



ЦЕНТР  
ДОПОЛНИТЕЛЬНОГО  
ОБРАЗОВАНИЯ  
МГТУ им. Н.Э. Баумана

# Программирование на языке Python. Уровень 1.

Элементы функционального программирования.

## Концепции функциональных языков

- В Python используются некоторые концепции из функциональных языков:
  - ✓ функции являются объектами первого класса;
  - ✓ язык поддерживает функции высших порядков.
- Объектами первого класса называются элементы, которые можно передавать как параметр, возвращать из функции и присваивать переменной.
- Функции высших порядков — это функции, которые могут принимать в качестве аргументов и возвращать другие функции.
- Объекты — базовая сущность в Python.
- Функция — объект!

## ФУНКЦИЯ КАК ОБЪЕКТ

- Имя функции в языке Python представляет собой обычновенную переменную, которая — согласно общему правилу — содержит ссылку на функцию.
- С этой переменной можно обращаться так же, как и с любой другой.

```
def func1(a,b,c): # Создание функции
    # команды, составляющие функцию
func2 = func1 # func2 содержит ссылку
    # на ту же функцию, что и func1
func1(1,2,3)
func2(1,2,3)
```

- ❖ Функцию можно также передавать в качестве параметра в другую функцию и возвращать из функции.
- ❖ Кроме того, в Python функция является специальным объектом, имеющим метод `__call__()`.

# Вложенные функции

- Функция, определенная внутри другой функции, называется **вложенной функцией**.
- Вложенные функции могут получать доступ к переменным из локальной области видимости объемлющих функций.
- В Python нелокальные переменные по умолчанию доступны только для чтения.
- Если необходимо модифицировать нелокальные переменные, то мы должны объявить их явно как нелокальные (используя ключевое слово `nonlocal`).
- Пример обращения к нелокальной переменной:

```
def print_msg(msg):  
    # объемлющая функция  
    def printer():  
        # вложенная функция  
        print(msg)  
    printer()  
print_msg("Hello!")
```

## Замыкание (closure)

- Метод, с помощью которого данные прикрепляются к некоторому коду, в Python называется замыканием.
- Условия, которые должны быть выполнены для создания замыкания в Python:
  - ❑ Наличие вложенной функция (функция внутри функции).
  - ❑ Вложенная функция должна ссылаться на значение, определенное в объемлющей функции.
  - ❑ Объемлющая функция должна возвращать вложенную функцию.
- Пример.

```
def print_msg(msg):  
    # объемлющая функция  
    def printer():  
        # вложенная функция  
        print(msg)  
    return printer      # возвращаем вложенную  
функцию!
```

## Замыкание (closure)

- Ссылка на переменную объемлющей функции действительна, даже когда объемлющая функция закончила работу, и переменная вышла из области видимости, или даже когда сама функция удалена!

```
another = print_msg("Здравствуйте!")  
del print_msg # Удаление функции!!!  
another()      # Осталось в памяти!!!
```

- Замыкания позволяют избежать использования глобальных (global) значений и обеспечивают некоторую форму сокрытия данных.
- Для этого также может использоваться объектно-ориентированный подход.
- Если в классе необходимо реализовать небольшое количество методов (в большинстве случаев один метод), замыкания могут обеспечить альтернативное решение.
- Когда количество атрибутов и методов достаточно велико, лучше реализовать класс.

## Каррирование (currying)

- Каррирование — преобразование функции от многих аргументов в набор функций, каждая из которых является функцией от одного аргумента.
- Существует возможность передать часть аргументов в функцию и получить обратно функцию, ожидающую остальные аргументы.

- Пример.
- Создадим простую функцию, которая принимает в качестве аргументов приветствие и имя:

```
def greet(greeting, name):  
    print(greeting + ', ' + name)  
greet('Hello', 'Ivan')  
# Выведет:  
# Hello, Ivan
```

## Каррирование (currying)

- Каррирование позволяет создать новую функцию для любого приветствия и передать этой новой функции имя:

```
def greet_curried(greeting):  
    def greet(name):  
        print(greeting + ', ' + name)  
    return greet
```

```
print("Через внутреннюю функцию:")  
greet_hello = greet_curried('Hello')  
greet_hello('Ivan')  
greet_hello('Alex')
```

```
print("Напрямую через каррирование:")  
greet_curried('Hi')('Peter')
```

## Понятие о декораторе

- Декоратором называется функция, которая имеет единственный параметр — другую функцию, и возвращает тоже функцию.
- Декоратор в Python добавляет дополнительный функционал к функции, не меняя её содержимое.
- Декоратор начинается с символа `@`, за которым следует название функции, которую необходимо «декорировать».
- Для получения декоратора нужно разместить его в строке перед определением функции.
- Последовательность создания декоратора:
  - ✓ Создание функции, которая создает функцию-обертку (*wrapper function*) для той функции, которая передана ей в качестве параметра.
  - ✓ Создание целевой функции.
  - ✓ Декорирование «обертывание» целевой функции.



ЦЕНТР  
ДОПОЛНИТЕЛЬНОГО  
ОБРАЗОВАНИЯ  
МГТУ им. Н.Э. Баумана



do.bmstu.ru