

Функции и объекты в Python

Игорь Рязанцев

Лекция 02

2021г.

Тестовое задание [Лекция 01]

Необходимо вывести наименование светильника с наибольшим световым потоком и наименьшей потребляемой мощностью

```
leds = [( 'LED1' , 40 , 6000) ,  
        ( 'LED2' , 60 , 9000) ,  
        ( 'LED3' , 90 , 12000) ,  
        ( 'LED4' , 80 , 12000) ,]
```

Тестовое задание [Лекция 01]

```
leds = [( 'LED1' , 40 , 6000) ,  
        ( 'LED2' , 60 , 9000) ,  
        ( 'LED3' , 90 , 12000) ,  
        ( 'LED4' , 80 , 12000) ,]  
  
led_max = None  
for led in leds :  
    if led_max == None :  
        led_max = led  
    elif led[2] > led_max[2] :  
        if led[1] < led_max[2] :  
            led_max = led  
print(led_max[0])
```

1 Функции

- Определение функции
- Передача информации в функцию
- Именованные аргументы
- Значения по умолчанию
- Возвращаемое значение

2 Импорт модулей

3 Объектно-ориентированное программирование

- Создание класса
- Создание экземпляра класса
- Обращение к методам
- Обращение к атрибутам
- Наследование

Преимущества использования функций:

- исключение дублирования кода;
- многократное выполнение ранее написанного кода;
- улучшение читаемости кода.

Определение функции

Пример без функций:

```
print("Hello World!")  
print("Hello World!")
```

Результат:
Hello Word!
Hello Word!

Определение функции

Функция – именованный блок кода, предназначенный для решения одной конкретной задачи.

Пример с использованием функций:

```
def greet_user():  
    print("Hello World!")
```

```
greet_user()  
greet_user()
```

Результат:
Hello Word!
Hello Word!

Передача информации в функцию

```
def greet_user(username):  
    print('Hello, {}!'.format(username))  
  
greet_user('Helen')
```

Переменная `username` в определении `greet_user()` – *параметр*, то есть условные данные, необходимые функции для выполнения ее работы. Значение `Helen` в `greet_user(Helen)` – *аргумент*, то есть конкретная информация, переданная при вызове функции.

Передача информации в функцию

Важен порядок передачи аргументов в функцию.
Он должен совпадать с соответствующими принимающими параметрами.

```
def greet_user(username, ages):  
    print( '{0} is {1}! '.format(username, ages))  
  
greet_user(21, 'Helen')
```

Результат:
21 is Helen!

Именованные аргументы

Именованный аргумент представляет собой пару «имя-значение», передаваемую функции. Имя и значение связываются с аргументом напрямую.

```
def greet_user(username, ages):  
    print(' {0} is {1}! '.format(username, ages))  
  
greet_user(ages=21, username='Helen')
```

Результат:
Helen is 21!

Значения по умолчанию

Если при вызове функции передается аргумент, соответствующий данному параметру, Python использует значение аргумента, а если нет — использует значение по умолчанию.

```
def greet_user(username, ages=21):  
    print(' {0} is {1}! '.format(username, ages))  
  
greet_user(username='Helen ')
```

Результат:
Helen is 21!

Возвращаемое значение

return – оператор возврата из функции с возможностью передачи значения.

```
def square(length , width):  
    return length * width
```

```
print(square(10 , 20))
```

Результат:

200

Возвращаемое значение

Функция может вернуть любое значение, в том числе и более сложную структуру данных (например, список).

```
def get_optimal_led(leds):  
    led_max = None  
    for led in leds:  
        if led_max == None:  
            led_max = led  
        elif led[2] > led_max[2]:  
            if led[1] < led_max[2]:  
                led_max = led  
    return led_max  
  
leds = [('LED1', 40, 6000), ('LED2', 60, 9000),]  
print(get_optimal_led(leds=leds)[0])
```

Тестовое задание

Необходимо написать функцию вычисления факториала

Факториалом числа **N** называется произведение всех чисел от единицы до N.

Например, $5! = 1*2*3*4*5$

Факториал нуля равен единице, как и факториал самой единицы:
 $0! = 1$ $1! = 1$

Это нужно запомнить, и учесть в своей программе!

