Тема лекции:

Функции в PYTHON

Лектор: Москаленко Елена Валерьевна

Типы функций

Функции в Python можно разделить на два типа:

1. **Встроенные функции** — функции, встроенные в Python.
2. **Пользовательские функции** — функции, созданные пользователями. Чтобы их использовать, нужно устанавливать дополнительные файлы.

Примеры встроенных функций в Python

- **abs(x)**

Возвращает абсолютную величину (модуль числа).

- **bool([x])**

Преобразование к типу bool, использующая стандартную процедуру проверки истинности.

- **chr(i)**

Возвращает символ по его числовому представлению.

- **complex([real[, imag]])**

Преобразование к комплексному числу.

- **float([x])**

Преобразование к числу с плавающей точкой. Если аргумент не указан, возвращается 0.0.

- **input([prompt])**

Считывает и возвращает строку входных данных.

- **int([x=0, [base=10]])**

Преобразует x к целому числу в десятичной системе счисления.

- **list([iterable])**

Создает список.

Примеры встроенных функций в Python

- **max(iterable, *args[, key, default])**

Возвращает элемент с набольшим значением из переданных в функцию.

- **min()**

Возвращает элемент с наименьшим значением из переданных в функцию.

- **pow(x, y[, z])**

Возвращает результат возведения числа в степень, с опциональным делением по модулю.

- **print(*objs, sep=' ', end='\n', file=sys.stdout, flush=False)**

Выводит заданные объекты на экран или отправляет их текстовым потоком в файл.

- **range(start, stop[, step])**

Арифметическая прогрессия от start до stop с шагом step.

- **round(number[, ndigits])**

Возвращает число с плавающей запятой, округлённое до указанного количества цифр после запятой.

Определение понятия «функция»

Функции – это многократно используемые фрагменты программы. Они позволяют дать имя определённому блоку команд с тем, чтобы в последствии запускать этот блок по указанному имени в любом месте программы и сколь угодно много раз. Это называется **вызовом функции**.

Функция — это группа связанных инструкций, выполняющих определенную задачу.

Функции определяются при помощи зарезервированного слова def. После этого слова указывается имя функции, за которым следует пара скобок, в которых можно указать имена некоторых переменных, и двоеточие в конце строки. Далее следует блок команд, составляющих функцию.

Функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.

Синтаксис функции:

```
def имя_функции(параметры):
    """строка документации"""
    операторы
```

Компоненты функции

- Ключевое слово **def** — начало заголовка функции.
- **Имя функции** — для однозначной идентификации функции. Оно соответствует правилам называния идентификаторов в Python.
- С помощью **параметров (аргументов)** мы передаем значения в функцию. Аргументов может и не быть.
- **Двоеточие** : обозначает конец заголовка функции.
- Необязательная **строка документации** (docstring) нужна для описания того, что делает функция.
- Один или несколько операторов Python составляют **тело функции**. Все инструкции должны иметь одинаковый отступ (4 пробела или 1 TAB).
- **Оператор return** возвращает переданное значение из функции. Он необязателен.

Пример функции

Пример функции

```
def greet(name):
    """
    Эта функция
    приветствует человека, имя которого
    хранится в параметре name.
    """
    print("Привет, " + name + ". Доброе утро!")
```

Вызов функции

После того, как мы определили функцию, мы можем вызвать ее в программе или даже из командной строки Python. Чтобы вызвать функцию - вводим ее имя с соответствующими параметрами.

```
>>> greet('Соня')
Привет, Соня. Доброе утро!
```

```
def greet(name):
    """
    Эта функция приветствует человека, имя которого хранится в параметре name.
    """
    print("Привет, " + name + ". Доброе утро!")
greet('Соня')
```

Пример функции

```
def sayHello():
    print('Привет, Мир!') # блок, принадлежащий функции
# Конец функции

sayHello() # вызов функции
sayHello() # ещё один вызов функции
```

Вывод:

```
$ python function1.py
Привет, Мир!
Привет, Мир!
```

Параметры функции

Параметры функции – это некие входные данные, которые мы можем передать функции, чтобы получить соответствующий им результат.

- вызывать одну и ту же функцию можно много раз, а значит нет необходимости писать один и тот же код снова и снова.
- параметры похожи на переменные, за исключением того, что значение этих переменных указывается при вызове функции, и во время работы функции им уже присвоены их значения.
- параметры указываются в скобках при объявлении функции и разделяются запятыми. Аналогично передаём значения, когда вызываем функцию.

