8.0     Покатин Р.В.    178.0    64.0              68.0               II
9   9.0     Бакурев И.И.    180.0    71.0              70.0               II
10  10.0     Семенов Л.П    176.0    91.0              66.0              III
>
```

Примеры решения задач вычислительной математики

Одним из методов научного познания является моделирование. Моделирование — процесс изучения объекта путем построения и исследования его модели, осуществляемый с определенной целью: заменить эксперимент с оригиналом экспериментом на модели.

Появление и непрерывное совершенствование быстродействующих вычислительных средств открыло невиданные ранее возможности для применения математических методов в науке и других сферах деятельности.

Рассмотрим некоторые методы вычислительной математики и их реализацию на Python для задач математического моделирования.

Приближенные методы решения уравнения $f(x) = 0$

Постановка задачи

Дано уравнение вида $f(x) = 0$. Найти один из корней этого уравнения с точностью $\varepsilon > 0$.

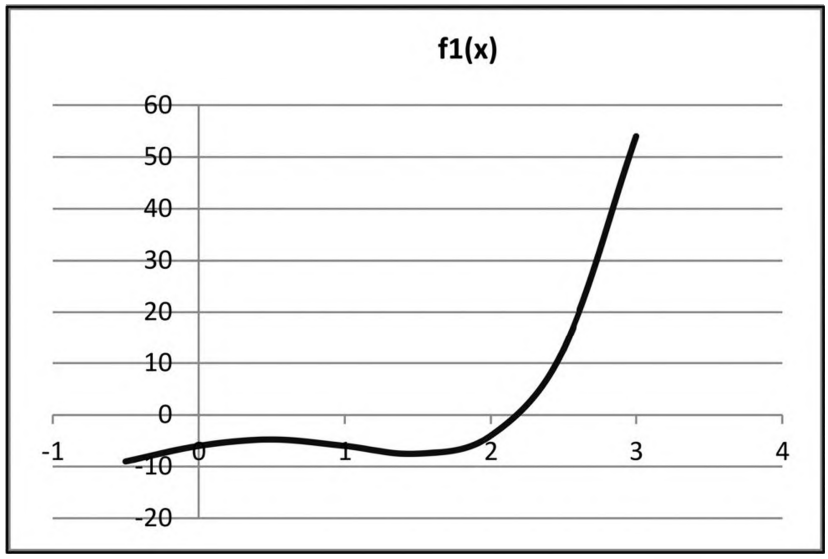
Этапы решения задачи

Приближенное решение уравнения $f(x) = 0$ включает следующие этапы.

- Отделение корней и выбор отрезка, на котором локализован искомый корень. Производится исследование функции $f(x)$ для нахождения отрезков, содержащих по одному корню.
- Уточнение корней. По выбранному алгоритму последовательно сужается отрезок, содержащий корень, до такой степени, пока не станет выполнять условие точности $\varepsilon > 0$.

Локализацию корней удобно выполнять в электронных таблицах.

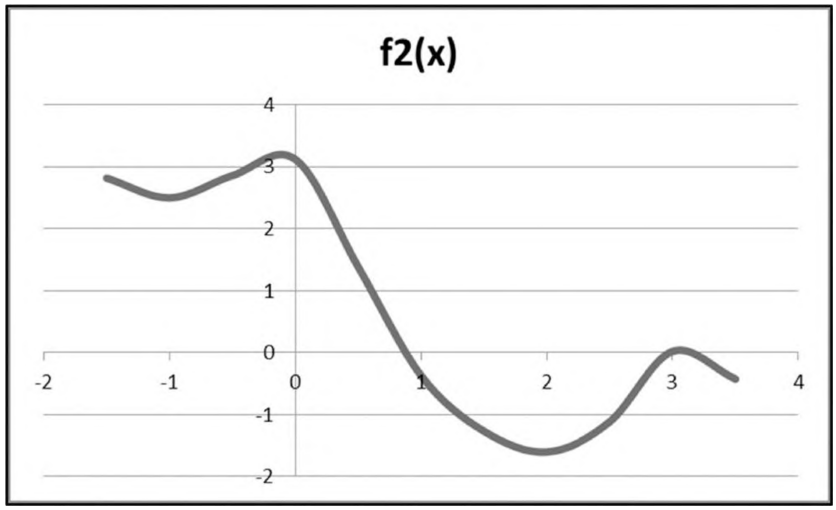
Строим график функции $f1(x)=2*x*x*x*x–3*x*x*x–4*x*x+5*x–6$.



Выбираем отрезок локализации корня [2;3].

Строим таблицу значений и график функции

$$f2(x)= \exp(\cos(2*x))–3*\sin(0.5*x)+0.4.$$



Выбираем отрезок с первым четко видимым на рисунке корнем [0;1].

Рассмотрим методы уточнения корней.

Метод половинного деления

Алгоритм метода

1. Ввод исходных данных: a, b, ε.
2. Расчет середины текущего отрезка $c=(a+b)/2$.

3. Проверка условия $f(a)*f(c)<0$. Если условие выполняется, то $b=c$, иначе $a=c$.

4. Проверка *условия* $(b–a)<=$ ε. Если условие выполняется, то переход к пункту 5, иначе переход к пункту 2.

5. Расчет закончен. Корень $x*=(a+b)/2$.

Функция метода

```
def del1 (a,b,e,f):
 n=0
 if f(a)*f(b) <= 0:
  while abs(b – a)>e:
   n+=1
   if n > 100:
    print(" Корень с заданной точностью не найден! ")
    break
   else:
    c = (a + b)*0.5
   if f(a) * f(c) < 0: b = c
   else: a = c
  return (a + b)*0.5
 else:
  print ("Метод половинного деления: на заданном отрезке корней нет")
  return "*"
```

Метод касательных (метод Ньютона)

Пусть корень уравнения $f(x) = 0$ локализован на отрезке $[a;b]$. Функция $f(x)$ на отрезке $[a;b]$ должна быть дважды дифференцируема. Требуется найти значение корня с точностью ε.

Для метода касательных в качестве начального приближения достаточно одной точки $x0$. Обычно выбирают тот конец отрезка $[a, b]$, для которого выполняется условие $f(x_0)\cdot f''(x_0)>0$.

Алгоритм метода

1. Ввод исходных данных: a, b, ε.

2. Расчет значений функции $f(a)$, $f(b)$ на концах отрезка и значения второй производной от функции $f''(a)$ и $f''(b)$.

3. Проверяем условие $f(a)\cdot f''(a) > 0$. Если условие выполняется, то $c = a$, $x = b$, иначе проверяем условие $f(b)\cdot f''(b) > 0$ $c = b$, $x = a$. Если условие «быстрой» сходимости не выполняется, то принимаем $c = a$, $x = b$.

4. $x_0 = x$.

5. Рассчитываем абсциссу точки пересечения хорды с осью OX по формуле

$$x = x_0 - \frac{f(x_0)}{f'(x_0)}$$

6. Проверка условия $|x–x_0|<=\varepsilon$. Если условие выполняется, то переход к пункту 7, иначе переход к пункту 4.

7. Расчет закончен. Корень $x^*=x$.

Функция метода

```
def newton (a,b,e,f,f1,f2):
 if f(a)*f2(a) > 0: x0 = a
 else: x0 = b
 x = x0 – f(x0) / f1(x0)
 n=0
 while abs(x – x0) > e:
  n+=1
  if n > 100:
   print(" Корень с заданной точностью не найден! ")
   break
  else:
   x0 = x
   x = x0 – f(x0) / f1(x0)
 return x
```

Задание

Составить программу для нахождения корня функции $f1(x) = 0$ и функции $f2(x) = 0$ двумя методами: половинного деления и касательных, с точностью $\varepsilon = 0{,}00001$.

Программа

Файл *func.py*

```
import math
def f1 (x): # первая функция
  return 2 * x*x*x*x – 3 * x*x*x – 4 * x*x + 5 * x – 6
def f11(x): # первая производная первой функции
  return 8*x*x*x – 9*x*x – 8 * x + 5
def f12(x): # вторая производная первой функции
  return 24*x*x – 18*x*x – 8
def f2 (x): # вторая функция
  return math.exp(math.cos(2*x))–3*math.sin(0.5*x)+0.4
def f21(x): # первая производная второй функции
return  –2*math.exp(math.cos(2*x))*math.sin(2*x)–1.5*math.cos(0.5*x)
def f22(x): # вторая производная второй функции
  return
4*math.exp(math.cos(2*x))*(math.cos(2*x)+math.sin(2*x))+0.75*math.sin
(0.5*x)
```

Файл *prog.py*

```
def newton (a,b,e,f,f1,f2): # метод касательных
  if f(a)*f2(a) > 0: x0 = a
  else: x0 = b
  x = x0 – f(x0) / f1(x0)
  n=0
  while abs(x – x0) > e:
    n+=1
    if n > 100:
      print(" Корень с заданной точностью не найден! ")
      break
```

```
    else:
      x0 = x
      x = x0 – f(x0) / f1(x0)
  return x
def del1 (a,b,e,f): # метод половинного деления
  n=0
  if f(a)*f(b) <= 0:
    while abs(b – a)>e:
      n+=1
      if n > 100:
        print(" Корень с заданной точностью не найден! ")
        break
      else:
        c = (a + b)*0.5
      if f(a) * f(c) < 0: b = c
      else: a = c
    return (a + b)*0.5
  else:
    print ("Метод половинного деления: на заданном отрезке корней нет")
    return "*"
```

Файл *main.py*

```
import prog
import func
a=float(input("a="))
b=float(input("b="))
e=1e–5
x=prog.del1 (a,b,e, func.f1)
if x!="*":
  print ("Метод половинного деления f1=",func.f1(x)," x=",x)
x=prog.newton (a,b,e, func.f1,func.f11,func.f12)
print ("Метод Ньютона f1=",func.f1(x)," x=",x)
```

```
a=float(input("a="))
b=float(input("b="))
x=prog.del1 (a,b,e, func.f2)
if x!="*":
  print ("Метод половинного деления f2=",func.f2(x)," x=",x)
x=prog.newton (a,b,e, func.f2,func.f21,func.f22)
print ("Метод Ньютона f2=",func.f2(x)," x=",x)
```

Результат

```
a=2
b=3
Метод половинного деления f1= 7.9517221632130 49e-05  x= 2.180713653564453
Метод Ньютона f1= 5.329070518200751e-15  x= 2.1807107847570397
a=0
b=1
Метод половинного деления f2= -9.753223630282193e-06  x= 0.8636817932128906
Метод Ньютона f2= 3.3306690738754696e-16  x= 0.8636785981703813
>
```

Для $f1$ корень равен 2,18071, для $f2$ корень равен 0,86368 с учетом заданной точности $e=1e-5$.

Сравниваем возможности используемых методов. Метод половинного деления ищет корень только на заданном отрезке, поэтому на отрезке [7;8] корней нет. Метод Ньютона не ограничен отрезком, в качестве отправной требуется только одна начальная точка, поэтому и для $f1$ находит корень 2,18071, и для $f2$ находит корень 17,98588.

