

# 01-Introduction-Environment-Interpreter

September 24, 2018

## Introduction to Python programming

Nicola Spallanzani  
*n.spallanzani@ Cineca.it*



### 0.1 Outlook

- Philosophy
- Environment
- Interpreter
- Built-in types and operations
- Program file
- Built-in containers
  - Tuples
  - Lists
  - Sets
  - Dicts

## 0.2 Outlook

- Flow control constructs
  - if-elif-else
  - For loops
  - While loops
  - Comparison and logical operators
    - \* Object reference
- File I/O
- Functions
  - Argument Passing
  - Doc string
  - Lambda functions

## 0.3 Outlook

- Introspection
- Functional programming
  - sort
  - lambda functions
  - filter, map
- String formatting
  - Old and new style
- Modules
  - Packages
  - Program arguments

## 0.4 Outlook

- Standard Library
  - re
  - datetime
  - math, random
  - os.path, glob
  - sqlite3, csv, json
  - sys, os, time
  - argparse
  - logging
  - subprocess

## 0.5 Outlook

- Scientific Modules (very quick introduction)
  - Numpy
  - Matplotlib
  - Scipy
  - Pandas

## 0.6 Outlook (extra)

- Classes
  - Instances
  - Methods
  - Attributes
  - Inheritance
- Iterables and Iterators
- Error handling

# 1 introduction

## 1.1 philosophy

Python is a programming language with many good features:

- Very easy to use
- Easy to learn (looks like pseudo-coding)
- Excellent readability (there is only one way to do anything)
- Excellent portability

## 1.2 philosophy

Do not call it a “scripting language”! Although it can be used as such, it is much more.

It is an high level language, modern, complete, with which it is possible to realize highly complex software.

It is an interpreted language, but it would be more appropriate to call it “dynamic language”.

## 1.3 philosophy

A dynamic language is a high level language in which many of the controls are executed run-time, while in other languages they are done at compile time.

Actually python “compile” the source into bytecode that runs on a virtual machine (like Java).

## 1.4 philosophy

It is a multiparadigm language: \* Imperative \* Object-oriented \* Functional \* Structural \* Aspect-oriented \* Design by contract (with an extension) \* ...

## 1.5 The Zen of python

```
In [1]: import this
```

The Zen of Python, by Tim Peters

```
Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

## 1.6 disavtanges?

Python is often considered a slow language. To a large extent this is true: it is slower than Java, for example.

But the speed is not always the bottleneck. Often, the management of the complexity is a problem more important than speed.

However, there are several ways to make faster the “critical” parts of a python program.

## 1.7 performance

“People are able to code complex algorithms in much less time by using a high-level language like Python (e.g., also C++). There can be a performance penalty in the most pure sense of the term.”

## 1.8 optimization

“The best performance improvement is the transition from the nonworking to the working state.”  
–John Ousterhout

“Premature optimization is the root of all evil.” –Donald Knuth

“You can always optimize it later.” – Unknown

## 1.9 python2 / python3

There are two main branches of development of python: \* python2: it is a “dead” branch, that is maintained, but that will not undergo updates / evolution \* python3: it is the new python, actually still little used compared to python2.

The differences are not huge, also there is the utility 2to3 to convert code from the old to the new python.

## 1.10 python3

- We will take care mainly of python3.
- I will try to point out the main differences.

## 2 environment

### 2.1 environment

With the installation of the Python interpreter it comes also the so called standard library.

In order to use it you need to import the modules using the keyword **import**.

```
In [1]: import math
        print(math.pi)
```

```
3.141592653589793
```

### 2.2 virtualenv

Virtualenv is a tool to create isolated Python environments.

It allow a user to install python modules in its private area without need **root** permissions.

Virtualenv has one basic command:

```
$ virtualenv ENV
```

Where ENV is a directory to place the new virtual environment.

### 2.3 virtualenv

A virtual environment needs to be activated before using it:

```
$ source ENV/bin/activate
```

on Windows:

```
> env\Scripts\activate
```

Simply type deactivate to exit from the virtual environment.

## 2.4 virtualenv

At the prompt will be added a prefix to emphasize that the environment is active.

