Python Installation and Crash Course

2020

Ando Ki, Ph.D.
adki@future-ds.com

Contents

- What is Python?
- Python package installation
- Python installation with Virtualenv on Ubuntu
- Running Python

What is Python?

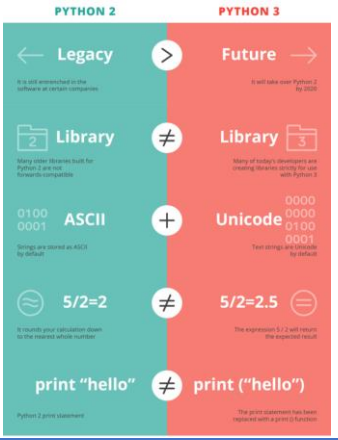
- Python is a programming language freely available from www.python.org.
- Python is an interpreted, interactive, object-oriented programming language.
  - interpreted: it is processed at runtime by the interpreter
  - interactive: interact with the interpreter directly
  - object-oriented: encapsulates code within objects

- Python version
  - Python 2.7: ~2010
  - Python 3.x: 2008~
  - Python 3.7.4: 2019
    - *not backward compatible*:
    - https://docs.python.org/3/whatsnew/3.0.html

Apollo

Python (dragon or snake)

Delphi (town in Greece)



Python package installation

- Ways of installing python
  - PIP: pip is a tool for installing Python packag es from the Python Package Index.
    - Python Package Index (PyPI) – The official thir d-party software repository for the Python prog ramming language; https://pypi.python.oprg/py pi

- Virtualenv
  - Python has version dependencies and direct ory sensitivity
    - To deal with this issue, use virtual environment
      - Virtualenv is a tool to isolate python environment
    - or use '**import __future__**'
      - a pseudo-module which programmers can use to enable new language features which are not co mpatible with the current interpreter.

- **pip**: the Python Package Manager. (pip is not included with python by default.)
- **pyenv**: Python Version Manager (easily switch between multiple versions of Python)
- **virtualenv**: Python Environment Manager.
- **Anaconda**: Package Manager + Environment Manager + Additional Scientific Libraries.

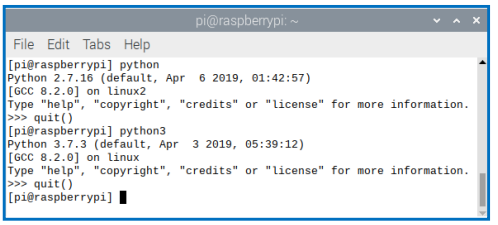
The main purpose of Python virtual environments is to create an isolated environment for Python projects, since each project can have its own dependencies.

Python installation: reference

- Unix and Linux
  - https://www.python.org/downloads/
  - interpreter: /usr/local/bin/python
  - library: /usr/local/lib/python*XX*
    - *XX*: version
      - Python 2.7: ~2010
      - Python 3.x: 2008~
- Windows
  - https://www.python.org/downloads/
- Environment variables
  - **PYTHONPATH**
    - where python interpreter, module files, and source code
  - **PYTHONSTARTUP**
    - where initialization file
  - **PYTHONCASEOK**
    - for Windows, find out case-insensitive match
  - **PYTHONHOME**
    - Alternative module search path
    - It may embedded in 'PYTHONPATH' or 'PYTHONSTARTUP'

Python installation

- On Ubuntu (Python 3)
  - $ sudo apt update
  - $ sudo apt install python3.6
  - $ sudo apt install python3-pip
  - $ pip3 --version
  - $ pip3 list
  - $ pip3 install package_name[==version]
  - $ pip3 install -upgrade package_name
  - $ pip3 uninstall package_name
- On Ubuntu (Python 2)
  - $ sudo apt update
  - $ sudo apt install python-pip
  - $ pip --version
  - $ pip list
  - $ pip install -upgrade package_name
  - $ pip uninstall package_name
- On Raspberry Pi
  - Python 2.6 and python 3.5 are installed by default on Raspbian (not lite version).
  - Check Python using 'python' and 'python3' commands.