```
a=7
b=8
Метод половинного деления: на заданном отрезке корней нет
Метод Ньютона f1= 8.430625086930377e-10  x= 2.1807107847874554
a=7
b=8
Метод половинного деления: на заданном отрезке корней нет
Метод Ньютона f2= -7.771561172376096e-16  x= 17.985877323368378
>
```

Решение систем линейных уравнений (СЛАУ)

Метод простой итерации (метод Якоби)

Дана система *n* линейных уравнений:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nn}x_n = b_n \end{cases}$$

и начальное приближение $x^0 = \{x_1^0,\ x_2^0, .., x_n^0\}$.

Найти решение этой системы с точностью ε. Выполняются условия существования единственного решения СЛАУ.

Достаточное условие сходимости: если выполнено условие диагонального преобладания

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{n} |a_{ij}|,$$

то итерационный процесс сходится при любом выборе начального приближения. Выбор начального приближения влияет на количество итераций, необходимых для получения приближенного решения. Наиболее часто в качестве начального приближения берут $x^0 = \left\{\frac{b_1}{a_{11}}, \frac{b_2}{a_{22}}, \ldots, \frac{b_n}{a_{nn}}\right\}$ или $x^0 = \{0,0, \ldots, 0\}$.

Алгоритм метода

1. Ввод исходных данных: A, b, ε.
2. Задание начального приближения x^1.
3. Присваиваем $x^0 = x^1$.
4. Расчет x^1. Расчетная формула:

$$x_i^1 = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j^0 - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} \cdot x_j^0, \quad i = 1,2, \ldots, n.$$

5. Вычисляем наибольшую из разностей $|x_j^1 - x_j^0|$.

6. Проверяем условие $\max|x_j^1 - x_j^0| \leq \varepsilon$. Если оно выполняется, то переходим к пункту 7, если нет — переходим к новой итерации, к пункту 3.

7. Расчет закончен. Результат — значения x^1.

Задание

Составить программу решения СЛАУ методом простой итерации. Исходные данные вводятся с клавиатуры.

Программа

Файл *clay*1*.py*

```
def clayP(n,a,b,eps,x ):
 x=[]; x0=[]; e=[]
 for i in range(0,n):
  x0.append(0)
  x.append(b[i]/a[i][i])
  e.append(abs(x[i]-x0[i]))
 k=0
 while max(e)>eps:
  k+=1
  for i in range(0,n):
   x0[i]=x[i]
  for i in range(0,n):
   s=b[i]/a[i][i]
   for j in range(0,i):
    s-=a[i][j]*x0[j]/a[i][i]
   for j in range(i+1,n):
    s-=a[i][j]*x0[j]/a[i][i]
   x[i]=s
  for i in range(0,n):
   e[i]=abs(x[i]–x0[i])
```

```
  x.append(k)
  return x

def vvod (n, a,b,eps):
  n=int(input("n="))
  eps=float(input("eps="))
  a = [ [0]*n for i in range(n) ]
  b=[0 for i in range(n)]
  for i in range(0,n):
    for j in range (0,n):
      print("a[{0:2},{1:2}]=".format(i,j),end=' ')
      a[i][j]=float(input())
    print("b[{0:2}]=".format(i),end=' ')
    b[i]=float(input())
  return n,eps,a,b
def main1():
  a=[[]]
  b=[]
  n=0; e=0
  n,e,a,b=vvod (n, a, b,e)
  x=[]
  x=clayP(n,a,b,e,x)
  print  ("---Результат-метод простой итерации---")
  print ("Количество итераций=",x[n])
  for i in range(0,n):
    print(x[i],end=" ")
  print(" ")
main1()
```

Файл *main.py*

```
import clay1
```

Результат

```
a[ 0, 0]= 5
a[ 0, 1]= 1
a[ 0, 2]= 1
b[ 0]= 7
a[ 1, 0]= 1
a[ 1, 1]= 5
a[ 1, 2]= 1
b[ 1]= 7
a[ 2, 0]= 1
a[ 2, 1]= 1
a[ 2, 2]= 5
b[ 2]= 7
---Результаты-метод простой итерации---
Количество итераций= 13
0.99999731564544 0.99999731564544 0.99999731564544
```

Метод Зейделя

Алгоритм метода

1. Ввод исходных данных: A, b, ε.
2. Задание начального приближения x^1.
3. Присваиваем $x^0 = x^1$.
4. Расчет x^1. Расчетная формула:

$$x_i^1 = \frac{b_i}{a_{ii}} - \sum_{j=1}^{i-1} \frac{a_{ij}}{a_{ii}} \cdot x_j^1 - \sum_{j=i+1}^{n} \frac{a_{ij}}{a_{ii}} \cdot x_j^0, \quad i = 1,2,\dots,n.$$

5. Вычисляем наибольшую из разностей $|x_j^1 - x_j^0|$.
6. Проверяем условие $\max|x_j^1 - x_j^0| \leq$ ε. Если оно выполняется, то переходим к пункту 7, если нет — переходим к новой итерации, к пункту 3.
7. Расчет закончен. Результат — значения x^1.

Задание

Составить программу решения СЛАУ методом Зейделя. Исходные данные читаются из файла, имя которого вводится в переменную *st*.

Программа

Файл *clayz.py*

```
def ClayZ(n,a,e,x):
 k=0
 x0=[]
 z=float(1)
 while z>e: # or k<20:
  k+=1
  x0=x[:]
  for i in range(0,n):
    r=float(a[i][n]/a[i][i])
    for j in range(0,i):
      r-=a[i][j]/a[i][i]*x0[j]
    for j in range(i+1,n):
      r-=a[i][j]/a[i][i]*x[j]
    x[i]=r
    z=abs(x[i]-x0[i]);
 return(k)

st=input()
f1 = open(st, 'r')
a=[]
e=1.0E-5
n=int(f1.readline())
for i in range(0,n):
  c=f1.readline()
  s1=c.rstrip()
  s=[]
  s1= s1.split(' ')
  for j in range(0,len(s1)):
    s.append(float(s1[j]))
  a.append(s)
```

```
x=[]
for i in range(0,n):
  x.append(float(0))
print (a)
print("__Результат метод Зейделя__")
print ("Число итераций=",ClayZ(n,a,e,x))
s=""
for i in range(0,n):
  d=str("{0:.4f} ".format(x[i]))
  s+=d
print (s)
```

Файл *main.py*

```
import clayz
```

Файл с исходными данными *st.txt*

```
3
5 1 1 7
1 5 1 7
1 1 5 7
```

Результат

```
st.txt
[[5.0, 1.0, 1.0, 7.0], [1.0, 5.0, 1.0, 7.0], [1.0, 1.0, 5.0, 7.0]]
__Результаты метод Зейделя__
Число итераций= 14
1.0000 1.0000 1.0000
```

Интерполяция по Лагранжу

Пусть некоторая функция $f(x)$ довольно сложная для исследования задана на отрезке $[x_0; x_n]$ в виде таблицы, называемой сеточной функцией:

x_i	x_0	x_1	...	x_n
y_i	y_0	y_1	...	y_n

где $x_i = x_0, x_1, ..., x_n$ — узлы интерполяции, определенные на отрезке $[x_0; x_n]$, и y_i равные значениям $f(x_i)$ в узлах интерполяции: $y_0 = f(x_0)$, $y_1 = f(x_1)$, ..., $y_n = f(x_n)$. В простейшем случае узлы интерполяции образуют равномерную сетку, то есть расстояние между соседними узлами одинаково, однако, сетка может быть и неравномерной. Необходимое и достаточное требование к сетке: $x_0 < x_1 < x_2 < ... < x_n$. Функция $f(x)$ может быть заранее не известна, а узлы интерполяции x_i и значения y_i являются результатом экспериментальных исследований.

Наиболее востребованными являются задачи двух типов:

- построить функцию $F(x)$ — интерполяционную функцию, принадлежащую известному классу, например, к многочленам, и принимающую в узлах интерполяции те же значения, что и $f(x)$: $F(x_0) = y_0$, $F(x_1) = y_1$, ..., $F(x_n) = y_n$;
- уплотнение таблицы — определение приближенного значения функции $F(x)$ для заданного a, удовлетворяющего условию: $x_0 < a < x_n$.

Задание

Составить программу для решения задачи уплотнения таблицы методом Лагранжа.

Алгоритм метода

Приближенное значение находим по формуле:

$$L_n(a) = \sum_{i=0}^{n} y_i \cdot \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(a - x_j)}{(x_i - x_j)}.$$

Программа

Файл *interL.py*

```
def intL(n,x,y,a): # Реализует метод Лагранжа
  s=0
  for i in range (0,n):
```

```
    p=1
    for j in range (0,n):
      if i!=j: p*=(a–x[j])/(x[i]–x[j])
    s+=y[i]*p
  return s

def vvod (st): # Реализует чтение из файла исходных данных
  f =open(st, 'r')
  n=int(f.readline())
  a=float(f.readline())
  x=[]; y=[]
  for i in range(0,n):
    x.append(0)
    y.append(0)
  for i in range(0,n):
    c=f.readline()
    s=c.rstrip()
    s= s.split(' ')
    x[i]=float(s[0])
    y[i]=float(s[1])
  return (n,a,x,y) # возвращает число

st="inter.txt"#st=input()
n,a,x,y=vvod(st)
ya=intL(n,x,y,a)
print ("y({0: .2f})={1: 0.4f} ".format(a,ya))
```

Файл *main.py*

```
import interL
```

Файл с исходными данными *inter.txt*

```
5
15
0 0
10 0.1736
```

20 0.3420
40 0.6428
50 0.7660

В файле первое число $n = 5$, второе $a = 15$. Далее приведена таблица, в каждой строке которой задается $x[i]$ и $y[i]$.

Результат

Вычисление определенных интегралов

Вычислить определенный интеграл $\int_a^b f(x) \cdot dx$, где $f(x)$ — непрерывная на отрезке $[a; b]$ функция.

Метод трапеций

Алгоритм метода

1. Вводим исходные значения ε, a, b.
2. Задаем n=1, S=0.
3. Приравниваем S_0=S, n=2·n.
4. Расчет h=(b–a)/n.
5. Расчет приближенного значения интеграла:

$$S = \sum_{i=1}^{n} S_i = \sum_{i=1}^{n} h \cdot \frac{f(x_{i-1}) + f(x_i)}{2} = h \cdot \left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(x_i) \right)$$

6. Проверка условия $|S_0–S|<=\varepsilon$. Если условие выполняется, то переходим к пункту 7, нет — переходим к пункту 3.
7. Расчет закончен. Результат равен S.

Функция, реализующая метод трапеций:

```
def Trap(n,a,b,f):
  s = (f(a) + f(b))*0.5
  h = (b – a) / n
```

```
for i in range(1,n):
  s += f(a + h*(i–1))
return s*h
```

Метод Симпсона (метод парабол)

Алгоритм метода

1. Вводим исходные значения ε, a, b.
2. Задаем n=1, S=0.
3. Приравниваем S_0=S, n=2·n.
4. Расчет h=(b–a)/n.
5. Расчет приближенного значения интеграла:

$$S = \frac{h}{3} \cdot (f_0 + f_n + 4 \cdot \sum_{i=1,3,5,\ldots}^{n-1} f_i + 2 \cdot \sum_{i=2,4,6,\ldots}^{n-2} f_i)$$

6. Проверка условия |S0–S|<=ε. Если условие выполняется, то переходим к пункту 7, нет — переходим к пункту 3.
7. Расчет закончен. Результат равен S.

Функция, реализующая метод Симпсона:

```
def Simp(n,a,b,f):
 s)= f(a) + f(b)
 h = (b - a) / n
 for i in range (1,n,2):
   s += 4*f(a + h*i)
 for i in range (2,n,2):
          s += 2 * f(a + h*i)
 return s*h/3
```

Задание

Составить программу для приближенного вычисления числа π, используя формулу: $\int_0^1 f1(x) \cdot dx$, где $f1(x)$=4.0/(1+x*x) и функции

Лапласа $\int_0^x f2(z) \cdot dz$, где $f2(z) = \frac{1}{\sqrt{2\pi}} e^{-(z^2/2)}$ двумя методами: трапеций и Симпсона с точностью ε.

Программа

Файл *Integral*()

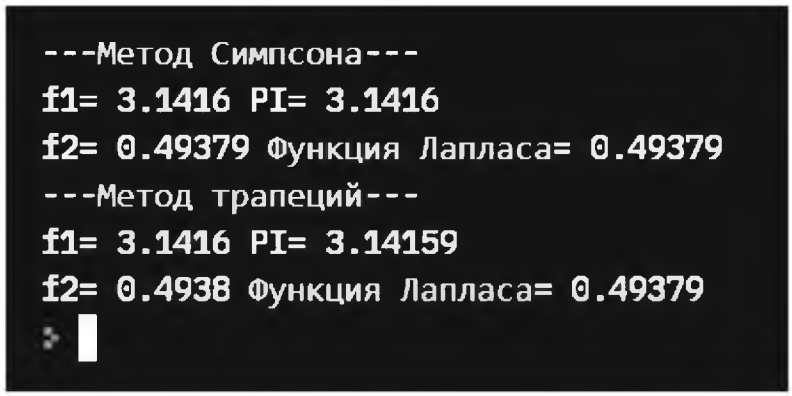
```
def f1(x):
  return 4.0/(1+x*x) # подынтегральная функция1
import math
def f2 (x):
  return math.sqrt(1/(2*math.pi))*math.exp(–x*x*0.5) # функция2

def Met(n,a1,b1,M,f):
 # M — имя метода
# f — имя подынтегральной функции
  return M (n,a1, b1, f)
# Метод трапеций
def Trap(n,a,b,f):
  s = (f(a) + f(b))*0.5
  h = (b – a) / n
  for i in range(1,n):
    s += f(a + h*(i–1))
  return s*h
# Метод Симпсона
def Simp(n,a,b,f):
  s = f(a) + f(b)
  h = (b – a) / n
  for i in range (1,n,2):
    s += 4*f(a + h*i)
  for i in range (2,n,2):
            s += 2 * f(a + h*i)
  return s*h/3
```

```
def inteG (a,b,eps,Met,f):
  n = 2
  In0 = Met(n, a, b, f)
  n=4
  In1 = Met(n, a, b, f)
  while abs(In1 – In0) > eps:
    n *= 2
    In0 = In1
    In1 = Met(n, a, b, f)
  return In1
```

Файл *main.py*

```
import math
import Integral
print ("---Метод Симпсона---")
a = 0; b = 1; eps = 1e-5
rez = Integral.inteG(a,b,eps,Integral.Simp,Integral.f1)
print ("f1={0: 0.5} PI={1: 0.5}".format(rez,math.pi ))
a = 0; b = 2.5; eps = 1e-5
rez = Integral.inteG(a,b,eps,Integral.Simp,Integral.f2)
v=math.erf(b/2**0.5)/2
print("f2={0: 0.5} Функция Лапласа={1: 0.5}".format(rez,v))
print ("---Метод трапеций---")
a = 0; b = 1; eps = 1e-5
rez = Integral.inteG(a,b,eps,Integral.Trap,Integral.f1)
print ("f1={0: 0.6} PI={1: 0.6}".format(rez,math.pi ))
a = 0; b = 2.5; eps = 1e-5
rez = Integral.inteG(a,b,eps,Integral.Trap,Integral.f2)
print("f2={0: 0.5} Функция Лапласа={1: 0.6}".format(rez,
math.erf(b/2**0.5)/2))
```

Результат



Решение обыкновенных дифференциальных уравнений (ОДУ)

Постановка задачи

Дано: ОДУ вида $y'=F(x,y)$ и начальные условия x_0, $y(x_0)=y_0$, а также значения x_n и ε. Найти: $y(x)$.

Метод Рунге — Кутта

Классический метод Рунге — Кутта четвертого порядка для решения ОДУ первого порядка в постановке Коши описывается следующей системой пяти равенств:

$$y_{i+1}=y_i+\frac{h}{6}(k_1+2k_2+2k_3+k_4),$$

где

$$k_1=F(x_i,y_i),$$

$$k_2=F(x_i+\frac{h}{2},y_i+\frac{k_1 h}{2}),$$

$$k_3=F(x_i+\frac{h}{2},y_i+\frac{k_2 h}{2}),$$

$$k_4=F(x_i+h,\ y_i+k_3 h).$$

Алгоритм метода

1. Вводим исходные значения x_0, y_0, x_n, ε.
2. Приравниваем $n = 5$, $yy_0 = y_0$, $yy_1 = y_0+(x_1 - x_0)\cdot f(x_0,y_0)$.
3. Приравниваем $yy_0 = yy_1$, $n = 2\cdot n$, $h = (x_1 - x_0) / n$.

4. Расчет приближенного значения $yy_1 = y(x_n)$.

$$k_1 = F(x_i,y_i),\ k_2 = F(x_i+\frac{h}{2},\ y_i+\frac{k_1 h}{2}),\ k_3 = F(x_i+\frac{h}{2},\ y_i+\frac{k_2 h}{2}),\ k_4 = F(x_i+h,\ y_i+k_3 h),$$

$$y_{i+1} = y_i+\frac{h}{6}(k_1+2k_2+2k_3+k_4).$$

5. Проверка условия $|yy_1–yy_0|<=\varepsilon$. Если условие выполняется, то переходим к пункту 6, нет — переходим к пункту 3.

6. Расчет закончен. Результат равен yy_1.

Задание

Составить программу для приближенного решения методом Рунге — Кутта двух дифференциальных уравнений первого порядка:

$$y' = f1(x, y),$$

$$f1(x,y) = e^{cos^2(x)} + 1{,}5 \cdot sin(y) - \frac{2{,}3}{x^2 + 0{,}2}$$

при начальных условиях $x_0 = 0$, $y(x_0) = 0{,}1$, для $x_n = 2$ с точностью $\varepsilon = 0{,}001$.

$$y' = f2(x, y), f2(x, y) = x + y/2$$

при начальных условиях $x_0 = 0$, $y(x_0) = –0{,}3$, для $x_n = 2{,}5$ с точностью $\varepsilon = 0{,}0001$.

Программа

Файл *diff.py*

```
import math
def f1 (x,y):
  return math.exp(math.cos(x)**2)+1.5*math.sin(y)–2.3/(x*x+0.2)
def f2 (x,y):
  return x+0.5*y
def PK4 (x0,y0,x1,eps,f):
  n=5
  yy0=y0
  yy1=y0+(x1–x0)*f(x0,y0)
  while abs(yy1–yy0)>eps:
```

```
    yy0=yy1
    n*=2
    h = (x1 – x0) / n
    rx=[]
    ry=[]
    for i in  range (0, n+1):
      rx.append(0)
      ry.append(0)
    x=x0
    rx[0]=x0
    ry[0] = y0
    for i in  range (0, n+1):
      k1 = f(rx[i – 1], ry[i – 1])
      k2 = f(rx[i – 1] + h*0.5, ry[i – 1] + k1*h*0.5)
      k3 = f(rx[i – 1] + h*0.5, ry[i – 1] + k2*h*0.5)
      k4 = f(rx[i – 1] + h, ry[i - 1] + k3*h)
      ry[i] = ry[i – 1] + h/6.0*(k1+2*k2+2*k3+k4)
      rx[i]=x
      x += h
      yy1=ry[i]
      return yy1
def Metod(a1,b1,ak,eps,M,f):
  return M (a1, b1, ak, eps,f)

x0 = 0; x1 =2.0; y0 =0.1
y=Metod(x0,y0,x1,0.001,PK4,f1)
print ("y({0: 0.2})= {1: 0.4}".format(x1,y))
x0 = 0; x1 = 2.5; y0 = –0.3
y=Metod(x0,y0,x1,0.0001,PK4,f2)
print ("y({0: 0.2})= {1: 0.4}".format(x1,y))
```

Файл main.py

```
import math
import diff
```

Результат

```
y( 2.0)= -3.184
y( 2.5)=  4.961
```

Нахождение минимума функции f(x)

Постановка задачи: дана функция $f(x)$. Найти локальный минимум этой функции с заданной точностью ε.

Этапы решения задачи:

- локализация минимума — выбор отрезка, на котором находится один минимум. Производится исследование функции $f(x)$ для нахождения отрезков, содержащих по одному минимуму;
- уточнение минимума — по выбранному алгоритму последовательно сужается отрезок, содержащий минимум, до такой степени, пока не станет выполнять условие точности ε>0.

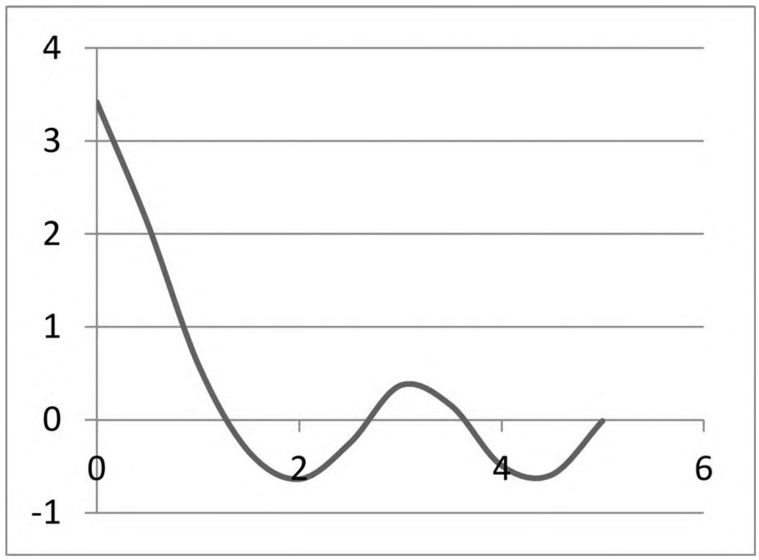
Метод двойного половинного деления

Локализация минимума

Строим таблицу значений и график функции:

$$f = \exp(\cos(x)\text{^}2)-3*\sin(x/2)+0.7.$$

График:



Выбираем отрезок с первым из видимых на рисунке минимумов [1; 3].

Алгоритм метода

1. Ввод исходных данных: a, b, ε.
2. Определение положения точек x_1, x_2: $x_1 = a+0{,}25\cdot(b-a)$; $x_2 = a+0{,}75\cdot(b-a)$.
3. Проверка условия $f(x_1) < f(x_2)$. Если условие выполняется, то $b = x_2$, иначе $a = x_1$.
4. Проверка условия $(b-a) <= \varepsilon$. Если условие выполняется, то переходим к пункту 5, нет — переходим к пункту 2.
5. Расчет закончен. Минимум находится в точке $x^* = (a+b)/2$, минимальное значение функции на отрезке $[a; b]$ равно $f(x^*)$.

Программа

Файл *optim.py*

