???????

> **Функция** — это группа связанных инструкций, выполняющих определенную задачу.

Функции помогают разбить нашу программу на более мелкие части. По мере того, как наша программа становится все больше и больше, функции делают ее более организованной и управляемой.

Кроме того, функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода.

????????? ???????

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

Давайте рассмотрим какие компоненты включает в себя определение функции:

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

?????? ???????

```
def greet(name):
    """
    Эта функция
    приветствует человека, имя которого
```

```
    хранится в параметре name.
    """
    print("Привет, " + name + ". Доброе утро!")
```

??? ??????? ???????

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

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

> **Примечание**. Попробуйте сами запустить приведенный выше код с определением функции и посмотрите результат.

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

greet('Джон')
```

?????? ???????????

Первая строка после заголовка функции называется строкой документации, она описывает, что делает функция.

Документирование кода — не обязательная, но очень хорошая практика. Если вы не помните, что ели на ужин на прошлой неделе, всегда документируйте свой код.

В приведенном выше примере у нас есть строка документации сразу под заголовком функции. Обычно используют тройные кавычки, чтобы документация могла занимать несколько строк. Получить доступ к строке документации можно через атрибут `__doc__`.

?????? ?????? ???????????

Попробуйте запустить в оболочке Python следующую команду и посмотрите на результат.

```
>>> print(greet.__doc__)

    Эта функция
    приветствует человека, имя которого
    хранится в параметре name.
```

????????????? ????????

Оператор `return` используется для выхода из функции и возврата в то место, откуда она была вызвана.

????????? ??????????? ????????

```
return список_выражений
```

Оператор `return` может содержать выражение, которое возвращает значение. Если в операторе нет выражения или самого оператора возврата нет внутри функции, функция вернет объект `None`.

?????? ??????????? ????????

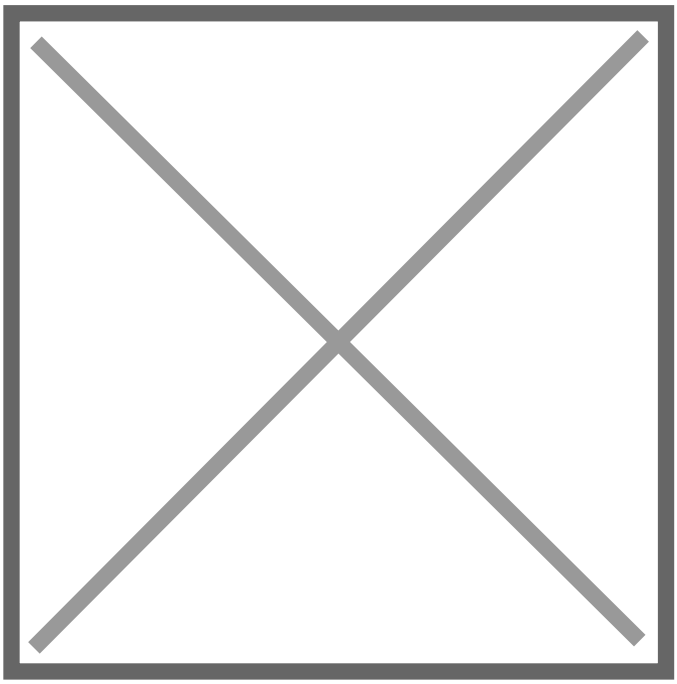
```
def absolute_value(num):
    """ Возвращает абсолютное значение
   введенного числа"""

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

print(absolute_value(2))

print(absolute_value(-4))
```

Вывод:

```
2
4
```

??? ???????? ???????

Рассмотрите схему. Так работают функции.



??????? ????????? ? ????? ????? ??????????

Область видимости переменной — это часть программы, в которой к данной переменной можно обращаться. Параметры и переменные, определенные в теле функции, доступны только внутри нее. Это значит, что они находятся в локальной области видимости.

Время жизни переменной — это период, в течение которого переменная находится в памяти. Время жизни переменной внутри функции длится до тех пор, пока функция выполняется. Переменные «уничтожаются», когда мы выходим из функции. Это значит, что функция не помнит значения переменных из предыдущих вызовов.

Вот пример, иллюстрирующий область видимости переменной внутри функции.

```
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
```

Здесь мы видим, что значение `x` изначально равно 20. Хотя функция `my_func()` изменила значение `x` на 10, это не повлияло на ее значение вне функции.

Так происходит потому, что локальная переменная `x` внутри функции отличается от глобальной переменной `x`, которая находится вне функции. Хотя у них одинаковые имена, это две разные переменные с разной областью видимости.

А вот переменные, которые объявлены вне функции, будут видны внутри нее. У них глобальная область видимости.

Если переменная определена вне функции, то внутри функции мы можем прочитать ее значение. Однако мы не сможем его изменить. Для того, чтобы изменить ее значение, нужно объявить ее как глобальную с помощью ключевого слова `global`.

???? ???????

Функции в Python можно разделить на два типа:

1. **Встроенные функции** — функции, предоставляемые языком программирования.
2. **Пользовательские функции** — функции, описанные пользователем в программном коде.

???????? ?? ????????? ???????

Python допускает значения по умолчанию для параметров функции. Если вызывающий абонент не передает параметр, используется значение по умолчанию.

