# 5 Функции (часть I)

Функция — это изолированный блок кода, который решает отдельную задачу. Использование функций делает программу более простой и легко читаемой, а также позволяет избежать ненужного повторения кода. Мы уже использовали стандартные функции Python, такие как `input()`, `len()`, `print()` и т. д. Теперь рассмотрим процесс создания собственных функций.

## 5.1 Описание функции

Описание функции

```
def имя_функции([список параметров]):
    '''
    документация функции,
    состоящая из одной
    или нескольких строк
    '''

    тело функции
    [return значение_функции] # в […] отмечена необязательная часть
```

Здесь:

- def — ключевое слово, с которого начинается заголовок функции (её имя и список параметров). В конце заголовка функции ставится двоеточие.
- имя_функции — уникальное имя функции. Также как и имена идентификаторов, оно может начинаться только с буквы или знака подчеркивания и не должно содержать пробелов и других специальных символов.
- список параметров — список аргументов функции, разделённых запятыми (может быть пустым, но круглые скобки надо ставить обязательно).
- тело функции — блок инструкций, выполняющийся при вызове функции. Тело функции начинается с новой строки с отступом в 4 пробела и может содержать любые операторы, в том числе условные и операторы циклов. Подключения библиотек внутри тела функции быть не должно!
- return — необязательный оператор, с помощью которого можно прекратить выполнение функции. Если инструкция return используется без аргументов или не используется вообще, то функция будет возвращать значение None. Функции

в Python могут возвращать значения любого типа. После выполнения инструкции return произойдет выход из функции, и программа продолжит свою работу, начиная с того места, где была вызвана эта функция. В теле функции может быть несколько инструкций return (например, в разных ветках условного оператора). В этом случае функция завершит свою работу, как только выполнится первый из них.

- документация функции — необязательный раздел, содержащий информацию о функции. Здесь может быть описано назначение функции, её входные и выходные параметры.

Для того чтобы функция возвращала более одного значения, после оператора return следует записать несколько переменных через запятую. В этом случае функция вернёт кортеж из двух переменных. Причём результат вызова такой функции можно использовать во множественном присваивании.

Все функции должны быть описаны до своего вызова, т. е. выше текста основной программы. Внутри одной функции можно вызывать любую другую функцию. Несколько пользовательских функций можно объединять в пользовательские модули. Подключить пользовательский модуль можно также, как и стандартные библиотеки Python, используя инструкцию import.

## 5.2 Вызов функции

Для вызова функции необходимо указать её имя и параметры вызова в круглых скобках. Список параметров может быть пустым, в этом случае при вызове функции необходимо оставить пустые круглые скобки. Если функция содержит инструкцию return, то результат её вызова необходимо присвоить переменной. Стоит различать понятия формальных и фактических параметров функции. Формальные параметры функции — это аргументы функции, которые объявляются в заголовке функции при её описании. Фактические параметры — это параметры, которые подставляются вместо формальных непосредственно при вызове функции.

Синтаксис вызова функции:

```
[имя_переменной] = имя_функции([список параметров])
```

**Пример 5.1** (функция без параметров и выходных значений)**.** Функции без параметров и выходных значений обычно используются для вывода на экран какого-либо статичного текста. Например, это может быть печать меню программы:

функция без параметров и выходного значения

```
def print_menu():
    print('Выберите пункт выполнения программы:')
```

```
    print('1: Ввод значения x')
    print('2: Вывод значения x на экран')
    print('3: Возведение x в квадрат')
    print('4: Выход из программы')

print_menu() # вызов функции без параметров и выходных значений
```

**Пример 5.2** (функции с параметрами)**.** Функция поиска максимума из трёх чисел $x, y, z$:

Описание и примеры вызова функции для поиска максимума

```
def max3(x, y, z):
    if x < y:
        max_ = y
    else:
        max_ = x

    if z > max_:
        max_ = z
    return max_  # функция вернёт 1 число -- максимум из трёх чисел

# результат функции рекомендуется присвоить переменной
m = max3(5, -12, 0)
print('Максимум:', m)
```

**Пример 5.3** (функция, возвращающая более одного значения)**.** Функция одновременного поиска максимума и минимума из двух чисел $x, y$:

Два результата у функции

```
def minMax(x, y):
    if x < y:
        return x, y
    else:
        return y, x
```

Если функция возвращает более одного значения, то результат её вызова может быть обработан несколькими способами.