  - To install Python3, if not installed yet.

```
$ sudo apt-get install python3
```

Python installation with Virtualenv on Ubuntu

- Virtualenv
  - Python has version dependencies and directory sensitivity
    - To deal with this issue, use virtual environment
      - Virtualenv is a tool to isolate python environment
  - Step 1: Install pip and virtualenv
  - Step 2: Create a virtual environment
  - Step 3: Activate virtualenv

Python installation with Virtualenv on Ubuntu

```
$ sudo apt-get install python-pip python-dev python-virtualenv
$ virtualenv --system-site-packages ~/my_python
$ source ~/my_python/bin/activate
(my_python)$
(my_python)$ python --version
Python 2.7.6
….
….
(my_python)$ deactivate
```

- Install pip and virtualenv
- Create a virtual env
- Activate the virtualenv
- Virtualenv prompt
- Check version of Python

Note your current working directory will be the directory where 'activate' called.

To uninstall Python, simply remove the directory as follows,
$ /bin/rm –rf ~/my_python

Running Python code using Virtualenv

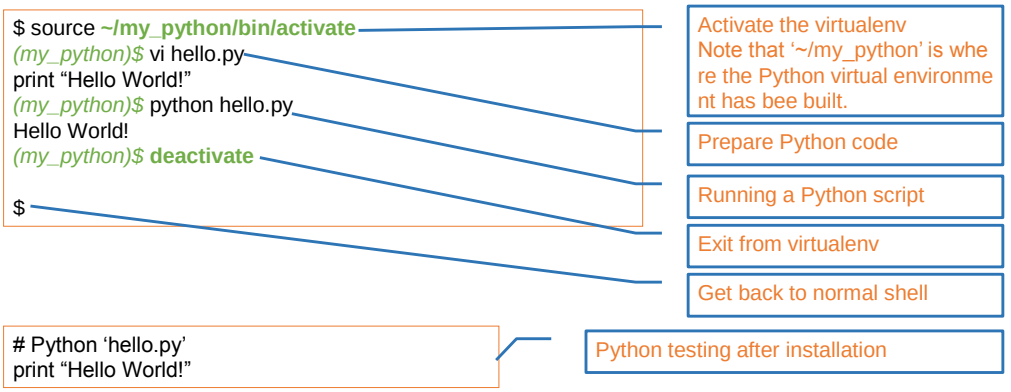


Table of contents

Running Python

- For Unix or Linux
  - $ python
- Windows
  - C:> python
- Exit from Python
  - >>> quit()
- Running script file
  - $ python scrpt.py
  - or
  - $ ./script.py

```
#!/usr/bin/python
print ("Hello, Python!")
```

```
#!/usr/bin/env python3
print ("Hello, Python!")
```

- Python file extensions
  - .py: Python source
  - .pyc: Python compiled byte code with most information
  - .pyo: Python compiled byte code optimized (-O), i.e., optimized .pyc
  - .whl: Python compressed format

- Command line syntax
  - $ python [option] [-c cmd | -m mod | file | -] [args]

- Options
  - -d: It provides debug output.
  - -O: It generates optimized bytecode (resulting in .pyo files).
  - -S: Do not run import site to look for Python paths on startup.
  - -v: verbose output (detailed trace on import statements).
  - -X: disable class-based built-in exceptions (just use strings); obsolete starting with version 1.6.
  - -c cmd: run Python script sent in as cmd string
  - -m mod: importing module
  - file: run Python script from given file

Python

- Comment
  - Comments begin with the hash character ("#") and are terminated by the end of line.
  - Python does not support comments that span more than one line.

- Python is not 'free-format' language, but indentation using whitespace delimits program blocks.
  - There are no block delimiters in Python. Instead, indentation does matter.
    - all the continuous lines indented with same number of spaces would form a block.
    - E.g., C language: {, }

- Python is a case sensitive.