```
def hello(year=2019):
  print(f'Hello World {year}')
hello(2020) # function parameter is passed
hello() # function parameter is not passed, so default value will be used
```

Вывод:

```
Hello World 2020
Hello World 2019
```

?????????? ??????????? ???????? ?????? ???????

Функция может иметь несколько операторов возврата. Однако при достижении одного из операторов возврата выполнение функции прекращается, и значение возвращается вызывающей стороне.

```
def odd_even_checker(i):
  if i % 2 == 0:
    return 'even'
  else:
    return 'odd'

print(odd_even_checker(20))
print(odd_even_checker(15))
```

??????? ??????????? ????????

Функция Python может возвращать несколько значений одно за другим. Это реализовано с использованием ключевого слова `yield`. Это полезно, когда вы хотите, чтобы функция возвращала большое количество значений и обрабатывала их. Мы можем разделить возвращаемые значения на несколько частей, используя оператор `yield`. Этот тип функции также называется функцией генератора.

```
def return_odd_ints(i):
  x = 1
  while x <= i:
    yield x
    x += 2

output = return_odd_ints(10)
```

```
for out in output:
  print(out)
```

Вывод:

```
1
3
5
7
9
```

?????????

Python допускает три типа параметров в определении функции:

- Формальные аргументы: те, которые мы видели в примерах до сих пор.
- Переменное количество аргументов без ключевых слов: например, def add(*args)
- Переменное количество аргументов ключевых слов или именованных аргументов: например, def add(**kwargs)

Некоторые важные моменты относительно переменных аргументов в Python:

- Порядок аргументов должен быть формальным, * args и ** kwargs.
- Не обязательно использовать имена параметров переменных как args и kwargs. Однако лучше всего использовать их для лучшей читаемости кода.
- Тип args — кортеж. Таким образом, мы можем передать кортеж для отображения с переменной * args.
- Тип kwargs — словарь. Таким образом, мы можем передать словарь для сопоставления с переменной ** kwargs.

```
def add(x, y, *args, **kwargs):
  sum = x + y
  for a in args:
    sum += a

  for k, v in kwargs.items():
    sum += v
  return sum

total = add(1, 2, *(3, 4), **{"k1": 5, "k2": 6})
```

```
print(total) # 21
```

??????????? ???????

Когда функция вызывает сама себя, она называется рекурсивной функцией. В программировании этот сценарий называется рекурсией.

Вы должны быть очень осторожны при использовании рекурсии, потому что есть вероятность, что функция никогда не завершится и перейдет в бесконечный цикл. Вот простой пример печати ряда Фибоначчи с использованием рекурсии.

```
def fibonacci_numbers_at_index(count):
  if count <= 1:
    return count
  else:
    return fibonacci_numbers_at_index(count - 1) + fibonacci_numbers_at_index(count - 2)


count = 5
i = 1
while i <= count:
  print(fibonacci_numbers_at_index(i))
  i += 1
```

Полезно знать о рекурсии, но в большинстве случаев при программировании это не нужно. То же самое можно сделать с помощью цикла `for` или `while`.

???? ?????? ???????

Функции Python являются экземплярами класса «функция». Мы можем проверить это с помощью функции `type()`.

```
def foo():
  pass


print(type(foo)) # <class 'function'>
```

????????? ??????? ? ???????

- Функция Python является частью файла сценария Python, в котором она определена, тогда как методы определены внутри определения класса.
- Мы можем вызвать функцию напрямую, если она находится в том же модуле. Если функция определена в другом модуле, мы можем импортировать модуль, а затем вызвать функцию напрямую. Нам нужен класс или объект класса для вызова методов.
- Функция Python может обращаться ко всем глобальным переменным, тогда как методы класса Python могут обращаться к глобальным переменным, а также к атрибутам и функциям класса.
- Тип данных функций Python — это «функция», а тип данных методов Python — «метод».

```
class Data:
  def foo(self):
    print('foo method')


  def foo():
    print('foo function')

# calling a function
foo()

# calling a method
d = Data()
d.foo()

# checking data types
print(type(foo)) # <class 'function'>
print(type(d.foo)) # <class 'method'>
```

Преимущества:

- Возможность повторного использования кода, потому что мы можем вызывать одну и ту же функцию несколько раз.
- Модульный код, поскольку мы можем определять разные функции для разных задач.
- Улучшает ремонтопригодность кода.
- Абстракция, поскольку вызывающему абоненту не нужно знать реализацию функции.

????????? ???????

Анонимные функции не имеют имени. Мы можем определить анонимную функцию в Python, используя ключевое слово `lambda`.

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

f_square = lambda x: x * x

print(square(10)) # 100
print(f_square(10)) # 100
```

?????????

1. Функции в Python
2. Функции Python
3. Функции
4. Возвращаемый тип функции

Revision #8
Created 2023-05-30 07:35:22 UTC by Антон Сергеевич Абраменко
Updated 2023-07-06 04:08:58 UTC by Антон Сергеевич Абраменко