**Пример 5.3.1** (обработка результата функции как кортежа)**.** Результат работы функции присваивается в одну переменную, первый элемент которой является минимумом из двух чисел, а второй — максимумом. Чтобы получить каждое из значений по отдельности необходимо обратиться к элементу с соответствующим индексом.

Обработка результата функции как кортежа

```
m = minMax(16, -4) # m - кортеж

print('min(x, y) =', m[0])
print('max(x, y) =', m[1])
```

Важно. Индексация в кортежах и списках в Python начинается с 0, поэтому минимальному значению будет соответствовать значение m[0], а максимальному — m[1].

**Пример 5.3.2** (обработка результата функции с использованием множественного присваивания)**.** Результат работы функции присваивается сразу в несколько переменных.

Использование множественного присваивания

```
minXY, maxXY = minMax(16, -4)

print('min(x, y) =', minXY)
print('max(x, y) =', maxXY)
```

Важно. Количество идентификаторов слева от знака присваивания должно совпадать с количеством значений, которые возвращает функция, иначе возникнет ошибка!

## 5.3 Способы передачи аргументов

**Пример 5.4** (позиционный способ передачи аргументов)**.** Вычислить расстояние между двумя точками с координатами $(x_1, y_1)$ и $(x_2, y_2)$.

Позиционный способ передачи аргументов

```
from math import sqrt

def distance(x1, y1, x2, y2):
    d = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))
    return d

res = distance(1, 2, -1, -4)

print('d = %5.2f' % res)
```

В примере 5.4 используется позиционный способ передачи аргументов. При таком вызове значения аргументам функции передаются в том порядке, в котором они описаны в заголовке, т. е. x1 = 1, y1 = 2, x2 = -1, y2 = -4.

Важно. Количество фактических параметров при вызове функции должно совпадать с количеством формальных, иначе возникнет ошибка!

**Пример 5.5** (передача значений по ключу)**.** Аргументы в функцию можно передавать по ключу. В этом случае при вызове функции указываются пары

идентификатор = значение

Передача значений по ключу

```
# аргументы в порядке описания параметров в функции
res1 = distance(x1 = 1, y1 = 2, x2 = -1, y2 = -4)
print('d = %5.2f' % res1)

# в произвольном порядке
res2 = distance(x2 = -1, y1 = 2, x1 = 1, y2 = -4)
print('d = %5.2f' % res2)
```

Аргументы функции при вызове могут идти в любом порядке.

**Пример 5.5.1** (комбинированная передача значений)**.** В одном вызове функции можно одновременно использовать оба способа передачи:

Комбинированная передача значений

```
res3 = distance(-1, 2, y2 = -4, x2 = 1)

print('d = %5.2f' % res3)
```

Важно. Позиционные параметры обязательно должны идти перед параметрами, передаваемыми по ключу!

## 5.4 Произвольное количество аргументов

Иногда при описании функции может быть заранее неизвестно количество и тип её аргументов. В этом случае при описании функции в качестве аргументов можно использовать `*args` для передачи разного числа аргументов.

**Пример 5.6** (произвольное количество параметров)**.** Функция вывода на экран всех аргументов функции:

*args

```
def any_print(*args):
    for el in args:
        print(el, end=' ')
    print()

any_print(1, 2, 4, 51)
any_print('Dog', 'Cat', 'World')
any_print(1.5, 'Hello', True)
```

args представляет собой произвольный список параметров, * здесь означает «распаковку» списка значений.

