Baku Higher Oil School
SPE Student Chapter

πton School

7th session of πton School:

Deeper view on functions

Presentation by Huseyn Gambarov

20 November, 2022

Baku Higher Oil School
SPE Student Chapter

Who am I?

- Application Security Engineer at Kapital Bank
- Information Security Student at the Baku Higher Oil School
- Teaching assistant at BHOS
- Medallion of Excellence in WorldSkills Asia 2021, 3rd place Winner in the Baku Food Hackathon organized by the UNDP in Azerbaijan

Baku Higher Oil School
SPE Student Chapter

Pton School

Contents

Baku Higher Oil School
SPE Student Chapter

Nton School

Advanced argument passing

Arguments with default values

When passing arguments to the function we can give them default values, thus if the value for the particular argument is not given it will assign to it default value.

```python
def func(arg1, arg2, arg3):
    print(arg1)
    print(arg2)
    print(arg3)
```

Function without arguments

```python
def func(arg1=0, arg2=0, arg3=0):
    print(arg1)
    print(arg2)
    print(arg3)
```

Function with default arguments

Arguments with default values

- We can define default argument to some predefined variable.
- It is important to remember here that the default arguments are defined during the function definition. This means that if we **change "y"** later in code the **default value won't change**.

```python
y = 4

def func(x=y):
    print(x)

func()

y = 8

func()
```

*args, **kwargs

There are cases when it is possible to encounter the following statements:

- *args - stays for arguments
- **kwargs - stays for keyword arguments

Why it is used for and why is it useful?

- With the help of these we can simplify the argument passing to the function and avoid junk code.
- We can accept as many arguments as needed without predefining them initially.

Baku Higher Oil School
SPE Student Chapter

*args, **kwargs examples

Ton School

```
[5] def func(x, y, arg1 = 0, arg2 = 1):
      print(f"Args: {x}; {y}")
      print(f"Kwargs: {arg1}; {arg2}")

    values_tuple = (1, 2)
    values_dict = {
        "arg1": 3,
        "arg2": 4
    }

    print("Common way to pass arguments:")
    func(values_tuple[0], values_tuple[1], values_dict['arg1'], values_dict['arg2'])
    print()
    print("Pass arguments with args and kwargs:")
    func(*values_tuple, **values_dict)
```

```
Common way to pass arguments:
Args: 1; 2
Kwargs: 3; 4

Pass arguments with args and kwargs:
Args: 1; 2
Kwargs: 3; 4
```

Baku Higher Oil School
SPE Student Chapter

*args, **kwargs examples

Ton School

```
[6] def func(*args, **kwargs):
        print(f"Positional arguments: {args}")
        print(f"Keyword arguments: {kwargs}")

    func(1, 2, 3, val1 = 4, val2 = 5, val3 = 6)
```

```
Positional arguments: (1, 2, 3)
Keyword arguments: {'val1': 4, 'val2': 5, 'val3': 6}
```

Big combination

Simply we can combine all these features together, but as with everything else it is better to know the limit.

```
[8] def func(pos1, pos2, *args, key1 ='this', **kwargs):
      print(pos1, pos2, args, key1, kwargs)

    func(1, 2, 3, 4, 5, key1 = "This is first key", key2 = "Second key", arbitrary_key = "arbitrary key")

    1 2 (3, 4, 5) This is first key {'key2': 'Second key', 'arbitrary_key': 'arbitrary key'}
```

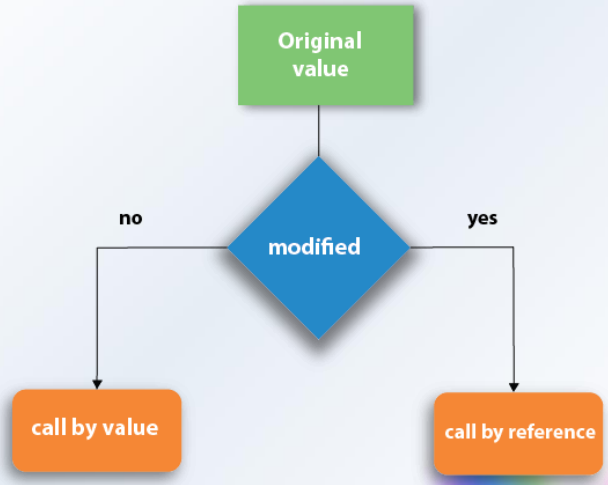
Baku Higher Oil School
SPE Student Chapter

Pton School

Mutability and side-effects of functions

Call by value & Call by reference

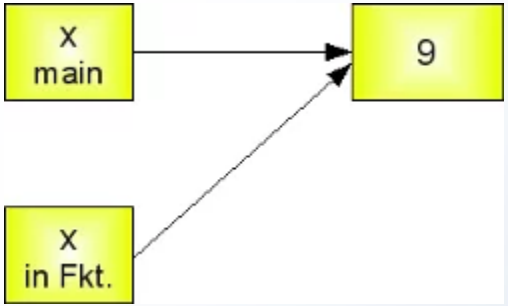
- These concepts were first introduced in C programming language and to some extend it exists in other programming languages as well.
- Why do we need to know this in python?
- Python has the concept of **mutability** and combined with functions it reintroduce this concept in Python.



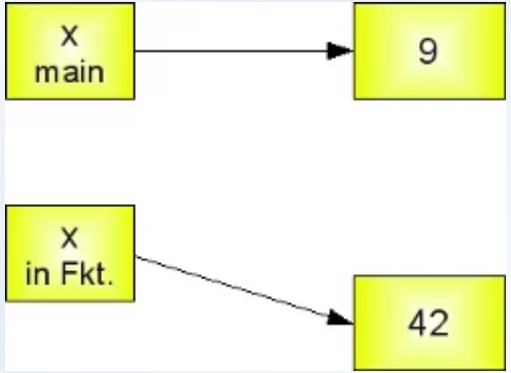
Immutable arguments

Immutable data types are:

- integer
- float
- bool
- string
- tuple



Behavior before the modification attempt



Behavior after the modification attempt

Mutable arguments

In addition to the immutable data types, there are mutable ones as well, such as:

- list
- dictionary
- set

When discussing this issue from the point of the mutable data types unwanted “side effects” arise.