Пример функции:

```
def printMax(a, b):
    if a > b:
        print(a, 'максимально')
    elif a == b:
        print(a, 'равно', b)
    else:
        print(b, 'максимально')

printMax(3, 4) # прямая передача значений

x = 5
y = 7

printMax(x, y) # передача переменных в качестве аргументов
```

Вывод:

```
$ python func_param.py
4 максимально
7 максимально
```

Локальные переменные

Локальная переменная создается внутри функции.
Инструкции, которые находятся за пределами функции, к ней доступа не имеют. Разные функции могут иметь локальные переменные с одинаковыми именами, потому что функции не видят локальные переменные друг друга.
Всякий раз, когда переменной внутри функции присваивается значение, в результате создается локальная переменная. Она принадлежит функции, в которой создается, и к такой переменной могут получать доступ только инструкции в этой функции.
Если инструкция в одной функции попытается обратиться к локальной переменной, которая принадлежит другой функции, то произойдет ошибка.

Локальные переменные

```
1 # Определение главной функции.
2 def main():
3     get_name()
4     print('Привет,', name) # Это вызовет ошибку!
5
6 # Определение функции get_name.
7 def get_name():
8     name = input('Введите свое имя: ')
9
10 # Вызвать главную функцию.
11 main()
```

В этой программе есть две функции: `main` и `get_name`. В строке 8 переменной `name` присваивается значение, которое вводится пользователем. Эта инструкция находится в функции `get_name`, поэтому переменная `name` является для этой функции локальной. Это означает, что к переменной `name` не могут обращаться инструкции, находящиеся за пределами функции `get_name`.

Функция `main` вызывает функцию `get_name` в строке 3. Затем инструкция в строке 4 пытается обратиться к переменной `name`. Это приводит к ошибке, потому что переменная `name` локальна для функции `get_name`, и инструкции в функции `main` получить к ней доступ не могут.

Локальные переменные

Область действия переменной - это часть программы, в которой можно обращаться к переменной.

При объявлении переменных внутри определения функции, они никоим образом не связаны с другими переменными с таким же именем за пределами функции – т.е. имена переменных являются **локальными** в функции. Это называется **областью видимости переменной**. Область видимости всех переменных ограничена блоком, в котором они объявлены, начиная с точки объявления имени.

Локальные переменные

```
x = 50

def func(x):
    print('x равен', x)
    x = 2
    print('Замена локального x на', x)

func(x)
print('x по прежнему', x)
```

Вывод:

```
$ python func_local.py
x равен 50
Замена локального x на 2
x по прежнему 50
```

```python
def my_func():
    x = 10
    print("Значение внутри функции:", x)

x = 20
my_func()
print("Значение вне функции:", x)
```

Вывод:

```
Значение внутри функции: 10
Значение вне функции: 20
```

Поскольку локальные переменные функции скрыты от других функций, другие функции могут иметь собственные локальные переменные с одинаковым именем.

```
1 # Эта программ демонстрирует две функции, которые
2 # имеют локальные переменные с одинаковыми именами.
3
4 def main():
5     # Вызвать функцию texas.
6     texas()
7     # Вызвать функцию california.
8     california()
9
10 # Определение функции texas. Она создает
11 # локальную переменную с именем birds.
12 def texas():
13     birds = 5000
14     print('В Техасе обитает', birds, 'птиц.')
15
16 # Определение функции california. Она тоже
17 # создает локальную переменную с именем birds.
18 def california():
19     birds = 8000
20     print('В Калифорнии обитает', birds, 'птиц.')
21
22 # Вызвать главную функцию.
23 main()
```

Вывод программы

```
В Техасе обитает 5000 птиц.
В Калифорнии обитает 8000 птиц.
```

Зарезервированное слово «global»

- Чтобы присвоить некоторое значение переменной, определённой на высшем уровне программы (т.е. не в какой-либо области видимости, как то функции или классы), необходимо указать Python, что её имя не локально, а глобально (global). Сделаем это при помощи зарезервированного слова global.
- Без применения зарезервированного слова global невозможно присвоить значение переменной, определённой за пределами функции.
- **Использование зарезервированного слова global** достаточно ясно **показывает, что переменная объявлена в самом внешнем блоке**.

Зарезервированное слово «global»

Зарезервированное слово global используется для того, чтобы объявить, что x – это глобальная переменная, а значит, когда мы присваиваем значение имени x внутри функции, это изменение отразится на значении переменной x в основном блоке программы.

Зарезервированное слово «global»

```
x = 50

def func():
    global x

    print('x равно', x)
    x = 2
    print('Заменяем глобальное значение x на', x)

func()
print('Значение x составляет', x)
```

Вывод:

```
$ python func_global.py
x равно 50
Заменяем глобальное значение x на 2
Знчение x составляет 2
```

```
 1 # Создать глобальную переменную.
 2 my_value = 10
 3
 4 # Функция show_value печатает
 5 # значение глобальной переменной.
 6 def show_value():
 7     print(my_value)
 8
 9 # Вызвать функцию show_value.
10 show_value()
```

Вывод программы

```
10
```

Инструкция присваивания в строке 2 создает переменную с именем `my_value`. Поскольку эта инструкция находится за пределами любой функции, она является глобальной. Во время исполнения функции `show_value` инструкция в строке 7 напечатает значение, на которое ссылается переменная `my_value`.