```
import math
def f1(x):
  return math.exp((math.cos(x))**2)–3*math.sin(x*0.5)+0.7
def Met(a, b, eps, M, f):
  return M (a, b, eps, f)
def delP (a, b, eps, f):
  n = 0
  while abs(b–a)>eps:
    n+ = 1
    x1 = a + 0.25 * (b – a)
    x2 = a + 0.75 * (b – a)
    if f(x1) < f(x2): b = x2
    else: a = x1
  #print("n=",n)
  return (a+b)*0.5
a = 1
b = 3
```

eps = 1*e*–8
x = *Met*(*a*, *b*, *eps*, *delP*, *f*1)
print("x=",*x*," *f*(*x*)=", *f*1(*x*))

Файл *main.py*

```
import math
import optim
```

Результат

```
x= 1.9686782139628888   f(x)= -0.6367338957026429
>
```

Парная регрессия

Линейная модель парной регрессии

Линейная регрессия сводится к нахождению уравнения вида $y_p = a \cdot x + b$ или $y = a \cdot x + b + \varepsilon$.

Уравнение вида $y_p = a \cdot x + b$ по заданным значениям фактора x позволяет рассчитать теоретические значения результирующего показателя, подставляя в него фактические значения фактора x.

Построение линейной регрессии сводится к оценке ее параметров a и b. Классический подход к оцениванию параметров линейной регрессии основан на методе наименьших квадратов (МНК). МНК позволяет получить такие оценки параметров a и b, при которых сумма квадратов отклонений фактических значений результирующего показателя y от теоретических y_p минимальна:

$$\emptyset = \sum_{i=1}^{n} (y_i - y_{pi})^2 = \sum_{i=1}^{n} \varepsilon^2 \to min$$

где n — число экспериментальных точек.

Для определения минимума функции вычисляем частные производные по каждому из параметров a и b и приравниваем их к нулю.

$$\begin{cases} \dfrac{\partial \emptyset}{\partial a} = -2\sum_{i=1}^{n}(y_i - a \cdot x_i - b) \cdot x_i = 0 \\ \dfrac{\partial \emptyset}{\partial b} = -2\sum_{i=1}^{n}(y_i - a \cdot x_i - b) = 0 \end{cases}$$

Производим простейшие преобразования и получаем систему линейных уравнений:

$$\begin{cases} a \cdot \sum_{i=1}^{n} x^2 + b \cdot \sum_{i=1}^{n} x = \sum_{i=1}^{n}(x_i \cdot y_i) \\ a \cdot \sum_{i=1}^{n} x + b \cdot n = \sum_{i=1}^{n} y_i \end{cases}$$

Решаем систему линейных уравнений и получаем:

$$a = \frac{\sum_{i=1}^{n}(y_i \cdot x_i) \cdot n - \sum_{i=1}^{n} y_i \cdot \sum_{i=1}^{n} x_i}{\sum_{i=1}^{n} x_i^2 \cdot n - \left(\sum_{i=1}^{n} x_i\right)^2}$$

$$b = \frac{\sum_{i=1}^{n}(y_i \cdot x_i) - a \cdot \sum_{i=1}^{n} x_i^2}{\sum_{i=1}^{n} x_i}$$

Введем обозначение (остаточная дисперсия):

$$s = \sum_{i=1}^{n} \varepsilon^2$$

Задание

Постановка задачи[1]

Д. И. Менделеев в труде «Основы химии» приводит данные растворимости y натриевой селитры $NaNO_3$ на 100 г воды в зависимости от температуры $t°$:

[1] Метод наименьших квадратов: методические указания и индивидуальные задания по выполнению лабораторной работы № 15 / Юго-Зап. гос. ун-т ; сост. Л. И. Студеникина, Т. В. Шевцова. — Курск, 2011. — 52 с.

t_i^0	0	4	10	15	21	29	35	51	68
y_i	66,7	71,0	76,3	80,6	85,7	92,9	99,4	113,6	125,1

n = 9. Составить программу для расчета коэффициентов по a и b линейной парной зависимости по МНК.

Файл *MNK2.py*

```
import math
def mnkl(n, x, y):
 z1=0; z2= 0; b1= 0; b2 = 0
 print("Линейная модель yp = a*x+b")
 print(" x     y      yp      |y–yp|")
 print("------------------------------ ")
 for i in range(0,n):
 #print("x*y=",x[i]*y[i],' y=', y[i])
   z1+=x[i]*x[i]
   z2+=x[i]
   b1+=x[i]*y[i]
   b2+=y[i]
 a=(b1*n–b2*z2)/(z1*n–z2**2)
 b=(b1–z1*a)/z2
 s=0
 for i in range(0,n):
   yp=a*x[i]+b
   d=abs(y[i]–yp)
   s+=d**2
   print("{0: 0.2f}  {1: 0.2f}   {2: 0.2f}  {3: .3f}".format(x[i],y[i], yp, d))
 print("------------------------------ ")
 rez=[0,0,0]
 rez[0]=a
 rez[1]=b
 rez[2]=s
 return rez
```

Файл *main.py*

```
import MNK2
x = [0, 4, 10, 15, 21, 29, 35, 51, 68]
y = [66.7, 71.0, 76.3, 80.6, 85.7, 92.9, 99.4, 113.6, 125.1]
n = 9
a, b, s = MNK2.mnkl(n, x, y)
print ('a = {0: 0.3f} b={1: 0.3f}'.format(a, b))
print("s = {0:0.3f}".format(s))
```

Результат

```
Линейная модель yp=a*x+b
  x      y       yp       |y-yp|
------------------------------
 0.00   66.70    67.56   0.857
 4.00   71.00    71.05   0.047
 10.00   76.30    76.28   0.018
 15.00   80.60    80.64   0.044
 21.00   85.70    85.88   0.179
 29.00   92.90    92.86   0.041
 35.00   99.40    98.09   1.306
 51.00   113.60    112.05   1.547
 68.00   125.10    126.89   1.785
------------------------------
a= 0.872 b= 67.557
s=8.059
>
```

Нелинейные модели парной регрессии

Чаще всего используются нелинейные модели, которые достаточно просто можно привести к линейным моделям относительно параметров a и b, например:

- степенная — $y_p = bx^a$;
- показательная — $y_p = ba^x$;
- экспоненциальная — $y_p = e^{ax+b}$;
- обратная — $y_p = \frac{1}{a \cdot x + b}$.

Линеаризация функции $y_p = bx^a$:

- прологарифмируем функцию $\ln(y_p) = \ln(b)+a\cdot\ln(x)$;
- введем новые переменные $Y = \ln(y_p)$, $B = ln(b)$, $X = \ln(x)$;
- получаем линейное уравнение $Y=a\cdot X+B$;
- рассчитываем a и B, $b = e^B$.

Линеаризация функции $y_p = ba^x$:

- прологарифмируем функцию $\ln(y_p) = \ln(b)+x\cdot\ln(a)$;
- введем новые переменные $Y = \ln(y_p)$, $B = \ln(b)$, $A = \ln(a)$;
- получаем линейное уравнение $Y = A\cdot x+B$;
- рассчитываем A и B, $b = e^B$, $a=e^A$.

Линеаризация функции $y_p = e^{ax+b}$:

- прологарифмируем функцию $\ln(y_p) = a\cdot x+b$;
- введем новую переменную $Y = \ln(y_p)$;
- получаем линейное уравнение $Y = a\cdot x+b$;
- рассчитываем a и b.

Линеаризация функции $y_p = \frac{1}{a\cdot x+b}$:

- введем новую переменную $Y = \ln(y_p)$;
- получаем линейное уравнение $Y = a\cdot x+b$;
- рассчитываем a и b.

Задание

По данным, приведенным в таблице:

x	1,08	1,63	1,04	1,49	0,97	0,90	0,77	0,69	0,57	0,62
y	0,49	0,49	0,46	0,52	0,38	0,33	0,34	0,34	0,33	0,28

Примечания. x — средняя заработная плата (тыс. руб.); y — прожиточный минимум на душу населения (тыс. руб.).

рассчитать параметры a и b по линейной, экспоненциальной и обратной моделям.

Линейная модель: $y_э = a*x+b$:

```
import math
def mnkl(n,x,y):
 z1=0; z2=0; b1=0; b2=0
 print("Линейная")
 for i in range(0, n):
   z1+=x[i]*x[i]
   z2+=x[i]
   b1+=x[i]*y[i]
   b2+=y[i]
 a=(b1*n–b2*z2)/(z1*n–z2**2)
 b=(b1–z1*a)/z2
 s=0
 for i in range(0, n):
   yp=a*x[i]+b
   d=abs(y[i]–yp)
   s+=d**2
   print("{0: 0.2f}  {1: 0.2f}   {2: 0.2f}  {3: 0.3f}".format(x[i], y[i], yp, d))
   rez=[0,0,0]
   rez[0]=a
   rez[1]=b
   rez[2]=s
 return rez
```

Экспоненциальная модель $y_э = e^{a \cdot x+b}$:

```
def mnke(n,x1,y1):
 z1=0; z2=0; b1=0; b2=0
 print("Экспоненциальная ")
 y=[]
 for i in y1:
   y.append (math.log(i))
```

```
 x=x1
 for i in range(0,n):
   z1+=x[i]*x[i]
   z2+=x[i]
   b1+=x[i]*y[i]
   b2+=y[i]
 a=(b1*n–b2*z2)/(z1*n–z2**2)
 b=(b1–z1*a)/z2
 b=math.exp(b)
 s=0
 for i in range(0,n):
   yp=b*math.exp(a*x[i])
   d=abs(y1[i]–yp)
   s+=d**2
   print("{0: 0.2f}  {1: 0.2f}   {2: 0.2f}  {3: 0.3f}".format(x[i], y1[i], yp, d))
   rez=[0,0,0]
   rez[0]=a
   rez[1]=b
   rez[2]=s
 return rez
```

Обратная модель $y_э = \frac{1}{a*x+b}$:

```
def mnkg(n,x1, y1):
 z1=0; z2=0; b1=0; b2=0
 print("Обратная")
 y=[]
 for i in y1:
   y.append (1/i)
 x=x1
 for i in range(0, n):
   z1+=x[i]*x[i]
   z2+=x[i]
```

```
    b1+=x[i]*y[i]
    b2+=y[i]
  a=(b1*n–b2*z2)/(z1*n–z2**2)
  b=(b1–z1*a)/z2
  s=0
  for i in range(0, n):
    yp=1/(a*x[i]+b)
    d=abs(y1[i]–yp)
    s+=d**2
    print("{0: 0.2f}  {1: 0.2f}   {2: 0.2f}  {3: 0.3f}".format(x[i], y1[i], yp, d))
    rez=[0,0,0]
    rez[0]=a
    rez[1]=b
    rez[2]=s
  return rez
```

Для исходных данных:

```
import MNK1
x=[64, 68, 82, 76, 84, 96, 100]
y=[64, 56, 52, 48, 50, 46, 38]
n=7
a,b,s=MNK1.mnkl(n,x,y)
print ('a={0: 0.3f} b={1: 0.3f}'.format(a,b))
print("s={0:0.3f}".format(s))
print ('----------------------')
a,b,s=MNK1.mnke(n,x,y)
print ('a={0: 0.3f} b={1: 0.3f}'.format(a,b))
print("s={0:0.3f}".format(s))
print ('----------------------')
a,b,s=MNK1.mnkg(n,x,y)
print ('a={0: 0.5f} b={1: 0.5f}'.format(a,b))
print("s={0:0.3f}".format(s))
```

Результат

Линейная

64.00	64.00	60.17	3.827
68.00	56.00	57.97	1.969
82.00	52.00	50.26	1.743
76.00	48.00	53.56	5.562
84.00	50.00	49.15	0.845
96.00	46.00	42.54	3.456
100.00	38.00	40.34	2.340

$a = -0.551$, $b = 95.431$,

$s = 70.636$

Экспоненциальная

64.00	64.00	60.61	3.391
68.00	56.00	57.99	1.992
82.00	52.00	49.69	2.309
76.00	48.00	53.09	5.092
84.00	50.00	48.61	1.393
96.00	46.00	42.58	3.422
100.00	38.00	40.74	2.740

$a = -0.011$, $b = 122.811$,

$s = 67.886$

Обратная

64.00	64.00	61.34	2.665
68.00	56.00	58.12	2.124
82.00	52.00	49.12	2.878
76.00	48.00	52.61	4.614
84.00	50.00	48.06	1.941
96.00	46.00	42.53	3.465
100.00	38.00	40.97	2.965

a = 0.00023, b= 0.00189,

s = 65.754

Для исходных данных:

```
import MNK1
x=[1.08,1.63,1.04,1.49,0.97,0.9,0.77,0.69,0.57,0.62]
y=[0.49,0.49,0.46,0.52,0.38,0.33,0.34,0.34,0.33,0.28]
n=10
a,b,s=MNK1.mnkl(n,x,y)
print ('a={0: 0.3f} b={1: 0.3f}'.format(a,b))
print("s={0:0.3f}".format(s))
print ('----------')
a,b,s=MNK1.mnkl(n,x,y)
print ('a={0: 0.3f} b={1: 0.3f}'.format(a,b))
print("s={0:0.3f}".format(s))
print ('----------')
a,b,s=MNK1.mnkl(n,x,y)
print ('a={0: 0.3f} b={1: 0.3f}'.format(a,b))
print("s={0:0.3f}".format(s))
```

Результат

Линейная

1.08	0.49	0.42	0.072
1.63	0.49	0.54	0.045
1.04	0.46	0.41	0.050
1.49	0.52	0.51	0.015
0.97	0.38	0.39	0.015
0.90	0.33	0.38	0.050
0.77	0.34	0.35	0.012
0.69	0.34	0.34	0.005
0.57	0.33	0.31	0.020
0.62	0.28	0.32	0.040

a = 0.213, b = 0.189,

s = 0.015

Экспоненциальная

1.08 0.49 0.41 0.080
1.63 0.49 0.55 0.058
1.04 0.46 0.40 0.059
1.49 0.52 0.51 0.011
0.97 0.38 0.39 0.007
0.90 0.33 0.37 0.043
0.77 0.34 0.35 0.008
0.69 0.34 0.33 0.007
0.57 0.33 0.31 0.017
0.62 0.28 0.32 0.041

$a = 0.530$, $b = 0.231$

$s = 0.017$

Обратная

1.08 0.49 0.40 0.089
1.63 0.49 0.57 0.081
1.04 0.46 0.39 0.067
1.49 0.52 0.52 0.005
0.97 0.38 0.38 0.001
0.90 0.33 0.37 0.036
0.77 0.34 0.34 0.004
0.69 0.34 0.33 0.009
0.57 0.33 0.31 0.015
0.62 0.28 0.32 0.041

a= –1.347, $b = 3.947$,

$s = 0.022$

Задание

По данным, приведенным в таблице:

x	10	21	34	40	56	60	70	80	90	100
y	0,01	0,08	0,14	0,21	0,38	0,45	0,52	0,61	0,75	0,91

Примечания. x — температура, y — скорость некоторой химической реакции.

рассчитать параметры a и b по линейной $y_p = ax+b$, экспоненциальной $y_p=e^{ax+b}$ и обратной $y_p = \frac{1}{a \cdot x+b}$ моделям. Указать лучшую модель.

Линейная модель $y_p = a*x+b$:

Текст функции	Результат

def mnkl(*n*,*x*,*y*):
  *z*1=0; *z*2=0; *b*1=0; *b*2=0
  print("Линейная")
  for i in range(0,*n*):
  *z*1+=*x*[*i*]*x[*i*]
  *z*2+=*x*[*i*]
  *b*1+=*x*[*i*]*y[*i*]
  *b*2+=*y*[*i*]
  a=(*b*1*n–*b*2*z2)/(*z*1*n–*z*2**2)
  b=(*b*1–*z*1*a)/*z*2
  s=0
  for i in range(0,*n*):
  yp=*a*x[*i*]+*b*
  d=*abs*(*y*[*i*]–*yp*)
  s+=*d*2
  print("{0: 0.2*f*} {1: 0.2*f*} {2: 0.2*f*} {3: 0.3*f*}".
format(*x*[*i*],*y*[*i*], *yp*, *d*))
  rez=[0,0,0]
  rez[0]=*a*
  rez[1]=*b*
  rez[2]=*s*
  return rez
	Линейная
	10.00 0.01 –0.05 0.062
	21.00 0.08 0.06 0.023
	34.00 0.14 0.19 0.046
	40.00 0.21 0.25 0.036
	56.00 0.38 0.41 0.025
	60.00 0.45 0.44 0.005
	70.00 0.52 0.54 0.024
	80.00 0.61 0.64 0.034
	90.00 0.75 0.74 0.007
	100.00 0.91 0.84 0.068
	a = 0.010, b = –0.152,
	s = 0.015

Экспоненциальная модель $y_p=e^{ax+b}$:

Текст функции	Результат

def mnke(*n*,*x*1,*y*1):
  *z*1=0; *z*2=0; *b*1=0; *b*2=0
  print("Экспоненциальная")
	Экспоненциальная
	10.00 0.01 0.04 0.026
	21.00 0.08 0.06 0.023

Продолжение табл.

Текст функции	Результат
y=[] *for i in y*1*:* *y.append* (*math.log*(*i*)) *x*=*x*1 *for i in range*(0,*n*)*:* *z*1+=*x*[*i*]**x*[*i*] *z*2+=*x*[*i*] *b*1+=*x*[*i*]**y*[*i*] *b*2+=*y*[*i*] *a*=(*b*1**n*–*b*2**z*2)/(*z*1**n*–*z*2**2) *b*=(*b*1–*z*1**a*)/*z*2 *b*=*math.exp*(*b*) *s*=0 *for i in range*(0,*n*)*:* *yp*=*b***math.exp*(*a***x*[*i*]) *d*=*abs*(*y*1[*i*]–*yp*) *s*+=*d***2 *print*("{0: 0.2*f*} {1: 0.2*f*} {2: 0.2*f*} {3: 0.3*f*}".*format*(*x*[*i*], *y*1[*i*], *yp*, *d*)) *rez*=[0,0,0] *rez*[0]=*a* *rez*[1]=*b* *rez*[2]=*s* *return rez*	34.00 0.14 0.10 0.041 40.00 0.21 0.13 0.083 56.00 0.38 0.25 0.133 60.00 0.45 0.29 0.159 70.00 0.52 0.44 0.078 80.00 0.61 0.67 0.061 90.00 0.75 1.02 0.267 100.00 0.91 1.54 0.633 a = 0.042, b = 0.024, s = 0.535

Обратная модель $y_э = \frac{1}{a*x+b}$:

Текст функции	Результат
def mnkg(*n*,*x*1,*y*1)*:* *z*1=0; *z*2=0; *b*1=0; *b*2=0 *print*("Обратная") *y*=[] *for i in y*1*:* *y.append* (1/*i*) *x*=*x*1	Обратная 10.00 0.01 0.02 0.013 21.00 0.08 0.03 0.052 34.00 0.14 0.04 0.104 40.00 0.21 0.04 0.168 56.00 0.38 0.07 0.306

Продолжение табл.

Текст функции	Результат

for i in range(0,*n*):
  *z*1+=*x*[*i*]**x*[*i*]
  *z*2+=*x*[*i*]
  *b*1+=*x*[*i*]**y*[*i*]
  *b*2+=*y*[*i*]
a=(*b*1**n*–*b*2**z*2)/(*z*1**n*–*z*2**2)
b=(*b*1–*z*1**a*)/*z*2
s=0
for i in range(0,*n*):
  yp=1/(*a***x*[*i*]+*b*)
  d=*abs*(*y*1[*i*]–*yp*)
  s+=*d***2
print("{0: 0.2*f*} {1: 0.2*f*} {2: 0.2*f*} {3: 0.3*f*}".*format*(*x*[*i*], *y*1[*i*], *yp*, *d*))
  rez=[0,0,0]
  rez[0]=*a*
  rez[1]=*b*
  rez[2]=*s*
  return rez
	60.00 0.45 0.09 0.359
	70.00 0.52 0.22 0.302
	80.00 0.61 –0.55 1.156
	90.00 0.75 –0.12 0.871
	100.00 0.91 –0.07 0.978
	a = –0.643, *b* = 49.574,
	s = 3.408

Ответ: наименьшее значение остаточной дисперсии *s* у линейной модели, следовательно, для данного расчета она является лучшей.

Графическая интерпретация данных. Библиотеки numpy, matplotlib, pandas

В Python есть много возможностей для графической интерпретации данных.

Библиотека *numpy* является одной из наиболее часто используемых библиотек для реализации алгоритмов вычислительной математики, анализа временных рядов и визуализации данных в форме массивов, в частности, представляемыми списками и кортежами.

Библиотека *matplotlib* предназначена для построения графиков, гистограмм различных видов и других способов графической интерпретации данных. Графики могут быть представлены в 2D и 3D-форматах.

Библиотека *pandas* широко используется для анализа данных (*Data Mining*). Данные в ней реализуются в виде структур *Series* и *DataFrame*. Пакет также содержит ряд методов для фильтрации данных и модули выполнения операций ввода-вывода.

Задача 14

Составить программу для построения семейства графиков функций:

$$f1(x) = \sin(0{,}8x)^2 + e^{\cos(x)}$$

$$f2(x) = e^{\cos(2x)} - 3\sin(0{,}5x) + 0{,}4$$

Программа

Текст модуля *Grafik.py*

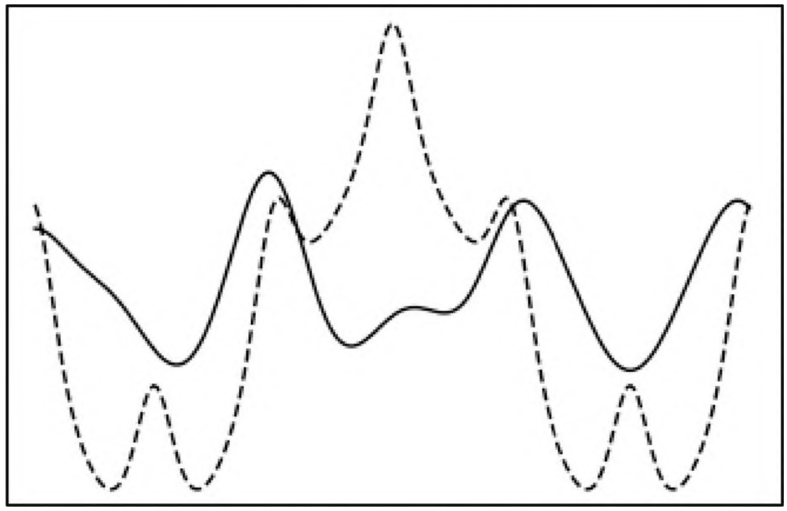
```
import numpy as np, matplotlib.pyplot as plt
# Семейство графиков
def grf1(x):
   return np.sin(0.8*x)**2+np.exp(np.cos(x))
def grf2(x):
   return np.exp(np.cos(2*x))–3*np.sin(0.5*x)+0.4
```