- Python identifiers
  - starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).
  - @, $, and % are not allowed within identifier
  - naming conventions
    - Class names start with an uppercase letter.
    - Identifier starting with '_' means private

Python

- Multi-line statements
  - use the line continuation character (\)

```
total = item_one + \
        item_two + \
        item_three
```

- Statements contained within [], {}, or () brac kets do not need to use \.

```
days = ['Monday', 'Tuesday', 'Wednesday',
        'Thursday', 'Friday']
```

- Multiple statements in a single line using se mi-colon (;)

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

- Quotation
  - single ('), double ("), and triple(''' or """") to de note string literals.
    - triple quotation: multi-line

```
word = 'word'
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

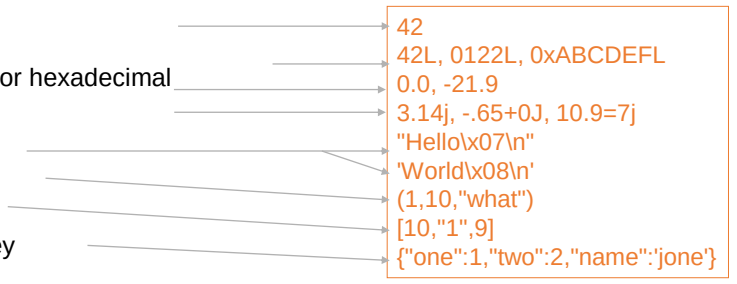
```
print r'C:\\nowhere' # results in C:\\nowhere
print u'Hello' # results in 16-bit Unicode
print b'Hello' # results in 8-bit byte literal
```


Python variable and data types

- Python variables do not need explicit declaration to reserve memory space.
  - The declaration happens automatically when you assign a value to a variable.
    - The equal sign (=) is used to assign values to variables.
  - Variables can be deleted by using 'del'

```
var1 = 1; var2=10
print var1, var2
del var1, var2
```

- Standard data types
  - int: signed, 32-bits
  - long: signed, infinite, octal or hexadecimal
  - float: real value
  - complex: complex number
  - str: string (immutable)
  - tuple: immutable group
  - list: mutable group
  - dict: mutable group with key



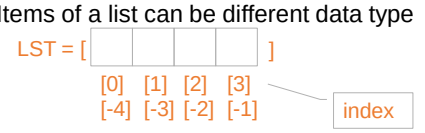
Python strings and lists

- String
  - a contiguous set of characters represented in the quotation marks
  - immutable
  - index 0: starting
  - slice operator: [] and [:]
  - '+' operator: concatenation
  - '*' operator: repetition

```
str = 'Hello World!'

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

- List
  - Ordered collection of data
  - mutable
  - A list contains items separated by commas a nd enclosed within square brackets ([]).
  - Items of a list can be different data type




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

print len(list)  # get the length of a list
print list       # Prints complete list
print list[0]    # Prints first element of the list
print list[1:3]  # from 2nd till 3rd – list[1], list[2]
print list[2:]   # from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```


Python tuple and dictionary

- Tuple
  - Something like *read-only list*, i.e., immutable)
    - ordered list of values.
  - Items separated by comma and enclosed wit hin parentheses (()).
  - Need ',' to differentiate from the mathematica l expression of number
    - y=(2,) ← not number 2 but a tuple containing ' 2'.

```
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
```

- Dictionary
  - Something like *hash table (lookup table)* wit h key-value pairs in unordered fashion
    - ':' for key-value separation
    - duplicate keys are not allowed
    - duplicated value are just fine
  - mutable
  - items separated by commas and enclosed wi thin curly braces ({}).
  - Items of a list can be different data type

```
dict = {}
dict['one'] = "This is one"
dict[2]     = "This is two"
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
```

Python set

- Set
  - Unordered collection of distinct elements
    - same element cannot be added
    - looks like dict without key
  - Items separated by comma and enclosed wit hin parentheses ({}).

```
animals = { 'cat', 'dog'}
print 'cat' in animals      # "True"
animals.add('fish')
print len(animals)          # "3" ← {'cat', 'dog', 'fish'}
animals.add('cat')          # nothing happens since 'cat' exists
print len(animals)          # "3"
animals.remove('cat')
print len(animals)          # "2"
```