Результат работы программы

```
1 2 4 51
Dog Cat World
1.5 Hello True
```

## 5.5 Значения параметров по умолчанию

При описании функции для её параметров можно указывать значения по умолчанию, которые будут использованы, если на их место не были переданы значения. В этом случае в заголовке функции указываются пары

идентификатор = значение

**Пример 5.7** (значения параметров по умолчанию)**.** Вычислить расстояние между двумя точками с координатами $(x_1, y_1)$ и $(x_2, y_2)$. В случае, если переданы координаты только одной точки — вычислить расстояние от неё до начала координат. Для это необходимо задать координаты точки $(x_2, y_2) = (0, 0)$ как параметры по умолчанию:

Значения параметров по умолчанию

```
from math import sqrt

def distance(x1, y1, x2=0, y2=0):
    d = sqrt((x2 - x1) * (x2 - x1) + (y2 - y1) * (y2 - y1))
    return d

# разные способы вызова функции distance:
D1 = distance(1, 2, -1, -4)    # все параметры переданы по позиции

# переданы x1, y1, x2; y2 значение по умолчанию
D2 = distance(1, 2, 4)

# переданы x1, y1; x2, y2 значения по умолчанию
D3 = distance(1, 2)

D4 = distance(1, y1=-10, x2=3)  # x1 передано по позиции;
                                # y1, x2 переданы по ключу;
                                # y2 значение по умолчанию
```

Если в функции используются и обычные параметры, и параметры со значением по умолчанию, то обычные параметры должны идти перед параметрами со значениями по умолчанию. Позиционные параметры обязательно должны идти перед параметрами, передаваемыми по ключу.

Важно. Значения по умолчанию задаются один раз при определении функции, а не при каждом её вызове! Поэтому не следует использовать в качестве значения по умолчанию случайно генерируемое значение.

## 5.6 Глобальные и локальные переменные

Все переменные, которые определяются внутри функции (в том числе и её формальные параметры) называются локальными. Локальные переменные доступны только внутри функции и только во время её выполнения. При попытке обратиться к локальной переменной из основной программы возникнет ошибка. Для того, чтобы использовать какую-либо переменную в любой части программы, её необходимо объявить как глобальную, используя инструкцию global внутри тела функции.

**Пример 5.8** (глобальные и локальные переменные)**.** Опишем две функции, изменяющие значение переменной $x$ с использованием инструкции `global` и без.

Глобальные и локальные переменные

```
def neg100():
    x = -100 # здесь определяется локальная переменная x

def neg100g():
    global x  # переменная x из основной программы
    x = -100

x = 23
neg100()   # вызов функции без инструкции global
print(x)   # Результат: 23

neg100g()  # вызов функции с инструкцией global
print(x)   # Результат: -100
```

## 5.7 Анонимные функции

Анонимная функция в Python — это функция, которая определяется без имени. Их также называют lambda-функциями. Обычно такие функции не используются

многократно и состоят из одного простого оператора. Описание анонимной функции начинается с ключевого слова lambda.

синтаксис описания lambda-функции

```
имя_функции = lambda [параметры функции] : выражение
```

Как и обычная функция, lambda-функция может не иметь параметров и не возвращать значение. При вызове анонимной функции происходит выполнение выражения, записанного после двоеточия. Синтаксис вызова анонимной функции идентичен вызову обычной функции.

**Пример 5.9** (сравнение синтаксиса описания обычной и lambda-функций)**.** В коде ниже приведены различные описания обычных функций и их «аналоги» в виде lambda-функций:

Описания функций

```
def x3(x):
    return x * x * x

def sumXY(x, y):
    return x + y

def printt(t):
    print('t =', t)

def hello():
    print('Hello! ;)')
```

Описания lambda-функций

```
x3 = lambda x: x * x * x


sumXY = lambda x, y: x + y


printt = lambda t: print('t =', t)


hello = lambda : print('Hello! ;)')
```

## 5.8 Примеры решения задач

**Пример 5.10** (функция поиска $\max(x, y)$)**.** Описание и примеры вызова функции для поиска максимума.

Поиск максимума из двух выражений

```
def maxXY(x, y):
    if x > y:
        return x
    else:
        return y

c = maxXY(13, 4)

print(c)
print(maxXY(abs(-5), c)) # max(13, 4, |-5|)
```

**Пример 5.11** (вычисление площади треугольника)**.** Вычислить площадь треугольника по формуле Герона (параметры $a$, $b$, $c$ — здесь стороны треугольника).

Нахождение площади треугольника по заданным сторонам

```
import math

def geron(a, b, c):
    p = (a + b + c) / 2
    return math.sqrt(p * (p - a) * (p - b) * (p - c))

print('Площадь =', geron(3, 4, 5))
```

**Пример 5.12** (сравнение двух выражений с заданной точностью)**.** Вещественные числа по умолчанию сравниваются с машинной точностью. На практике такая точность бывает нужна редко. Напишем функцию для сравнения двух выражений с заданной точностью. Точность по умолчанию возьмём $\varepsilon = 10^{-6}$.