Тестовое задание

```
def calc_factorial(num):  
    factorial = 1  
    if num >= 0:  
        for i in range(1, num + 1):  
            factorial = i * factorial  
        return factorial  
  
factorial = calc_factorial(5)  
print(factorial)
```

Результат:

120

Вариант 1

```
import math  
  
number = 5  
print(math.factorial(number))
```

Результат:
120

Вариант 2

```
from math import factorial
```

```
number = 5
```

```
print(factorial(number))
```

Результат:

120

Вариант 3

```
from math import factorial as my_factorial

number = 5
print(my_factorial(number))
```

Результат:
120

1 Функции

- Определение функции
- Передача информации в функцию
- Именованные аргументы
- Значения по умолчанию
- Возвращаемое значение

2 Импорт модулей

3 Объектно-ориентированное программирование

- Создание класса
- Создание экземпляра класса
- Обращение к методам
- Обращение к атрибутам
- Наследование

Создание класса

Класс определяет общее поведение для целой категории объектов. **Класс – это спецификация.**

```
class LED:
    def __init__(self, power, flux):
        self.power = power
        self.flux = flux
        self.switch_on = False

    def switch(self):
        self.switch_on = not self.switch_on
        if self.switch_on:
            print('LED is switch on!')
        else:
            print('LED is switch off!')
```

Создание экземпляра класса

Создание объекта на основе класса называется **созданием экземпляра класса**

```
class LED:
    def __init__(self, power, flux):
        self.power = power
        self.flux = flux
        self.switch_on = False
    ...
    def switch(self):
        ...

led = LED(60, 2000)
led.switch()
```

Обращение к методам

- Функция, являющаяся частью класса, называется *методом*.
- Параметр *self* обязателен в определении метода; он должен предшествовать всем остальным параметрам.
- Поведение методов аналогично функциям. Отличие состоит в вызове метода («точечная» запись):

`экземпляр_класса.метод()`

```
led = LED(60, 2000)
led.switch()
```

Обращение к атрибутам

Для обращения к атрибутам экземпляра используется «точечная» запись.

```
led = LED(60, 2000)
led.switch()
```

```
if led.switch_on:
    print('LED is switch on!')
else:
    print('LED is switch off!')
```

Результат:

LED is switch on!

LED is switch on!

Назначение атрибуту значения по умолчанию

Каждый атрибут класса должен иметь исходное значение, даже если оно равно 0 или пустой строке.

```
class LED:
    def __init__(self, power, flux):
        ...
        self.flux = flux
        self.switch_on = False
```


Наследование

```
class LED:
    def __init__(self, power, flux):
        ...

class LED_ext(LED):
    def __init__(self, power, flux, type_lid):
        super().__init__(power, flux)
        self.type_lid = type_lid

    def print_type_lid(self):
        print('type_lid={0}'.format(self.type_lid))

led = LED_ext(60, 2000, 1)
led.print_type_lid()
```

Результат:

type_lid=1

Правила наследования

- Один класс, наследующий от другого, автоматически получает все атрибуты и методы первого класса, а также может определять собственные атрибуты и методы.
- Исходный класс называется **родителем**, а новый класс — **потомком**.
- Для потомка необходимо вызывать метод `__init__()` класса-родителя. При этом инициализированные атрибуты родителя становятся доступными для класса-потомка.
- Функция **`super()`** – специальная функция, которая позволяет вызвать метод родительского класса.
- Любой метод родительского класса можно переопределить. Для этого в классе-потомке определяется метод с тем же именем, что и у метода класса-родителя.

Тестовое задание

Необходимо описать класс осветительной установки, создать список объектов и вывести на экран спецификацию объекта освещения.



[1] Презентация [Лекции 01-04]

https://github.com/IRyazantsev/mpei_python_mini-course_2021/tree/main/bin

[2] Python 3. Самое необходимое | Дронов В.А., Прохоренок Н.А.

[3] Изучаем Python. Том 1, 2 | Лутц Марк

[4] Python 3 и PyQt 5. Разработка приложений | Прохоренок Н.А., Дронов В.А.

[5] Django 3.0. Практика создания веб-сайтов на Python | Дронов В. А.

[6] Разработка веб-приложений с использованием Flask | Гринберг Мигель

Вопросы