Python operator

- Types of operator
  - Arithmetic Operators
    - +, -, *, /, %, **, //
      - **: exponent
      - //: floor division (get integer part of the result)
  - Comparison (Relational) Operators
    - ==, !=, >, <, >=, <=
  - Assignment Operators
    - =, +=, -=, *=, /=, %=, **=, //=
  - Bitwise Operators
    - &, | ^, ~
  - Logical Operators (not &&, ||)
    - 'and', 'or', 'not'
  - Membership Operators
    - 'in', 'not in'
  - Identity Operators
    - 'is'

No unary increment x++, decrement y--.

```
a = 10
b = 20
list = [1, 2, 3, 4, 5 ];

if ( a in list ):
   print "Line 1 - a is available in the given list"
else:
   print "Line 1 - a is not available in the given list"

if ( b not in list ):
   print "Line 2 - b is not available in the given list"
else:
   print "Line 2 - b is available in the given list"
```

```
if ( a is b ):
   print "Line 1 - a and b have same identity"
else:
   print "Line 1 - a and b do not have same identity"

if ( id(a) == id(b) ):
   print "Line 2 - a and b have same identity"
else:
   print "Line 2 - a and b do not have same identity"
```

Python control flow

- Control flow constructs (i.e., compound statements)
  - if elif else
  - for
    - else, break, continue, pass
  - while
    - else, break, continue
  - try except else
- suite
  - a group of individual statement making a single code block
  - one or more lines following a colon (:) after header line
    - header line consists of statement (with the key word) and terminated with a colon.

```
if cond1:
    instruct
    instruct
elif cond2:
    instruct
    instruct
else:
    instruct
    instruct
```

```
for var in set:
    instruct
    instruct
else:
    instruct
```

```
while cond:
    instruct
    instruct
else:
    instruct
```

```
try:
    instr
except exception:
    instruct when exception
else:
    instruct when no exception
```

```
# one-line if
if cond1: instruction
```


Python function

- Function definition
  - def
  - first statement can be optional string
    - docstring
      - >>> print module.__doc__
      - >>> print module.function.__doc__
      - >>> print module.class.__doc__
      - >>> help(module)
      - >>> help(module.function)
      - >>> help(module.class)
  - call-by-object argument passing
    - naming the object without copying
    - without assignment (=), it looks like call-by-referenc
    - with assignment (=), it looks like call-by-value
  - use 'global' statement if required
  - 'return expression' or 'return' or nothing

```
def new_function():
    instruct
```

```
def new_function(arg1):
    "documentation string"
    instruct
    instruct
```

```
def new_function(arg1, arg2):
    "documentation string"
    instruct
    instruct
    return var
```

```
def new_function(arg1, arg2):
    "documentation string"
    instruct
    instruct
    return var1, var2
```

Python modules

- Module
  - Python module is just a file with Python code
- To import modules you use the filename without the .py extensions
  - When imported, the module name is set to filename without .py extension even if it's renamed with ">>> import module as other_name."
- Top-level statements will be executed once even if the file is imported several times even from different files

Python modules

- Module importing
  - 'import' statement enables to use Python source file[s]
    - module is loaded only once, regardless of the number of times it is imported.
    - ➲ >>> import module1[, module2[, … moduleN]]
      - This looks for 'module1.py' in the search path
        - ❖ current directory; PYTHONPATH; /usr/local/lib/python
        - ❖ >>> import sys
        - ❖ >>> print sys.path
    - ➲ Now use any functions in the module as follows
      - >>> module1.func(x, y)
  - Module under a specific directory
    - ➲ >>> import sys
    - ➲ >>> sys.path.insert(0, '***directory***')
    - ➲ >>> import ***module***

import *mymodule*	Brings all elements of mymodule in, but must refer to as *mymodule.*<elem>
import *mymodule* as *my*	Brings all elements of mymodule in, but must refer to as *my.*<elem>
from *mymodule* import *x*	Imports *x* from *mymodule* right into this namespace
from *mymodule* import *	Imports all elements of *mymodule* into this namespace - No need of *mymodule* to refer element of it

