ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

Урок 6: Функции

Уровень: начинающий
Курс: введение Python

Содержание

- Добро пожаловать на этот урок по функциям! Вы узнаете о:
- Определяющие функции
- Область видимости переменной
- Документация
- Лямбда-выражения
- Итераторы и генераторы

Функции

- Функции - это полезные фрагменты программного кода, которые позволяют вам инкапсулировать задачу
- Инкапсуляция это способ выполнить целый ряд шагов с помощью одной простой команды

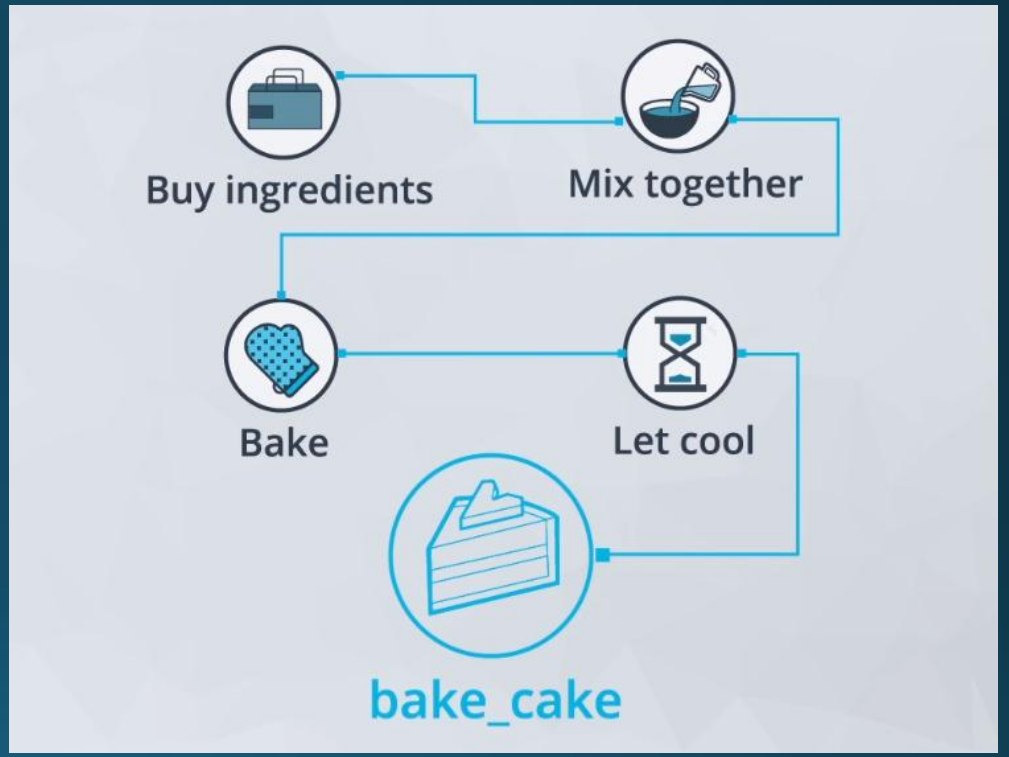
Пример из жизни

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Например, представьте, что вы хотите испечь торт
- Вам нужно будет:
  - купить ингредиенты;
  - смешать их;
  - положить все это в духовку;
  - а затем охладить.

Пример из жизни

- В компьютерном программировании функции инкапсулируют все этапы процесса в одну команду;
- В этом случае мы бы могли просто использовать функции выпекания торта и бросить все эти направления в одну функцию;
- Теперь, когда вы хотите испечь торт, мы просто используем эту функцию, не беспокоясь о деталях;
- Функции также используются, чтобы помочь организовать и оптимизировать код.



Определяющие функции

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- ExПример определения функции:

```
def cylinder_volume(height, radius):
    pi = 3.14159
    return height * pi * radius ** 2
```

- Пример определения функции цилиндр _ объем, мы можем **вызвать** функцию таким образом.

```
cylinder_volume(10, 3)
```

- Это называется оператором **вызова функции**.

Заголовок функции

Давайте начнем с заголовка функции, который является первой строкой определения функции

1. Заголовок функции всегда начинается с ключевого слова **def**, которое указывает, что это **определение функции**.
2. Затем следует **название функции** (здесь, **цилиндр _ объем**), которое следует тем же соглашениям об именовании, что и переменные.
3. Сразу после имени ставятся *круглые скобки*, которые могут включать **аргументы**, разделенные запятыми (здесь **высота** и **радиус**).