Глобальные переменные

Если нужно, чтобы инструкция внутри функции присваивала значение глобальной переменной, то требуется дополнительный шаг. В этом случае, глобальная переменная должна быть объявлена внутри функции:

```
1 # Создать глобальную переменную.
2 number = 0
3
4 def main():
5     global number
6     number = int(input('Введите число: '))
7     show_number()
8
9 def show_number():
10    print('Вы ввели число', number)
11
12 # Вызвать главную функцию.
13 main()
```

Вывод программы (вводимые данные выделены жирным шрифтом)

```
Введите число: 55 Enter
Вы ввели число 55
```

Оператор «return»

Оператор **return** используется для возврата из функции, т. е. для прекращения её работы и выхода из неё. При этом можно также вернуть некоторое значение из функции.

```
def maximum(x, y):
    if x > y:
        return x
    elif x == y:
        return 'Числа равны.'
    else:
        return y

print(maximum(2, 3))
```

Вывод:

```
$ python func_return.py
3
```

Пример возвращения значений

```
def absolute_value(num):
    """ Возвращает абсолютное значение
   введенного числа"""

    if num >= 0:
        return num
    else:
        return -num

print(absolute_value(2))

print(absolute_value(-4))
```

Вывод:

```
2
4
```

Если в операторе нет выражения или самого оператора возврата нет внутри функции, функция вернет объект None.

Функцию можно записать в одну строку, если блок инструкций представляет собой простое выражение:

```
def sum(a, b): return a + b
```

Функции могут быть вложенными:

```
def func1(a, b):

    def inner_func(x):
        return x*x*x

    return inner_func(a) + inner_func(b)
```

Функции — это объекты, поэтому их можно присваивать переменным.

Передача аргументов в функцию

Аргумент - это любая порция данных, которая передается в функцию, когда функция вызывается. **Параметр**- это переменная, которая получает аргумент, переданный в функцию. Порции данных, которые отправляются в функцию, называются аргументами. Функция может использовать свои аргументы в вычислениях или других операциях.

Если требуется, чтобы функция получала аргументы, когда она вызывается, то необходимо оборудовать эту функцию одной или несколькими параметрическими переменными. Параметрическая переменная, часто именуемая просто параметром, - это специальная переменная, которой присваивается значение аргумента, когда функция вызывается.

П
```
def show_double(number):
    result = number * 2
    print(result)
```
еской переменной:

Передача аргументов в функцию

```
1 # Это программа демонстрирует аргумент,
2 # передаваемый в функцию.
3
4 def main():
5     value = 5
6     show_double(value)
7
8 # Функция show_double принимает аргумент
9 # и показывает его удвоенное значение.
10 def show_double(number):
11     result = number * 2
12     print(result)
13
14 # Вызвать главную функцию.
15 main()
```

Вывод программы

```
10
```

Рекурсивные функции

Рекурсия — это тот случай, когда функция вызывает сама себя.
Самый известный пример — вычисление факториала n! = n * n — 1 * n -2 * … 2 *1.
Зная, что 0! = 1, факториал можно записать следующим образом:

```
def factorial(n):
    if n != 0:
        return n * factorial(n-1)
    else:
        return 1
```

Передача многочисленных аргументов

Часто имеет смысл писать функции, которые могут принимать многочисленные аргументы.

В примере показана функция show_sum, которая принимает два аргумента, складывает их и показывает сумму.

```
 1 # Эта программа демонстрирует функцию, которая принимает
 2 # два аргумента.
 3
 4 def main():
 5     print('Сумма чисел 12 и 45 равняется')
 6     show_sum(12, 45)
 7
 8 # Функция show_sum принимает два аргумента
 9 # и показывает их сумму.
10 def show_sum(num1, num2):
11     result = num1 + num2
12     print(result)
13
14 # Вызвать главную функцию.
15 main()
```

Вывод программы

```
Сумма чисел 12 и 45 равняется
57
```

```
1 # Эта программа демонстрирует передачу в функцию двух
2 # строковых аргументов.
3
4 def main():
5     first_name = input('Введите свое имя: ')
6     last_name = input('Введите свою фамилию: ')
7     print('Ваше имя в обратном порядке')
8     reverse_name(first_name, last_name)
9
10 def reverse_name(first, last):
11     print(last, first)
12
13 # Вызвать главную функцию.
14 main()
```

Вывод программы (вводимые данные выделены жирным шрифтом)

```
Введите свое имя: Мэт Enter
Введите свою фамилию: Хойл Enter
Ваше имя в обратном порядке
Хойл Мэт
```