Сравнение значений двух выражений с заданной точностью

```
def Compare(x, y, eps = 1e-6):
    '''
    Сравнивает значения двух выражений с заданной точностью.

    Входные параметры:
        x   (float) - первое вещественное число
        y   (float) - второе вещественное число
        eps (float) - точность сравнения (по умолчанию 1e-6)
```

```
    Выходной параметр:
        _   (bool) - результат сравнения двух вещественных чисел
    '''

    return abs(x - y) <= eps

print(Compare(1.55, 1.6))       # False
print(Compare(1.55, 1.6, 0.1))  # True
print('-' * 30)
help(Compare) # печать документации функции Compare
```

Функция help(f) принимает в качестве аргумента имя некоторой функции f и выводит на экран её документацию.

Результат работы программы

```
False
True
------------------------------
Help on function Compare in module __main__:

Compare(x, y, eps=1e-06)
    Сравнивает значения двух выражений с заданной точностью.

    Входные параметры:
        x   (float) - первое вещественное число
        y   (float) - второе вещественное число
        eps (float) - точность сравнения (по умолчанию 1e-6)
    Выходной параметр:
        _   (bool) - результат сравнения двух вещественных чисел
```

**Пример 5.13** (sympy; построение графиков)**.** Продемонстрируем простейший способ построения графиков средствами библиотеки sympy.

Дано $x \in \mathbb{R}$. Для функций

$$f_1(x) = \sqrt{x} + 1, \qquad f_2(x) = \sin 2x$$

выполнить действия

| | |
|---|---|
| найти $f_1(x)$ и построить её график на $[0; x]$, | если $x > 0$; |
| найти $f_2(x)$ и построить её график на $[x; -x]$, | если $x < 0$; |
| выдать сообщение Not plot, | если $x = 0$. |

Графики функций

```
from sympy import sqrt, sin, plot, Symbol

f1 = lambda x: sqrt(x) + 1
f2 = lambda x: sin(2*x)

x = float(input('x >>> '))
t = Symbol('t')

if x < 0:
    print('sin(2*x) =', f2(x))
    plot(f2(t), (t, x, -x))
elif x > 0:
    print('sqrt(x) + 1 =', f1(x))
    plot(f1(t), (t, 0, x))
else:
    print('Not plot')
```

Результат работы программы

```
x >>> 0
Not plot
```

Результат работы программы

```
x >>> -10
sin(2*x) = -0.91295
# см. рисунок справа
```

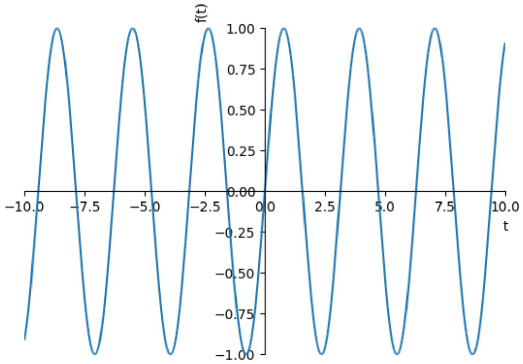


Рис. 4. График $f_2(x)$ на интервале $[-10; 10]$

Результат работы программы

```
x >>> 5
sqrt(x) + 1 = 3.23607
# см. рисунок справа
```

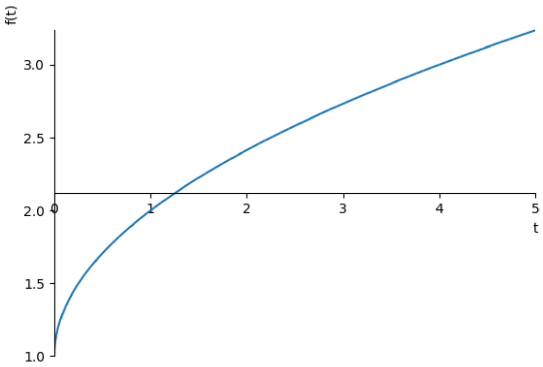


Рис. 5. График $f_1(x)$ на интервале $[0; 5]$