- Аргументы или **параметры** это значения, которые передаются в качестве **входных данных** при вызове функции и используются в теле функции. Если функция не принимает аргументы, эти скобки оставляют пустыми.

4. Заголовок всегда заканчивается двоеточием**:** .

Тело функции

- Остальная часть функции содержится в теле, где функция выполняет свою работу.
- **Тело** функции - это код с отступом после строки заголовка. Здесь две строки, которые определяют **пи** и возвращают **объем**.
- В этом теле мы можем ссылаться на **переменные аргумента** и определять новые переменные, которые могут использоваться только внутри этих строк с отступом.
- Тело часто включает в себя оператора **return**, который используется для возврата **выходного значения** из функции оператору, который вызвал функцию.
  - Оператор **return** состоит из ключевого слова **return**, за которым следует выражение, которое вычисляется для получения выходного значения для функции. Если нет оператора **return**, функция просто возвращает None.

Соглашение об именовании для функций

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

Названия функций следуют тем же соглашениям об именовании, что и переменные.

1. Используйте только обычные буквы, цифры и подчеркивание в названиях ваших функций. В них не может быть пробелов, и они должны начинаться с буквы или подчеркивания.
2. **Вы не можете использовать зарезервированные слова или встроенные идентификаторы,** которые выполняют важную функцию в Python, о которых вы узнаете в ходе этого курса.
3. Попробуйте использовать описательные имена, которые помогут читателям понять предназначение функции.

Тест

- Какие из перечисленных ниже приемлемых способов начать функцию в Python? (Выбрать все, что подходит.)
  - ❑def my_fuction(arg1, arg2):
  - ❑def do_stuff(arg1 arg2):
  - ❑def my function(arg1, arg2)
  - ❑def my_function(arg2, arg1, arg4):

Print vs. Return в функциях

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- Вот две действительные функции.
- Один возвращает значение, а другой просто печатает значение, ничего не возвращая.
- Протестируйте этот код и экспериментируйте, чтобы понять разницу.

```
1  # this prints something, but does not return anything
2  def show_plus_ten(num):
3      print(num + 10)
4
5  # this returns something
6  def add_ten(num):
7      return(num + 10)
8
9  print('Calling show_plus_ten...')
10 return_value_1 = show_plus_ten(5)
11 print('Done calling')
12 print('This function returned: {}'.format(return_value_1))
13
14 print('\nCalling add_ten...')
15 return_value_2 = add_ten(5)
16 print('Done calling')
17 print('This function returned: {}'.format(return_value_2))
```

Аргументы по умолчанию

- Мы можем добавить аргументы по умолчанию в функцию, чтобы получить значения по умолчанию для параметров, которые не указаны в вызове функции.

```
def cylinder_volume(height, radius=5):
    pi = 3.14159
    return height * pi * radius ** 2
```

- В приведенном выше примере **радиус** установлен на 5, если этот параметр пропустили в вызове функции.
- Если мы вызовем **цилиндр_объем (10)**, функция будет использовать 10 в качестве высоты и 5 в качестве радиуса.
- Однако, если мы вызовем **цилиндр _объем (10,7)**, 7 просто перезапишет значение по умолчанию 5.

Аргументы по умолчанию

- Также обратите внимание, что здесь мы передаем значения нашим аргументам по позиции.
- Передавать значения можно двумя способами - **по позиции и по имени**.
- Каждый из этих вызовов функций оценивается одинаково.

```
cylinder_volume(10, 7)  # pass in arguments by position
cylinder_volume(height=10, radius=7)  # pass in arguments by name
```

Тест

1. Даны четыре действительных числа: x1, y1, x2, y2. Напишите функцию distance(x1, y1, x2, y2), вычисляющая расстояние между точкой (x1,y1) и (x2,y2). Считайте четыре действительных числа и выведите результат работы этой функции.Если вы не знаете, как решить эту задачу, то вы, возможно, не изучали в школе теорему Пифагора. Попробуйте прочитать о ней на Википедии.
2. Дано действительное положительное число a и целое число n. Вычислите a^n. Решение оформите в виде функции power(a, n).Стандартной функцией возведения в степень пользоваться нельзя. Учитывайте n может быть отрицательным числом