```
xn=0
xk=np.pi*6
nn=200
X=np.linspace(xn,xk,nn)
Y,Z=grf1(X),grf2(X)
plt.plot(X,Y)
plt.plot(X,Z)
plt.show()
```

Модуль *main.py*

```
import Grafik
```

Результат



Задача 15

Составить программу для построения семейства графиков функций:

$$f1 = -0.12x$$

$$f2 = 10\sin(0.2x)$$

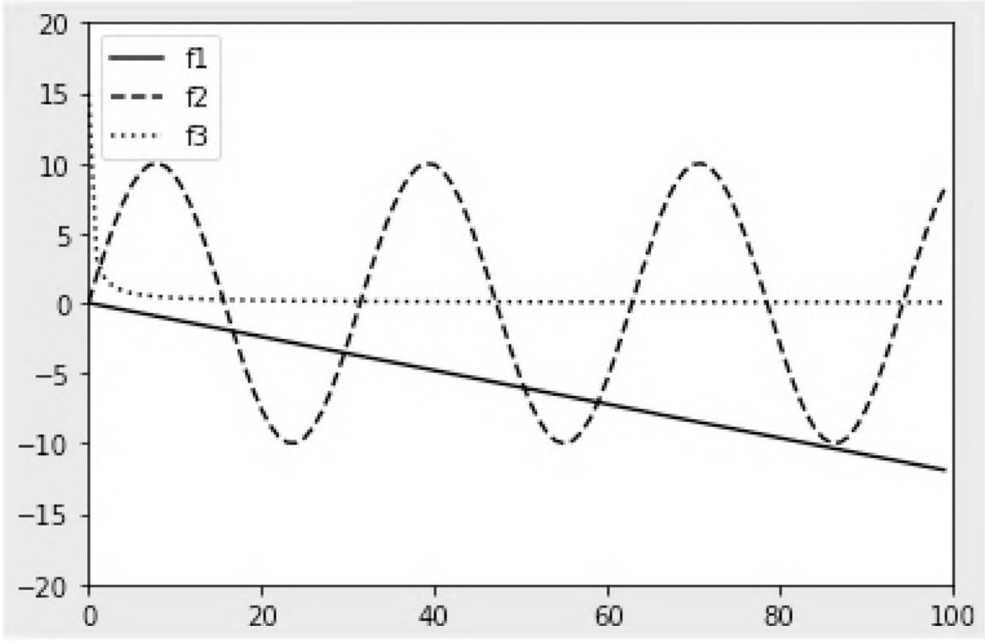
$$f3 = 15/(1 + 4{,}1 \cdot x)$$

с использованием структуры *DataFrame*.

Программа

```
#с использованием фреймов
import pandas as pd
import matplotlib.lines
import matplotlib.pyplot as plt
import numpy as np
data = {'f1':[–0.12*x for x in range(100)],
        'f2':[10*np.sin(0.2*x) for x in range(100)],
        'f3':[15/(1+4.1*x) for x in range(100)]}
df = pd.DataFrame(data)
plt.axis([0,100,–20,20])

plt.plot(df["f1"],'-k',df["f2"],'--k',df["f3"],':k')
plt.legend(data, loc=2) # вывод легенды
plt.show()
```

Результат



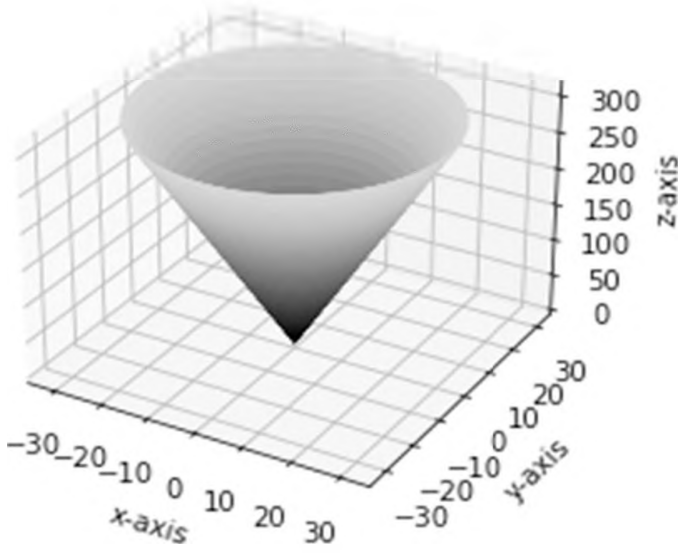
Задача 16

Составить программу для построения перевернутого конуса в 3D-формате.

Программа

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 5 * np.outer(np.cos(u), v)
y = 10 * np.outer(np.sin(u), v)
z = 50 * np.outer(np.ones(np.size(u)), v)
ax.plot_surface(x, y, z,  rstride=4, cstride=4, cmap = cm.copper)
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
plt.show()
```

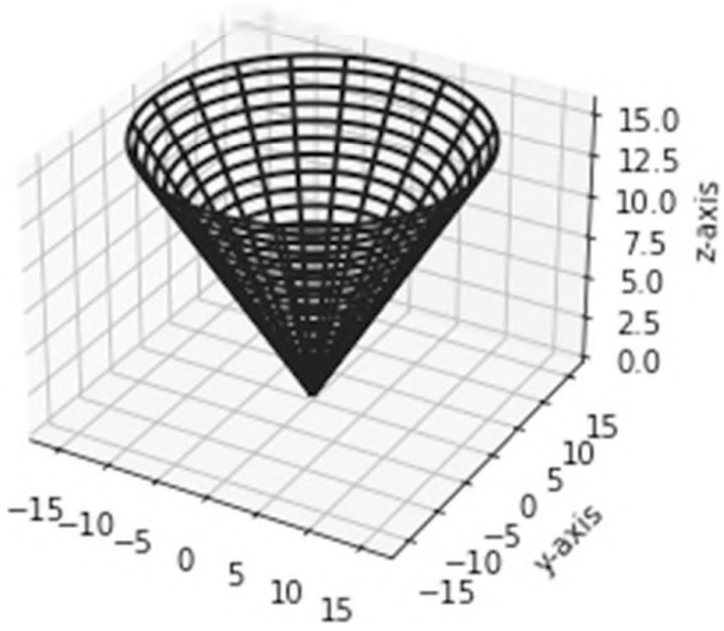
Результат



Программа

```
import matplotlib.pyplot as plt
from matplotlib import cm
import numpy as np
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
u = np.linspace(0, 2 * np.pi, 100)
v = np.linspace(0, np.pi, 100)
x = 5 * np.outer(np.cos(u), v)
y = 5 * np.outer(np.sin(u), v)
z = 5 * np.outer(np.ones(np.size(u)), v)
ax.plot_wireframe(x, y, z, rstride=5, cstride=5, color= (0.1,0.1,0.1),
linewidth=2)
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
plt.show()
```

Результат



Построение рисунков. Библиотека turtle

Черепашья графика входила в язык программирования Logo, разработанный Уолли Фейрцейгом (Wally Feurzeig), Сеймуром Папертом (Seymour Papert) и Синтией Соломон (Cynthia Solomon) в 1967 году. Рабочим инструментом (карандашом) является черепашка (*turtle*), которая по умолчанию находится в точке с координатами (0, 0) в плоскости *x*-*y* и может выполнять ряд команд:

forward(), или *fd*() — черепашка перемещается вперед на заданное число пикселей;

backward(), или *bk*(), или *back*() — черепашка перемещается назад на заданное число пикселей;

right(), или *rt*() — повернуть черепашку вправо на заданное число градусов;

left(), или *lt*() — повернуть черепашку влево на заданное число градусов;

dot() — рисовать точку;

circle() — рисовать окружность;

position(), или *pos*() — текущее положение карандаша на холсте;

xcor(), *ycor*() — координата *x* карандаша на холсте и координата *y* соответственно;

pendown(), или *pd*(), или *down*() — опустить карандаш, начать рисовать;

penup(), или *pu*(), или *up*() — поднять карандаш, перестать рисовать;

pensize(), или *width*() — установить толщину карандаша;

pencolor() — установить цвет карандаша;

fillcolor() — установить цвет заливки;

begin_fil() — начало заливки;

end_fill() — окончание заливки.

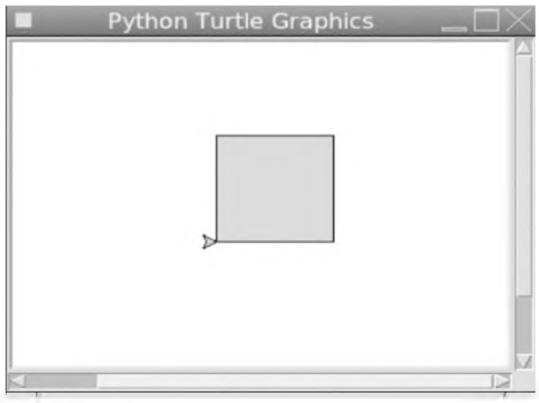
Задача 17

Составить программу для построения квадрата.

Программа

```
main.py
import turtle
from turtle import *
color('blue', 'pink') # голубой цвет карандаша, розовый для заливки
begin_fill() # начать заливку
for i in range (4):
  turtle.fd(80) # вперед на 80 пикселей
  turtle.left(90) # повернуть влево на 90 градусов
end_fill() # закончить заливку
```

Результат



Задача 18

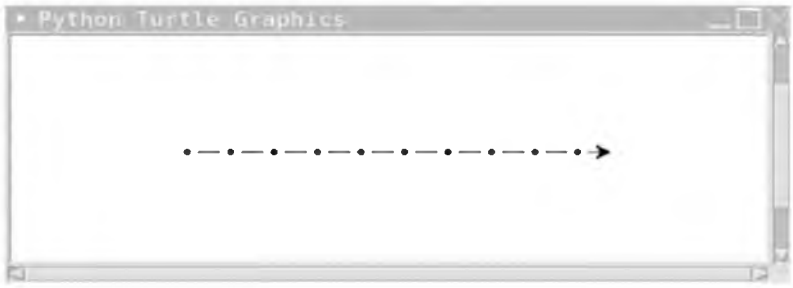
Составить программу для построения штрихпунктирной линии.