Side effects

What are the side effects?

A function is said to have a side effect, if, in addition to producing a return value, it modifies the caller's environment in other ways. It might result in the function doing the following things:

- modify a global or static variable
- modify one of its argument
- raise an exception
- write data to a display or file
- etc

Baku Higher Oil School
SPE Student Chapter

Time for a break!

Let’s cool our brains!

Baku Higher Oil School
SPE Student Chapter

Ton School

Functions as Objects

Overview

- Functions behave like any other object, such as an int or a list.
- It means that functions could be easily used as:
  - arguments to other functions
  - store functions as dictionary values
  - return a function from another function.
- This leads to many powerful ways to use functions.

Higher Order Functions

A function that uses another function as an input argument or returns a function (HOF) is known as a **higher-order function.**

The simplest examples of such are map and filter functions that are known to you already.

```
[16] def is_even(x):
         return x%2 == 0

     print(list(filter(is_even, range(5))))

     [0, 2, 4]
```

Filter function example

```
[17] def square(x):
         return x ** 2

     print(list(map(square, range(5))))

     [0, 1, 4, 9, 16]
```

Map function example

Baku Higher Oil School
SPE Student Chapter

Custom functions

```
[21] def square(x):
       return x ** 2

     def cube(x):
       return x ** 3

     def custom_sum(values, function):
         return sum(map(function, values))

     values = range(5)
     print(custom_sum(values, square))
     print(custom_sum(values, cube))
```

```
30
100
```

Baku Higher Oil School
SPE Student Chapter

Returning functions

```
[25] def make_logger(target):
         def logger(data):
             with open(target, 'a') as f:
                 f.write(data + '\n')
         return logger

     foo_logger = make_logger('foo.txt')
     foo_logger('Hello')
     foo_logger('World')
```

Logger example

```
foo.txt ×
1 Hello
2 World
3
```

Logger result

```
[24] def by_factor(factor):

       def multiply(number):
         return number * factor

       return multiply

     double = by_factor(2)
     print("Double examples:")
     print(f"\t2 * 2 = {double(2)}")
     print(f"\t3 * 2 = {double(3)}")

     triple = by_factor(3)
     print("Triple examples:")
     print(f"\t2 * 3 = {triple(2)}")
     print(f"\t3 * 3 = {triple(3)}")
```

```
Double examples:
        2 * 2 = 4
        3 * 2 = 6
Triple examples:
        2 * 3 = 6
        3 * 3 = 9
```

Multiplication example

Baku Higher Oil School
SPE Student Chapter

Ton School

Lambda functions

What is lambda function?

Description of lambda

- **Lambda functions** are unnamed functions that will allow us return function values on the fly using lambda expressions.
- A lambda expression evaluates to a function that has **a single return expression** as its body. Assignment and control statements are not allowed.

Drawbacks

- **Lambda functions** are limited
- They are only useful for **simple, one-line functions** that **evaluate** and **return a single expression**. In those special cases where they apply, lambda expressions can be quite expressive.

Baku Higher Oil School
SPE Student Chapter

Lambda example

```
[31] def compose1 (f,g):
       return lambda x: f(g(x))


     square = lambda x: x * x

     print("Simple lambda function:")
     print(square)
     print(square(12))

     print("Higher order function used with lambda functons")
     f = compose1 ( lambda x: x * x, lambda x: x + 1)

     result = f(12)
     print(f"Evaluates to f(g(x+1)) where x = {12} => {result}")
```

```
Simple lambda function:
<function <lambda> at 0x7ff95cde5dd0>
144
Higher order function used with lambda functons
Evaluates to f(g(x+1)) where x = 12 => 169
```

Baku Higher Oil School
SPE Student Chapter

Tasks

Baku Higher Oil School
SPE Student Chapter

Args/kwargs

- Write a function that has four optional parameters (with defaults):
  - fore_color
  - back_color
  - link_color
  - visited_color
- It should print each variable and its values in the following format:
  - fore_color - <value>
  - back_color - <value>
  - link_color - <value>
  - visited_color - <value>
- Call the function using different ways:
  - Using just positional arguments:
    - func('red', 'blue', 'yellow', 'chartreuse')
  - Using just keyword arguments:
    - func(link_color='red', back_color='blue')
  - using *args and/or* *kwargs
    - regular = ('red', 'blue')
    - links = {'link_color': 'chartreuse'}

Baku Higher Oil School
SPE Student Chapter

Lambda functions

Write and demonstrate a Lambda function named stg() that appends .txt to its argument. What happens when you call the function with an integer?

Side effects of functions in Python

Perform research and submit a code that will show the side effects of functions in Python.

Baku Higher Oil School
SPE Student Chapter

Function analysis

Assume that s1 = "Hi" and s2 = "ya". In the function call model_two(s1 + s2):

* What is the argument for the function call?
* Write the implied assignment statement that happens during the call.
* What will be the value of the parameter word when model_two begins executing?
* Predict the output that will be produced by the function call

```
def model_two(word):
  ans = word * len(word)
  print(ans)

def main():
  print("Starting main...")
  w = input("Enter a word: ")
  model_two(w)
  print("All done!")
```

Baku Higher Oil School
SPE Student Chapter

Πton School

Thank you for your attention!

Hope It was a code-ful lesson!