Programming with python™

By

Rami Tailakh
Senior Software Engineer and Data Science Practitioner
MSc in Applied Computing and Information Technology

Day-3 Agenda

- Day-2 Quick Review
- User-Defined Functions
- Built-in Functions
- Lambda Functions (Expressions)
- Modules

Day-2 Challenge Review-String Methods

1. Convert a dictionary into a list. Each key is repeated according to its value. E.g.:

   ```
   {'A':1, 'B': 2, 'C': 3}

      =>

   ['A', 'B', 'B', 'C', 'C', 'C']
   ```

2. Find: count, sum, and average of numbers in a list.

Python Functions

- A function in any programming language is a (named) block of code in form of a sequence of statements in a certain order
- These statements are executed when a function is called

Importance of Functions

- Re-usability
- No duplication
- Clarity and Readability
- Breaking down a complex problem into simpler blocks
- Maintainability
- Scalability

User-defined Functions

You can define your own Python function using the following syntax:

```
def <function_name>(<arguments comma separated>):
    """
    This is a document comment of this function (docstring)
    """
    # do something
    return <something>
```

Note: the docstring can be called by: <function_name>.**__doc__**

User-defined Functions
Rules for naming a Python function

- It must begin with: A-Z, a-z, or (_)
- It can contain: A-Z, a-z, 0-9, or (_)
- Keywords must be avoided
- For best practice, a function name should represent the implementation (what a function does)

User-defined Functions-Calling

```
>>> def sum(a,b):

         print("{}+{}={}".format(a,b,a+b))
```

Simply, by its name in addition to passing arguments, if any.

```
>>> sum(1,2)
```

```
Out:    1+2=3
```

User-defined Functions-Scope of Variables

```
def test():
        x='INNER'
        print(x)
>>> test()
```

Out: **INNER**

```
def test():
        x='OUTER'
        print(x)
>>> test()
```

Out: **OUTER**

User-defined Functions-Scope of Variables

```
>>> x='OUTER'

>>> def test():

        x='INNER'

        print(x)

>>> test()

>>> print(x)
```

Out:
```
INNER

OUTER
```

See also "global"

```
>>> x='OUTER'
>>> def test():
        global x
        x='INNER CHANGED'
        print(x)
>>> test()
>>> print(x)
```

Out:
```
INNER CHANGED

INNER CHANGED
```

User-defined Functions-Arguments Default Arguments

```
>>> def hello(name='mate'):
        print('Hello {}'.format(name))
>>> hello('Bob')
```

Out: **Hello Bob**

```
>>> hello()
```

Out: **Hello mate**

User-defined Functions-Arguments Default Arguments-Order of Passing

Arguments can be passed in any order, only, if we specify their argument names as shown below:

```
>>> def divide(x,y):
          return x/y
>>> divide(y=2,x=1)
```

Out: **0.5**

User-defined Functions-Arguments Arbitrary Arguments

- Number of arguments may not be known
- Use an asterisk (*) before an argument name

```
>>> def hello(*names):
    for name in names:
        print("Hello {}".format(name))
>>> hello('Adam', 'Samer','Everyone')
```

Out:

```
Hello Adam
Hello Everyone
```

Built-in Functions

- map, abs, ascii, type, str, float, list, enumerate and many more ...

```
>>> for i,x in enumerate(['a','b','c']):
            print(i,x)
```

Out:

```
0 a
1 b
2 c
```

Recursive Functions

- In programming, functions can call other functions
- In programming, a function can also call itself; aka **recursion**
- However, there should be a decrement statement, to avoid an **infinite loop**

Recursive Functions-Continue

```
def factorial(n):
    result = 1
    for i in list(range(n,0,-1)):
            result *= i
    return result
```

```
def factorial(n):

    return n*factorial(n-1) if n > 1 else 1
```

On Margin-Ternary Operator in Python

- Code compact; allows to test a condition in a single line replacing the multiline if-else

```
a if condition else b
```

```
>>> number = 2
>>> message = 'Positive' if number > 0 else
'Negative'
>>> print(message)
Out:    Positive
```

Lambda Functions

- Python also allows to define a function anonymously besides the “`def`”
- Anonymously; lambda concerns the function instead of assigning it to a name
- Lambda functions does some computation and implicitly returns a value

Lambda Functions-Continue

```
>>> divide_by_two = lambda x: x/2
>>> divide_by_two(4)
Out:   2
```

Functions vs Methods

- A Python method is just like a function, but it is related to an **object**

```
>>> my_list = [10, 5, 12, 22, 4]
```

function	method
`>>> sorted(my_list)`	`>>> my_list.sort()`

Try this:

```
>>> print(sorted(my_list))

>>> print(my_list)
```

What do you find?

Modules

- A Python module is a file that contains Python statements and definitions
- Good practice for longer programmes
- It is saved in a file with .py extension
- Python modules can be imported using the keyword `import`

module_1.py

```
def func1():
	<do something>
def func2():
	<do something>
```

In a different block of code:

```
import module_1
```

Challenges

1. Write a Python function that computes the multiplication of all the numbers in a list.
2. Write a Python Program to Display Fibonacci Sequence. For example: 0, 1, 1, 2, 3, 5, 8, 13 and so on...
3. Write a Python program to print Even Numbers in a List. Hint: use **filter** function and **lambda** expression