Программа

```
import turtle
turtle.penup() # поднять карандаш
turtle.left (180)
turtle.pendown() # опустить карандаш
for i in range (5):
  turtle.pendown()
```

```
  turtle.forward(15)
  turtle.penup()
  turtle.forward(8)
  turtle.pendown()
  turtle.dot()
  turtle.penup()
  turtle.forward(8)
turtle.home()
for i in range (5):
  turtle.penup()
  turtle.forward(8)
  turtle.pendown()
  turtle.dot()
  turtle.penup()
  turtle.forward(8)
  turtle.pendown()
  turtle.forward(15)
```

Результат



Задача 19

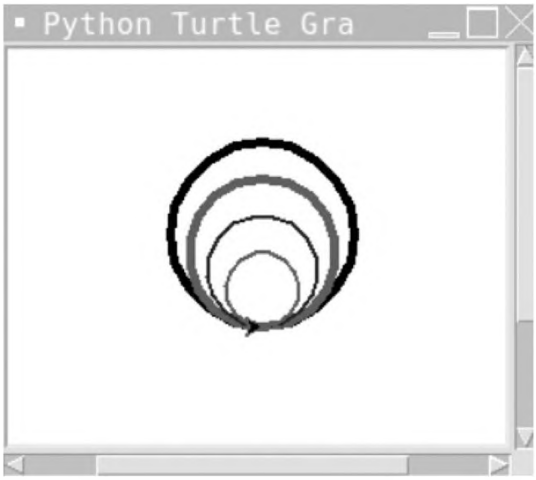
Составить программу для построения окружностей.

Программа

```
import turtle
turtle.pensize(5) # толщина карандаша пять пикселей
turtle.circle(50) # окружность радиуса 50 пикселей
turtle.pencolor('red') #  строковое представление цвета
```

```
turtle.circle(40)
turtle.pensize(2)
turtle.pencolor("blue")
turtle.circle(30)
turtle.pencolor("green")
turtle.circle(20)
```

Результат



Построение фракталов

У понятия «фрактал» нет строгого определения, поэтому слово «фрактал» не является математическим термином. Слово фрактал образовано от латинского fractus и в переводе означает «состоящий из фрагментов». Обычно так называют геометрическую фигуру. По определения Бенуа Мандельброта, «фракталом называется структура, состоящая из частей, которые в каком-то смысле подобны целому».

Роль фракталов в машинной графике сегодня достаточно велика. Они приходят на помощь, например, когда требуется, с помощью нескольких коэффициентов, задать линии и поверхности очень сложной формы. С точки зрения машинной графики, фрактальная геометрия незаменима при генерации искусственных облаков, гор, поверхности моря.

Из определения Мандельброта следует одно из основных свойств фракталов — самоподобие, то есть небольшая часть фрактала, содержит информацию обо всем фрактале. Фрактал строится рекурсивно из фигур, имеющих разный масштаб.

Понятие *L*-системы было введено А. Лидермайером. Формально *L*-система состоит из алфавита, аксиомы (инициализатора) и набора порождающих правил. В алфавит, в частности, входят символы:

F — перемещение на один шаг вправо, прорисовывая след;

b — перемещение на один шаг вправо, не прорисовывая след;

+ — увеличение угла на заданную величину;

– — уменьшение угла на заданную величину.

Терл-графика является подсистемой вывода графического представления фрактального объекта.

Задача 20

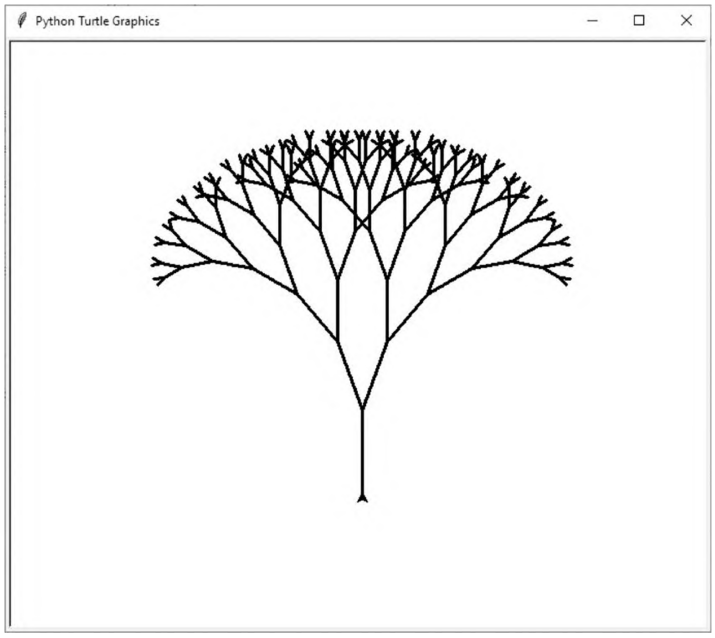
Составить программу для рисования дерева.

Программа

```
import turtle
def tree(branchLen,t):
    if branchLen > 3:
        t.forward(branchLen)
        t.right(20)
        tree(branchLen–10,t)
        t.left(40)
        tree(branchLen–10,t)
        t.right(20)
        t.backward(branchLen)
t = turtle.Turtle()
myWin = turtle.Screen() # определить холст
t.pencolor((0.001, 0.001, 0.001))
t.left(90)
t.up()
```

```
t.backward(125)
t.down()
t.color("green")
tree(80,t)
myWin.exitonclick()
```

Результат



Из рисунка видно, что дерево построено из однотипных элементов:

Задача 21

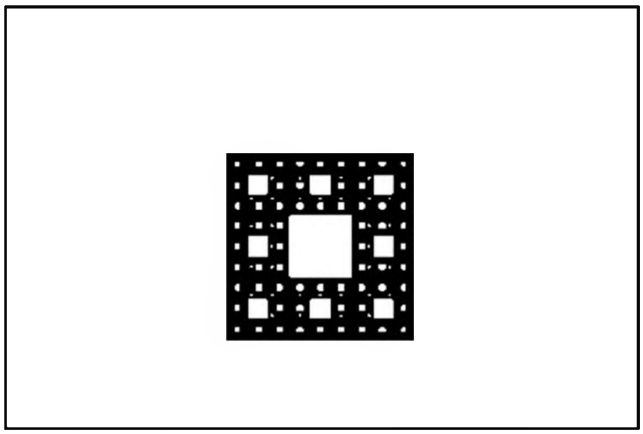
Составить программу для рисования ковра Серпинского.

Программа

```
import turtle
def s(n, m):
    if n == 0:
        turtle.color('deeppink')
```

```
        turtle.begin_fill()
        for _ in range(4):
            turtle.forward(m)
            turtle.left(90)
        turtle.end_fill()
    else:
        for _ in range(4):
            s(n – 1, m / 3)
            turtle.forward(m / 3)
            s(n – 1, m / 3)
            turtle.forward(m / 3)
            turtle.forward(m / 3)
            turtle.left(90)
turtle.ht()
turtle.tracer(10)
s(4, 100)
turtle.done()
```

Результат для числа итераций 4



Из рисунка видно, что ковер построен из однотипных элементов:

— число итераций 0:

— число итераций 1:

Задача 22

Составить программу для рисования фрактала Вичека

Программа

Модуль *B*1*.py*

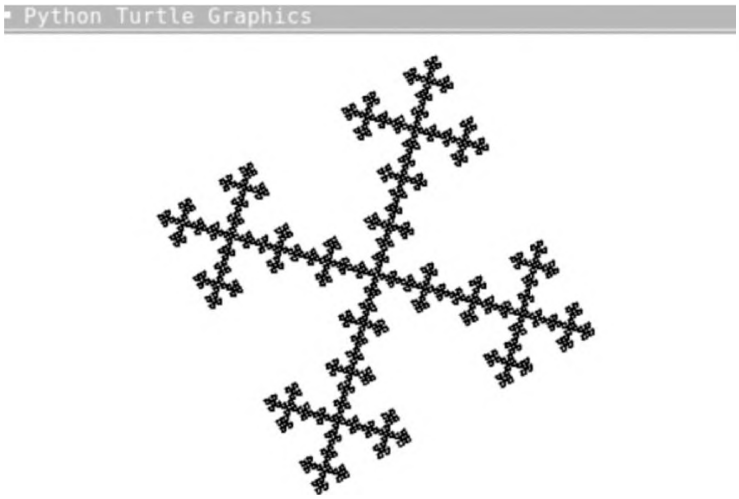
```
import turtle
def create_l_system(iters, axiom, rules):
    start_string = axiom
    if iters == 0:
        return axiom
    end_string = ""
    for _ in range(iters):
        end_string = "".join(rules[i] if i in rules else i for i in start_string)
        start_string = end_string
    return end_string
def draw_l_system(t, instructions, angle, distance):
    for cmd in instructions:
        if cmd == 'F':
            t.forward(distance)
        elif cmd == '+':
            t.right(angle)
        elif cmd == '-':
            t.left(angle)
axiom = "F--F--F"
rules = {"F":"F+F--F+F"}
iterations = 4
angle = 60
length=3
size=2
y_offset=–30
x_offset=100
```

```
offset_angle=30
width=450
height=450
inst = create_l_system(iterations, axiom, rules)
t = turtle.Turtle()
print(t)
wn = turtle.Screen()
wn.setup(width, height)
t.up()
t.backward(-x_offset)
t.left(90)
t.backward(–y_offset)
t.left(offset_angle)
t.down()
t.speed(0)
t.pensize(size)
draw_l_system(t, inst, angle, length)
t.hideturtle()
wn.exitonclick()
```

Модуль *main.py*

```
import B1
```

Результат для аксиомы *F--F--F* и порождающего правила *F+F--F+F*



Задача 23

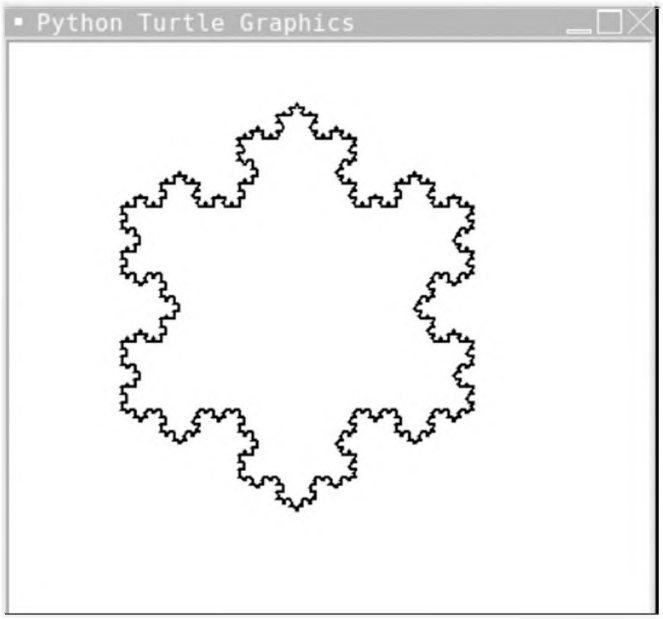
Составить программу для рисования снежинки Коха.