Область видимости переменной

ELABORATION.ASIA
INTERNATIONAL EDUCATION CENTRE

- **Область видимости переменной** относится к тому, к какой части программы переменная может ссылаться или использоваться.
- Важно учитывать область видимости при использовании переменных в функциях.
- Если переменная создается внутри функции, она может использоваться только внутри этой функции. Доступ к ней за пределами этой функции невозможен.

```
# This will result in an error
def some_function():
    word = "hello"

print(word)
```

Область видимости переменной

- В приведенном ниже примере **слово** имеет область видимости, которая является **локальной** только для каждой функции.
- Это означает, что вы можете использовать одно и то же имя для разных переменных, которые используются в разных функциях.

```
# This works fine
def some_function():
    word = "hello"

def another_function():
    word = "goodbye"
```

Область видимости переменной

- Переменные, определенные за пределами функций, как в примере ниже, все еще могут быть доступны внутри функции.
- Здесь **слово** имеет **глобальную область видимости.**

```
# This works fine
word = "hello"

def some_function():
    print(word)

some_function()
```

Область видимости переменной

- Обратите внимание, что мы все еще можем получить доступ к значению **слова** с глобальной областью видимости внутри этой функции.
- Однако значение переменной с глобальной областью видимости нельзя **изменить** внутри функции.
- Если вы хотите изменить значение этой переменной внутри функции, она должна быть передана в качестве аргумента.

Задание

- Создайте строковую переменную и создайте функцию, не получающее переменную в качестве аргумента, меняющее ее значение и выведите значение переменной внутри и вне функции. Посмотрите на результат

Документация

- Документация используется для облегчения понимания и использования вашего кода.
- Функции особенно удобочитаемы, потому что они часто используют строки документации или docstrings.
- Строки документации - это тип комментария, используемый для объяснения назначения функции и способа ее использования.
- Вот функция плотности населения со строкой документации.

Документация


```
def population_density(population, land_area):
    """Calculate the population density of an area. """
    return population / land_area
```

- Строки документации заключены в тройные кавычки.
- Первая строка документации представляет собой краткое объяснение назначения функции.
- Если вы полагаете, что это достаточная документация, вы можете закончить строку документации здесь;
- Однострочные строки документации вполне приемлемы, как в примере выше.

Документация


```
def population_density(population, land_area):
    """Calculate the population density of an area.

    INPUT:
    population: int. The population of that area
    land_area: int or float. This function is unit-agnostic, if you pass in va
    of square km or square miles the function will return a density in those u

    OUTPUT:
    population_density: population / land_area. The population density of a pa
    """
    return population / land_area
```

- Если вы считаете, что для данной функции подходит более подробное описание, вы можете добавить больше информации после однострочного описания.
- В приведенном выше примере вы можете увидеть, что мы написали объяснение аргументов функции, указав назначение и тип каждого из них.
- Также принято предоставить некоторое описание выхода функции..

Задание

- Напишите программу возвращающую квадрат числа, и напишите на нее краткую документацию

Lambda функции

- Вы можете использовать лямбда-выражения для создания анонимных функций. То есть, функции без названия.
- Они целесообразны для создания быстрых функций, которые не будут нужны впоследствии в вашем коде.
- Это может быть особенно полезно для функций высокого порядка или функций, которые принимают другие функции в качестве аргументов.
- С лямбда-выражением эта функция:

```
def multiply(x, y):
    return x * y
```

Может сводиться к:


```
multiply = lambda x, y: x * y
```

- Обе эти функции используются одинаково. В любом случае, мы можем вызвать умножение следующим образом:

```
multiply(4, 7)
```

- Это возвращает 28.

Компоненты лямбда-функции

1. Ключевое слово **lambda** используется, чтобы указать, что это - лямбда-выражение.
2. После **лямбды следует** один или несколько аргументов для анонимной функции, разделенных запятыми, за которыми следует двоеточие: . Подобно функциям, именование аргументов в лямбда-выражении произвольное.
3. Last - это выражение, которое оценивается и возвращается в этой функции. Это очень похоже на выражение, которое вы могли видеть, как оператор возврата в функции.

- При такой структуре лямбда-выражения не являются идеальными для сложных функций, но могут быть очень полезными для коротких, простых функций.

Задание

- Напишите лямбда функции сложения, вычитания, умножения и деления