```
(ENV) $
```

An interesting option allow you to choose the interpreter executable to use to create the virtual environment.

```
$ virtualenv --python=python2.7 ENV
```

## 2.5 pip

pip is the Python Packaging Authority ([PyPA](#)) recommended tool for installing Python packages from the Python Package Index ([PyPI](#)).

```
(ENV) $ pip search peppercorn
peppercorn (0.5)          - A library for converting a token stream into a data structure for use :
pepperedform (0.6.1)    - Helpers for using peppercorn with formprocess.
```

## 2.6 pip

pip take care of installing dependencies before their dependents, i.e. in “topological order”.

```
(ENV) $ pip install peppercorn
```

or, if you prefer not the latest version

```
(ENV) $ pip install peppercorn==0.4
```

## 2.7 pip

It is possible to use pip to install packages downloaded by the user.

```
(ENV) $ wget https://some_url/SomeProject.zip
```

```
(ENV) $ unzip SomeProject.zip
```

```
(ENV) $ pip install -e SomeProject
```

The option `-e` allow to install local projects in “editable” mode.

## 2.8 test (virtualenv)

- Create a virtual environment
- Activate it
- Install the latest version of jupyter on it

## 3 interpreter

### 3.1 interpreter

- Python is an interpreted language
- The interpreter performs a compilation from source to bytecode, which is then executed on a virtual machine, as in Java
- The interpreter is also an excellent “calculator”, to be used in interactive!

```
In [2]: 2**1024
```

```
Out [2]: 17976931348623159077293051907890247336179769789423065727343008115773267580550096313270
```

### 3.2 interactive mode

To launch the interpreter in interactive mode:

- from a terminal or a command prompt simply type python
- launch the idle program

### 3.3 interactive mode

- When used in interactive mode, the interpreter acts a little differently.
- This is the prompt: “>>>”
- If an expression has a value, it is printed automatically:

### 3.4 interactive mode

- Any error can occur during the execution of the interpreter in interactive mode, the interpreter survives, even in case of SyntaxError:

```
In [3]: 5/0
```

```
-----  
  
ZeroDivisionError                                Traceback (most recent call last)  
  
  <ipython-input-3-67a69f72677d> in <module>()  
----> 1 5/0  
  
ZeroDivisionError: division by zero
```

### 3.5 interactive mode

```
In [4]: fact(100)
```

```
-----  
NameError                                Traceback (most recent call last)  
  
  <ipython-input-4-dd06d576a896> in <module>()  
----> 1 fact(100)  
  
NameError: name 'fact' is not defined
```

```
In [5]: @@@ausfd?=
```

```
File "<ipython-input-5-0dc3d51d84fc>", line 1  
@@@ausfd?=  
^  
SyntaxError: invalid syntax
```

### 3.6 print()

To print on standard output you may use the `print()` function.  
It can take any number of arguments:

```
In [6]: print(0, 2, 4)  
        print(6)
```

```
0 2 4  
6
```

```
In [7]: print(0, 2, 4, end='')  
        print(6)
```

```
0 2 46
```

### 3.7 python2

Please note, in python2 `print` is an operator:

### 3.8 ipython interpreter

Born in 2001 as a work of a student (Fernando Perez).

Based on features he liked in Mathematica and trying to create a system for everyday scientific computing.



- Command history, which can be browsed with the up and down arrows on the keyboard.
- Tab auto-completion.
- In-line editing of code.
- Object introspection, and automatic extract of documentation strings from python objects like classes and functions.
- Good interaction with operating system shell.
- Support for multiple parallel back-end processes, that can run on computing clusters or cloud services like Amazon EE2.

### 3.9 ipython notebook

[IPython notebook](#) is an HTML-based notebook environment for Python

- Based on the IPython shell
- Provides a web cell-based interactive environment powered with Javascript
- System profiles to access unix-terminal-like capability
- Comments and notes with HTML and markdown formats
- Integrates embedded plots

### 3.10 jupyter project

In 2014, Fernando Perez announced a spin-off project from IPython called Project Jupyter (**J**ulia **PY**thon and **R**).

IPython will continue to exist as a Python shell and a kernel for Jupyter, while **the notebook** and *other language-agnostic parts of IPython* will move under the Jupyter name.

Jupyter added support for Julia, R, Haskell and Ruby.

source: [https://en.wikipedia.org/wiki/IPython#Project\\_Jupyter](https://en.wikipedia.org/wiki/IPython#Project_Jupyter)

## jupyter notebook

Jupyter notebooks are based on **jupyter kernels**.

- A ‘kernel’ is a program that runs and introspects the user’s code.

A notebook is crazy simple and fun:

- markdown and notes (consider to [learn markdown language](#))
- download ipynb, python, html
- install a library and use
- slideshow it

### 3.11 jupyter notebook

You can start it from a terminal by running `jupyter notebook`

First, we need to explain how to run cells. Try to run the cell below!

```
In [ ]: print("Hi! This is a cell. Press the button above to run it")
```

You can also run a cell with Ctrl+Enter, Shift+Enter or Alt+Enter. Experiment a bit with that.

### 3.12 jupyter notebook

One of the most useful things about IPython notebook is its tab completion.

Try this: click just after `read_csv()` in the cell below and press Shift+Tab 4 times, slowly

```
In [4]: import pandas as pd
```

```
In [ ]: pd.read_csv()
```

Okay, let's try tab completion for function names!

```
In [ ]: pd.r
```

### 3.13 magic functions

IPython has all kinds of magic functions. Here's an example of comparing `sum()` with a list comprehension to a generator comprehension using the `%time` magic.

```
In [5]: %timeit sum([x for x in range(1000000)])
```

60 ms ± 1.59 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)

```
In [6]: import numpy as np
        %timeit np.arange(1000000).sum()
```

2.34 ms ± 116 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)

For a list of magic functions type `%quickref`

### 3.14 launch a python program

Form a terminal or command prompt you can simply type `python` and the name of a file containing python code

```
$ python codes/hello.py    (for Linux and MacOS)
```

```
> python codes\hello.py    (for Windows)
```

From a jupyter notebook use the `%run` magic

```
In [10]: %run codes/hello.py
```

Hello World!

## 4 python program file

### 4.1 python program file

### 4.2 python program file

Let us analyze the content: \* The first line `#!/usr/bin/env python3` It is nothing more than a comment but instructs the Unix shell that will be used the interpreter `python3` to run the program. The trick `/usr/bin/env` is to avoid worrying about the exact path of the python installation, indeed the `env` command is able to find it.

- It must be the first line of the file.
- The file name is arbitrary (not necessarily have to end in `.py`).

### 4.3 python program file

Let us analyze the content: \* The line `# this is a useless comment` is a comment, and it is ignored. \* Also the next line contains an inline comment. All that is after a `#` until the end of the line is not interpreted.

### 4.4 source encoding

- The python code can be written in any system of characters. By default, if you do not declare anything, it is `utf-8`.
- But it can be anything, even Chinese.

### 4.5 python2

- In `python2` the default encoding is `latin1`, and it is nothing but ASCII characters. Accented letters, for example, are not allowed, not even in the comments.
- To change the encoding is used a line like this at the beginning of the file:

```
# -*- coding: utf-8 -*-
```

(PEP 0263)

### 4.6 test (name.py)

- Define a string `name` that contains your name, a string `surname` containing your last name;
- Print the length of the two strings;
- Concatenate the two strings forming `name_surname`;
- Print the length of `name_surname`.