Pythonista Notes Documentation

Release 0.1

Nguyen Cao

Aug 05, 2017

Contents

CHAPTER 1

Table of Contents

About Python

What's Python ?

- Python is an interpreted language, it is not a compiled language. This means that it needs to be run by another program, called the interpreter (*/usr/bin/python* in Linux or *python.exe* in Window) to generate the bytecode `*.pyc`, rather than the processor itself. Not like as C, which runs directly on the processor, after a compilation to bytecode.
- Because of interpreted languages, Python is a high-level programming language. For exemple, an important feature of high-level programming language is *garbage collector*, which free automatiquement the variable on memory at the end of program or function, to avoid memory leak.
- In France, in 2017, python becomes the best programmation language (after this french site)

Compare Python and other language

This discussion is very huge on the internet, and there's some notes :

Feature	Python	C
Pointer	In python we dont have the definition of pointer. All variables are names bound to objects ! We will speak it at topic pointer .	Yes
Type	Dynamic ! we can change object type type of variables at run-time	Static typing !
Varaible location	All objects in python stock in **heap** (whatever `int`, `float`...). Only the name of variable sits in **stack**	We can define where we want to stock via pointer
Memory & Performance	Very inefficient ! Example, a list [1,2,3] take 200 bytes. Slowly !	Very efficient ! Ex, a list {1,2,3} take only 16 bytes Fast !

Python 2.7

- This version is the most stable release, and the most frequently used. This is scheduled to be the last major version in the 2.x series before it moves into an extended maintenance period. For me, I always work with version 2.7, so all of my notes in this site go with this version.
- We must know that in 2020, python 2.7 will be not supported anymore and all modules will be released only for python3 :(Sometimes I try it, but the syntax is very different from version 2 but we have no choice, so now this's time to move to python 3 ! I advise that if we start a project now, we should try & work with version 3 with their packages/modules, and our project will be safe in 2020 :)
- To show the current python using, on linux, we do

```
$ ls -l /usr/bin/python
/usr/bin/python -> python2.7
```

Install Python & Package

Install Python

Linux

Gererally, python is installed by default in Linux OS at the path */usr/bin/python*. To find this path, there are many ways, for me I like :

```
$ which python
/usr/bin/python
```

Window

To isntall python, we do these steps:

- Download python installation at this (site). Select the version you need to.
- Double click on the downloaded file, then click next next ... to install. You can choose the specific directory also. For me I put it as default directory *D:Python27*
- To use easily, we should create a python environment variable on window :
  - Open Control Panel, then System
  - Click 'Advanced system settings' on the left
  - Click the 'Environment Variables' button
  - Under 'System variables' click the variable called 'Path' then the 'Edit...' button. (This will set it for all users, you could instead choose to edit the User variables to just set python as a command prompt command for the current user)
  - Without deleting any other text, add D:Python27; (include the semi-colon) to the beginning of the 'Variable value' and click OK.
  - Click OK on the 'Environment Variables' window.

Install Module Python

- As a popular open source development project, Python has an **active** supporting community of contributors and users that also make their software available for other Python developers to use under open source license terms.
- *This allows Python users to share and collaborate effectively*, benefiting from the solutions others have already created to common (and sometimes even rare!) problems, as well as potentially contributing their own solutions to the common pool.
- **pip** is the preferred installer program. Starting with Python 3.4, it is included by default with the Python binary installers.

Pip Installation

Linux

To install python

```
$ sudo apt-get install python-pip python-dev build-essential
$ sudo pip install --upgrade pip
$ sudo pip install --upgrade virtualenv
```

Window

With the current python vesion, *pip* is located at <path_to_python_dir>/Python27/Scripts/ We see pip.exe : that is pip application. So to use easily, we should create a environement variable. Unless, each time we want to install a package, navigating to pip directory, and typing the pip commands.

Basic Usage

To install the package, you can use the following command :

- Search a package :

  $pip search package_name

- Install :

  $pip install package_name

- Uninstall :

  $pip uninstall package_name

Run Python Script

Run Python

To run a script in python, just type *python script.py* , then our program will compile and run at the same time

---

Note: In fact, python use a program, called the interpreter (*/usr/bin/python* in Linux or *python.exe* in Window) to compile our source code to the bytecode `*.pyc`, then execute this bytecode.

Python Interactive

The Python interpreter is usually installed at the path of python installation. To open this, juste typing `python` to the shell, we have :

```
$ python
Python 2.7.9 (default, Mar  1 2015, 12:57:24)
[GCC 4.9.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

and from now, we can apply and test all python commands here !!

Tip: For me, this tool is very important on my work. When I write a new function, always I test it directly on Interactive Console with maximum testcase possible. This way helps me avoid many stupid mistakes.

Python command options

We mostly use `python -m mymodule` to run a python source code . But there are other common command-lines options :

```
python [-c command | -m module-name | script | - ] [args]
```

-c

The -c cmd option can be used to execute short programs in the form of a command-line option—for example:

```
$ python -c "print('hello world')".
hello world
```

-m

Runs a library module as a script which executes inside the `__main__` module prior to the execution of the main script. For this command, example `python -m mymodule` : Python does 2 things :

- First, import the packages `mymodule`. If the given module isn't located on the Python module path, an error is handled here and the program will be stop.
- Second, run this module `mymodule` like as a script.

Exemple : I have a script **foo.py**

```
print 'hello'
print __name__
```

then we run this script by 2 ways:

```
$ python foo.py
hello
__main__
$ python -m foo
hello
```

We have the same result ! Attention with the path to our module, it raise an error if the module isn't in the PYTHON_PATH.We shall see it at sys module

---

Tip: Programs must be written for people to read, and only incidentally for machines to execute. — Abelson & Sussman, Structure and Interpretation of Computer Programs

---

Python Code Style

Text Editor

I recommend these text editors for python development :

- Sublime text : very beautiful interface, Python syntax highlighting, Python plugins.
- vim : for all *linuxer*
- NotePad++ : I always use this editor although my friends mocking me :((Having a perfect NppFPT for virtual machine, and mostly it has an option to backup all my source code each time I do Ctrl+S.
- Pycharm : Full-featured IDE for Python. I tried it once, a very nice interface, autocorrect following PEP8 standart, and efficient but it's so slow.

Indentation

Whitespace 1

- 4 spaces per indentation level.
- Never mix tabs and spaces.
- One blank line between functions.
- Two blank lines between classes.

Whitespace 2

- Add a space after "," in `dict`, `list`, `tuple`, & *argument lists*, and after ":" in dicts, but not before.
- Put spaces around assignments & comparisons (except in argument lists).
- No spaces just inside *parentheses* or just before argument lists.

Exemple:

```
def make_squares(key, value=0):
    """
    Return a dictionary and a list.

    @param
        key : string or numeric
        value : any type, by default is 0
    @return
        a tuple 2 element, the first is dictionnary, other is list

    - Exemple :
        make_squares(4)
        make_squares(4,99)
    """
    d = {key: value}
    l = [key, value]
    return d, l
```

Convention in Python for variable and function names

- Class: ClassName
- Method name : method_name
- Function : names should be lowercase, with words separated by underscores as necessary to improve readability , example `function_name`.
- Variables : lowercase with words separated by underscores as necessary to improve readability.
- Private methods and properties start with __double_underscore
- "Protected" methods and properties start with _single_underscore
- global_var_name
- instance_var_name
- local_var_name
- function_parameter_name
- Constant name : GLOBAL_CONSTANT_NAME
- ExceptionName : ExceptionName

Ignored variable

If you need to assign something but will not need that variable, use *the double underscores* __ (not a single underscore _ in order to avoid confusion with variable to store the result of the last evaluation) :

```
filename = 'foobar.txt'
basename, __, ext = filename.rpartition('.')
```

Docstrings & Comments

- **Docstrings** : Explain **how** to use code, and are for the users of our code. This is written between 2 triple-quotes. This must always have 3 things :
  - Purpose of the function
  - Description the given parameters (name, type, note), we use **@param** ; the return values (name, type, note), we use **@return**.
  - Un example to run this function

A linebreak after a block. Exemple :

```
def sum3(a,b,c) :
    """
    This function to get the sum of 3 given numbers.

    @param:
        a, b, c : numeric type, raise exception if it lacks one
    @return:
        my_sum : numeric type

    - Example : sum3(3, 4.4, -1)
    """
    return a + b + c
```

Note: When the function is called, the *Docstrings* is in method `__doc__`. For the above example, typing `print sum3.__doc__` or `help(sum3)` in python interactive and it show our docstring.

- **Comments** : Explain **why**, and are for the maintainers of our code. Genarally there are 3 types :
  - Block Comments
  - Inline Comments
  - Commenting Out Code for Testing

autopep8

The library **autopep8** automatically formats Python code to conform to the PEP 8 style guide. So good ! For example, I have a python script named *my_script.py* was bad written, by using this lib, we are safe !

```
pip install autopep8
autopep8 --in-place my_script.py
```

if __name__ == "__main__"

Sometimes we see this notion in source code, that means if we run directly the script from terminal, these command-lines in `if` block will be executed .By example we have a script **a.py** :

```
if __name__ == "__main__":
    print 'hello'
```

Then run in cmd:

```
>>> python a.py
hello
```

But if we import **a** into another script python, all commands in if `__name__ == "__main__"` will be not execute, because in this case, `__name__` become 'a'. Exemple we have the script **a.py** like as above, then we import **a.py** into **b.py**:

```
import a
if __name__ == "__main__":
    print 'hello b'
    print a.__name__
```

we run :

```
>>> python b.py
hello b
a
```

What's the use ?

This thing's used for testing when we write a new module or new sub-script in a grand project. For my above exemple, I can write some testsuite after *if __name__ == "__main__":*

Operating_System_Modules

This section speak about 2 important modules in Python, `os` and `sys`.

Basic

Tips

Variable Types

Built-In Types : 12

Basic

Mutable Object

Pointer

```
>>> i = 5
>>> j = i
>>> j = 3
>>> print(i)
5
```

We see that `i` refers to an integer on memory has value 5 at first line, then `j` refers to `i`, means `j` also refers to 5. But when we change `j` =3, that means `j` points to another location on memory. Because `i` is an integer which is an immutable object, so there'is not any change on `i`. And whats about mutable object *list* ?

```
>>> a = [0, 1, 2, 3, 4]
>>> b = a
>>> b[2] = 9999
>>> a
[0, 1, 9999, 3, 4]
```

If 2 *lists* `a` and `b` refer at same object, when `a` changes, `b` changes also !

Tips

Type Numbers

Boolean

```
>>> x = -1
>>> if x :
        print "display !"
>>> 'display'
```

Basic

Tips

Type Sequence (1) : String

Basic

Tips

Type Sequence (2) : Unicode

Basic

Tips

Type Sequence (3) : List

Basic

Tips

Type Sequence (4) : Tuple

Basic

Tips

Types Mapping Set (1) : Dictionary

Basic

Tips

Types Mapping Set (1) : Set

Basic

Tips

Program Structure

Basic

- **Loop Iterations** `for`, `while`
- **Conditional statement** `if`

Tips

Function

Basic

Tips

Class

Basic

Tips

Multithreading

High-performance computing HPC Parallel computing

Basic

Tips

Django : Web FrameWork

Basic

Template

Multiple

Using a*b : {% widthratio a 1 b %}