Программа

```
import turtle
def create_l_system(iters, axiom, rules):
    start_string = axiom
    if iters == 0:
        return axiom
    end_string = ""
    for _ in range(iters):
        end_string = "".join(rules[i] if i in rules else i for i in start_string)
        start_string = end_string
    return end_string
def draw_l_system(t, instructions, angle, distance):
    for cmd in instructions:
        if cmd == 'F':
            t.forward(distance)
        elif cmd == '+':
            t.right(angle)
        elif cmd == '-':
            t.left(angle)
axiom = "F++F++F"
rules = {"F":"F-F++F-F"}
iterations = 4
angle = 60
length=3
size=2
y_offset=–100
x_offset=–25
offset_angle=30
width=450
height=450
inst = create_l_system(iterations, axiom, rules)
t = turtle.Turtle()
```

```
wn = turtle.Screen()
wn.setup(width, height)
t.up()
t.backward(-x_offset)
t.left(90)
t.backward(-y_offset)
t.left(offset_angle)
t.down()
t.speed()
t.pensize(size)
draw_l_system(t, inst, angle, length)
t.hideturtle()
wn.exitonclick()
```

Результат



Основы функционального программирования

Функциональное программирование — парадигма программирования, в которой процесс вычисления трактуется как вычисление значений функций в математическом понимании последних (в отличие от функций как подпрограмм в процедурном программировании).

Противопоставляется парадигме императивного программирования, которая описывает процесс вычислений как последовательное изменение состояний. При необходимости в функциональном программировании вся совокупность последовательных состояний вычислительного процесса представляется явным образом, например как список.

Функциональное программирование предполагает обходиться вычислением результатов функций от исходных данных и результатов других функций, и не предполагает явного хранения состояния программы. Соответственно, не предполагает оно и изменяемость этого состояния (в отличие от императивного, где одной из базовых концепций является переменная, хранящая своё значение и позволяющая менять его по мере выполнения алгоритма).

На практике отличие математической функции от понятия функции в императивном программировании заключается в том, что императивные функции могут опираться не только на аргументы, но и на состояние внешних по отношению к функции переменных, а также иметь побочные эффекты и менять состояние внешних переменных. Таким образом, в императивном программировании при вызове одной и той же функции с одинаковыми параметрами, но на разных этапах выполнения алгоритма, можно получить разные данные на выходе из-за влияния на функцию состояния переменных. А в функциональном языке при вызове функции с одними и теми же аргументами всегда получаем одинаковый результат: выходные данные зависят только от входных. Это позволяет средам выполнения программ на функциональных языках кешировать результаты функций и вызывать их в порядке, не определяемом алгоритмом и распараллеливать их без каких-

либо дополнительных действий со стороны программиста (что обеспечивают функции без побочных эффектов — чистые функции). Чистые функции обладают несколькими полезными свойствами, многие из которых можно использовать для оптимизации кода. Результат вызова чистой функции может быть сохранён в таблице значений вместе с аргументами вызова. За счет этого если функция повторно вызывается с этими же аргументами, её результат может быть взят прямо из таблицы значений без вычислений (иногда это называется принципом прозрачности ссылок). Цена такого подхода увеличение расхода памяти, позволяет существенно увеличить производительность и уменьшить порядок роста некоторых рекурсивных алгоритмов.

Лямбда-исчисление являются основой для функционального программирования, многие функциональные языки можно рассматривать как «надстройку» над ними.

Функции высших порядков — это такие функции, которые могут принимать в качестве аргументов и возвращать другие функции. В математике такие функции чаще всего называют операторами, например, оператор взятия производной или оператор интегрирования.

Преимущества функционального программирования:

- повышение надёжности кода;
- возможности оптимизации при компиляции;
- возможности параллелизма;
- удобство организации модульного тестирования.

Недостатки: отсутствие присваиваний и замена их на порождение новых данных приводят к необходимости постоянного выделения и автоматического освобождения памяти, поэтому в системе исполнения функциональной программы обязательным компонентом становится высокоэффективный сборщик мусора. Нестрогая модель вычислений приводит к непредсказуемому порядку вызова функций, что создаёт проблемы при вводе-выводе, где порядок выполнения операций важен.

В Python включено много инструментов, ориентированных на функциональный стиль.

Функция *zip*()

Встроенная функция *zip* объединяет отдельные элементы из каждой последовательности в кортежи, то есть она возвращает итерируемую последовательность, состоящую из кортежей.

zip (последовательность, последовательность, ...)

Последовательность — это итерируемая последовательность, то есть список, кортеж, диапазон или строковые данные.

Функция *map*()

Встроенная в Python функция *map* — это функция более высокого порядка, которая позволяет обрабатывать одну или несколько последовательностей с использованием заданной функции.

map (функция, последовательности)

Функция — это ссылка на стандартную *def*-функцию, либо лямбда-функция. Последовательности — это одна или несколько отделенных запятыми итерируемых последовательностей.

Объект *map* вычисляется во время преобразования в список.

Пример

```
lambda_func = lambda x, y: x + y
rez=lambda_func(5.5, –7)
print ("rez=",rez)
print((lambda x, y: x+y)(5, 7))
seq1 = (1, –2, 3, 4, -5, 6, 7, 8, –9)
seq2 = (–5, 6, 7, 8, 9, 0, 3, 2, 1)
rez1=lambda_func(seq1, seq2)
print ("rez1=",rez1)
print((lambda x, y: x+y)(seq1, seq2))
sp11 = [x + y for x, y in zip(seq1, seq2)]
```

```
print("sp11=",sp11)
result = map(lambda_func, seq1, seq2)
print("list(result)=",list(result))
```

Результат

```
rez= -1.5
12
rez1= (1, -2, 3, 4, -5, 6, 7, 8, -9, -5, 6, 7, 8, 9, 0, 3, 2, 1)
(1, -2, 3, 4, -5, 6, 7, 8, -9, -5, 6, 7, 8, 9, 0, 3, 2, 1)
sp11= [-4, 4, 10, 12, 4, 6, 10, 10, -8]
list(result)= [-4, 4, 10, 12, 4, 6, 10, 10, -8]
>
```

Рассмотрим, как работает этот пример.

rez=lambda_func(5.5, –7) — вызов лямбда-функции с сохранением результата как объекта *rez*. При выводе объекта *rez* выводится на экран значение –1,5. Значение вещественного типа определяется при выполнении лямбда-функции исходя из типов аргументов: 5.5 — вещественное число и –7 — целое. Целое автоматически переводится в вещественное значение, и операция выполняется.

print((*lambda x, y: x+y*)(5, 7)) — лямбда-функция и ее вызов являются параметрами функции *print*(). Результат 12 — целое число, так как оба параметра лямбда-функции 5 и 7 целые числа.

*rez*1=*lambda_func*(*seq*1, *seq*2) — вызов лямбда-функции с сохранением результата как объекта *rez*1. Объект *rez*1 — это новая последовательность, включающая последовательности *seq*1, *seq*2 в соответствии с правилами выполнения операции сложения последовательностей.

print((*lambda x, y: x+y*)(*seq*1, *seq*2)) — операция сложения последовательностей, реализованная описанием лямбда-функции и ее вызовом как параметрами функции *print*().

*sp*11 = [*x* + *y for x, y in zip*(*seq*1, *seq*2)] — использование функции *zip* для формирования новой последовательности, которая, в отличие от вышеприведенного вызова, выполняет поэлементное сложение элементов последовательностей *seq*1, *seq*2.

result = *map*(*lambda_func*, *seq*1, *seq*2) — использование функции *map*() для формирования новой последовательности с помощью лямбда-функции *lambda_func*() для поэлементного сложения элементов последовательностей *seq*1, *seq*2.

Заключение

Изучение основ алгоритмизации и программирования на Python является важной частью подготовки студентов инженерных специальностей. Знание базовых конструкций языка и умение их применять для решения задач математического моделирования позволяет последовательно от простого к сложному научиться понимать сущность исследовательской и научной деятельности. Сначала рассматриваются примеры простейших программ в императивном стиле программирования и примеры решения несложных задач линейной, разветвляющейся и циклической структур, задач с последовательностями и файлами. Далее даны алгоритмы методов вычислительной математики и их реализация в виде программ на Python. Задания по аппроксимации парной регрессией служат примерами исследовательской деятельности студентов младших курсов. Практика использования библиотек *numpy*, *matplotlib*, *pandas* и *turtle* повышает комфортность работы, в частности за счет графической интерпретации данных. Краткое описание основ функционального программирования и примеры конструкций, поддерживающих этот стиль программирования, способствуют дальнейшему освоению языка Python, существенно облегчают переход к работе с пакетами прикладных программ для решения задач технических вычислений и являются базой для изучения профессиональных дисциплин, использующих компьютерные технологии, а также при выполнении курсовых и дипломных работ.

Список литературы

1. *Никитина, Т. П.* Основы программирования на Python: учебное пособие / Т. П. Никитина. — Ярославль : Изд-во ЯГТУ, 2021. — 96 с.

2. *Прохоренок, Н. А.* Python 3. Самое необходимое / Н. А. Прохоренок, В. А. Дронов. — СПб. : БХВ-Петербург, 2016. — 464 с.

3. *Федоров, Д. Ю.* Программирование на языке высокого уровня Python : учебное пособие для вузов / Д. Ю. Федоров. — 3-е изд., перераб. и доп. — М. : Юрайт, 2021. — 210 с.

Оглавление

Татьяна Павловна НИКИТИНА,
Леонид Владимирович КОРОЛЕВ

ПРОГРАММИРОВАНИЕ. ОСНОВЫ PYTHON ДЛЯ ИНЖЕНЕРОВ

Учебное пособие

Зав. редакцией
литературы по информационным технологиям
и системам связи *О. Е. Гайнутдинова*
Ответственный редактор *В. В. Яески*
Корректор *Т. А. Быченкова*
Выпускающий *В. А. Иутин*

ЛР № 065466 от 21.10.97
Гигиенический сертификат 78.01.10.953.П.1028
от 14.04.2016 г., выдан ЦГСЭН в СПб

Издательство «ЛАНЬ»
lan@lanbook.ru; www.lanbook.com;
196105, Санкт-Петербург, пр. Юрия Гагарина, д. 1, лит. А
Тел.: (812) 412-92-72, 336-25-09.
Бесплатный звонок по России: 8-800-700-40-71

Подписано в печать 25.11.22.
Бумага офсетная. Гарнитура Школьная. Формат 60×90 $^{1}/_{16}$.
Печать офсетная/цифровая. Усл. п. л. 9,75. Тираж 30 экз.

Заказ № 006-23.

Отпечатано в полном соответствии
с качеством предоставленного оригинал-макета
в АО «Т8 Издательские Технологии».
109316, г. Москва, Волгоградский пр., д. 42, к. 5.