

# Introduction to Python

## #### About Python Language

Remember that you are intelligent and you can learn, but the computer is simple and very fast, but can not learn by itself. Therefore, in order for you to communicate instructions to the computer it is easier for you to learn a computer Language (e.g. Python) than for the computer to learn English.

Python can be **asy to pick up and friendly to learn**. Python is a **general-purpose** interpreted , interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. There are two main python versions: 2.7 and 3. For this course we will use 2.7 since it is the most common or popular used.

## Basic Practise

Let's get familiar with python by playing in the terminal in the interactive mode (you type a line at a time and the interpreter responds). You invoke the interpreter and brings up the following prompt:

```
$python
Python 2.7.9 (default, Sep 17 2016, 20:26:04)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Strings, integers, and floating points:

```
>>> print "Hello, Python!"
>>> x = 1          # Integer assignment
>>> y = 1005.00    # Floating points
>>> name = "John"  # A string
>>> print x
>>> print y
>>> print name
```

In Python, the **standard order of operations** are evaluated from left to right following order (memorized by many as PEMDAS):

Name	Syntax	Description
<b>Parentheses</b>	( ... )	Happening before operating on anything else.
<b>Exponents</b>	**	An exponent is a simply short multiplication or division, it should be evaluated before them.
<b>Multiplication and Division</b>	* /	Multiplication is rapid addition and must happen first.
<b>Addition and Subtraction</b>	+ -	

*Note:* // is the floor division in which the digits after the decimal point are removed. But if one of the operands is negative, the result is floored, i.e., rounded away from zero.

```
>>> 3/4 * 5 # First division and then Multiplication
>>> 3.0 / 4 * 5
>>> (3.0 / 4) * 4
>>> 2**8
>>> -11.0//3
>>> 11.0//3 # Result floored (rounded away from zero)
>>> -11.0/3
>>> z = float(5)
>>> z
>>> z = int(5.25)
>>> z
>>> 10%7 # Remainder of a division
>>> 'abc' + 'fgb' # strings
```

Comparison operators:

Name	Syntax
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
==	Equal to
!=	Not equal to

```
>>> 2 == 3
False
# We got a boolean
>>> 3 == 3
True
>>> 2 < 3
True
>>> "a" < "aa"
True
```

## Data Types

The data stored in memory can be of different types, python has 5 types: **Numbers, Strings, List, Tuple, and Dictionary.**

```
>>> type(x) # numbers
>>> type(y)
```

```
>>> type(name) # String
```

**Strings** in Python are set of characters represented in the quotation marks. Python allows for either pairs of single or double quotes.

A subsets of strings can be taken using the slice operator ([ ] and [ : ] ) with indexes starting at 0 in the beginning of the string and working their way from -1 at the end.

The plus (+) sign is the string concatenation operator and the asterisk (\*) is the repetition operator. For example:

```
>>> string = 'Hello World!'
>>> print string          # Prints complete string
>>> print string[0]       # Prints first character of the string
>>> print string[2:5]     # Prints characters starting from 3rd to 5th
>>> print string[2:]      # Prints string starting from 3rd character
>>> print string * 2      # Prints string two times
>>> print string + "TEST" # Prints concatenated string
```

**Lists** are the most versatile data types in Python. A list contains items separated by commas and enclosed within square brackets ([ ])—similar to arrays in C. One difference between them is that all the items belonging to a list can be of different data type.

The values stored in a list can be accessed using the slice operator ([ ] and [ : ]) with indexes starting at 0 in the beginning of the list and working their way to end -1. The plus (+) sign is the list concatenation operator, and the asterisk (\*) is the repetition operator.

```
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tinylist = [123, 'john']

>>> print list          # Prints complete list
>>> print list[0]       # Prints first element of the list
>>> print list[1:3]     # Prints elements starting from 2nd till 3rd
>>> print list[2:]      # Prints elements starting from 3rd element
>>> print tinylist * 2   # Prints list two times
>>> print list + tinylist # Prints concatenated lists
```

A **tuple** is another sequence data type that is similar to the list. It consists of a number of values separated by commas. Unlike lists, however, tuples are enclosed within parentheses.

The main differences between lists and tuples are: Lists are enclosed in brackets ( [ ] ) and their elements and size can be changed, while tuples are enclosed in parentheses ( ( ) ) and cannot be updated—**immutable**. Tuples can be thought of as read-only lists.

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
>>> tinytuple = (123, 'john')

>>> print tuple          # Prints complete list
>>> print tuple[0]       # Prints first element of the list
>>> print tuple[1:3]     # Prints elements starting from 2nd till 3rd
>>> print tuple[2:]      # Prints elements starting from 3rd element
>>> print tinytuple * 2   # Prints list two times
>>> print tuple + tinytuple # Prints concatenated lists
```

Invalid operations on a tuple but valid on a list:

```
>>> tuple = ( 'abcd', 786 , 2.23, 'john', 70.2 )
>>> list = [ 'abcd', 786 , 2.23, 'john', 70.2 ]
>>> tuple[2] = 1000      # Invalid syntax with tuple
>>> list[2] = 1000      # Valid syntax with list
```

Python's **dictionaries** are kind of hash table type. They work like associative arrays and consist of key-value pairs. A dictionary key can be almost any Python type, but are usually numbers or strings. Values, on the other hand, can be any arbitrary Python object. Dictionaries are enclosed by curly braces ({}), and values can be assigned and accessed using square braces ([]).

```
>>> dict = {}
>>> dict['one'] = "This is one"
>>> dict[2] = "This is two"
# keys are: name, code and dept; values are: john, 6734 and sales
>>> tinydict = {'name': 'john', 'code':6734, 'dept': 'sales'}

>>> print dict['one']      # Prints value for 'one' key
>>> print dict[2]          # Prints value for 2 key
>>> print tinydict         # Prints complete dictionary
>>> print tinydict.keys()  # Prints all the keys
>>> print tinydict.values() # Prints all the values
```

To quit the Python interpreter:

```
>>> quit()
```

## Scripts

A Script is a sequence of statements (lines) into a file using a text editor and tell Python interpreter to execute the statements in the file.

- We can write a program in our script like a recipe or installation of a software. At the end of the day, a program is a **sequence** of steps to be done in order.
- Some of the steps can be **conditional**, that means that sometimes they can be skipped.
- Sometimes a step or group of steps are to be **repeated**.
- Sometimes we store a set of steps that will be used over and over again in several parts of the program (**functions**).

**Note:** Have a look on the code [style guide](#) for a good coding practise. As a first good practise, do not name files or folders with space in between: Aful! → example 1.py; Great! → **example\_1.py**, **exampleOne.py**, **example\_one.py**

We will make a simple script:

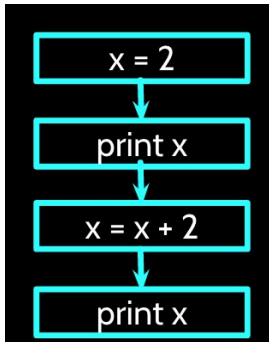
```
$ pwd
$ /home/pi
$ mkdir codes/python_examples
$ cd codes/python_examples
$ nano example_flow.py
```

Then you can type in the editor:

```
#!/usr/bin/env python
x = 2
```

```
print x
x = x + 2
print
```

When a program is running, it flows from one step to the next. As programmers, we set up “paths” for the program to follow.



Close the text editor and then you can execute it on two ways:

```
$ python example_flow.py
```

The other is to give the script the access permissions to be an executable file through the [chmod](#) linux command:

```
$ chmod +x example_flow.py
$ ./example_flow.py
```

References [ [1](#), Charles Severance course: Python for everybody]