Functions

January 14, 2021

1 Module 0 - Functions

A PDF version of this notebook is available at Module 0 - Python Basics

Python has many built-in functions. Some are available by default:

[1]:
```
## length function
s = [1, 2, 3]
len(s)
```

[1]:
```
3
```

Some other functions need to be imported from modules

[2]:
```
from math import sqrt
sqrt(4)
```

[2]:
```
2.0
```

The module can be imported entirely and the function can be retrieved from the module:

[3]:
```
import math
math.exp(1)
```

[3]:
```
2.718281828459045
```

1.0.1 *User-Defined Functions*

We can use the `def` keyword for a user-defined function. The `def` function is followed by the name of the function and in parenthesis the **optional** parameters. The `return` keywork is optional.

```
def name_of_function(parameter1, parameter2, ...):
    code block
    return parameter_to_return
```

The function below has no parameters, and returns `The print function is returning running func_1` when called.

[4]:
```
def func_1():
    print('The print function is returning running func_1')
```

[5]:
```
## to run the function just use the name followed by () with the optional␣
↪parameters
func_1()
```

```
The print function is returning running func_1
```

Notice that to call the function we need to write the function name followed by (), in this example, `func_1()`. Simply using the function name without the () refers to the function, but does not call it:

[6]:
```
func_1
```

[6]:
```
<function __main__.func_1>
```

A function can have more than 1 parameter.

[7]:
```
def func_2(a, b):
    return a * b
```

[8]:
```
func_2(3, 2)
```

[8]:
```
6
```

[9]:
```
## You can annotate the variable types by using : followed by var type
def func_3(a: int, b:int):
    return a * b
```

[10]:
```
func_3(2, 3)
```

[10]:
```
6
```

[11]:
```
## Annotating the variable type does not force the argument to have the␣
↪specific type. It is only for documentation
func_3('a', 3)
```

[11]:
```
'aaa'
```

[12]:
```
## Functions are objects which point to a memory location. This means we can␣
↪copy them to other objects
my_func = func_3
my_func(4,4)
```

[12]:
```
16
```

1.0.2 *Lambda Functions*

The lambda keyword creates a new function, but does not assign it to any specific name - instead it just returns the function object. It is generally used for short 1-line functions

[13]:
```
func_5 = lambda x: x**2
```

[14]:
```
func_5(3)
```

[14]:
```
9
```

You can use default values. These are called **keyword** arguments, while arguments with no default values are called **positional** arguments. **keyword** arguments are placed at the end of the function definition.

[15]:
```
def func_6(x, y = 2):
  return x/y
```

[16]:
```
## if y is ommitted, then the default value is used
func_6(4)
```

[16]:
```
2.0
```

[17]:
```
## You can call a function by using the parameter names
func_6(y = 3, x = 9)
```

[17]:
```
3.0
```