- Import only specific attributes from a module into the current namespace
  - >>> from module1 import my_func
  - >>> from module2 import * # import all names

Python __name__ and __main__

- When Python reads a source file,
  - 1. sets a few special variables including __name__, and then
  - 2. executes all of the code found in the file.
- '__name__' will be '__main__' when the module (the source file) as the main program.

```
#my_modA.py
print(__file__)
print(__name__)
if __name__ == '__main__':
  print("Test code here for ", __file__)
```

```
#my_modB.py
import my_modA
print(__file__)
print(__name__)
if __name__ == '__main__':
  print("Test code here for ", __file__)
```

```
Terminal File Edit View Search Terminal Help
[adki@AndoUbuntu] ls
my_modA.py*  my_modB.py*
[adki@AndoUbuntu] python my_modA.py
my_modA.py
__main__
('Test code here for ', 'my_modA.py')
[adki@AndoUbuntu]
```

Note that __name__ will be __main__.

```
adki@AndoUbuntu: ~/work/projects/Python/python-projects/name
[adki@AndoUbuntu] ls
my_modA.py*  my_modB.py*
[adki@AndoUbuntu] python my_modB.py
/home/adki/work/projects/Python/python-projects/name/my_modA.py
my_modA
my_modB.py
__main__
('Test code here for ', 'my_modB.py')
[adki@AndoUbuntu]
```

Note that __name__ of 'my_modA.py' will not be __main__..

Python package

- Package is a namespace for a collection of modules
- Package is the minimal unit of code distribution in Python

packae of 'my_pkg'

my_pkg/ "my_pkg" package

```
+-- __init__.py
+-- myA.py
+-- myB.py
```

module of 'myA'

- Making a package
  - (To make a package, create the directory with __init__.py file
  - 1. make a directory
  - 2. put all modules (i.e., files) to the directory
  - 3. create an empty "__init__.py" file under the directory
- Any directory with __init__.py by convention is a Python package
  - Additional purpose of '__init__.py' is initialization
  - When you import a package, the __init__.py module of the package is executed
- To use module in the package
  - package is a namespace for modules, so you don't import the package itself, you import a module from a package
    - >>> **import** my_pkg.myA
    - >>> **from** my_pkg **import** myA
    - >>> from my_pkg import *

import module from package

Star import

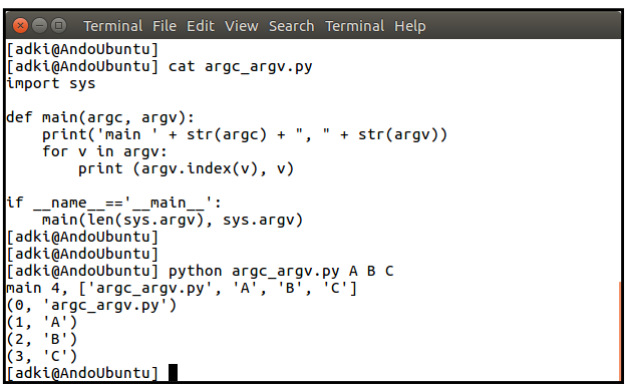
Python search path

- Python order of searching
  - 1. current directory
  - 2. 'sys.path'
    - Usually it contains the current directory
  - 3. 'PYTHONPATH' environment variable
- Python has module search path available at runtime as sys.path.
  - If you run a module as a script file, the containing directory is added to sys.path, otherwise, the current directory is added to it

Python command line arguments

- Python 'sys' module provides access to any command-line arguments via the 'sys.argv'
- There is no 'argc' with Python.
  - >>> import sys
  - >>> len(sys.argv)
  - >>> sys.argv

How to get argc using argv



Python modules

- 'dis' module
  - Python disassembler
  - > import dis
- 'pdb' module
  - Python debugger
  - > import pdb
- 'profile' module
  - Python profiler
  - $ cProfile.py script.py
- 'tabnanny' module
  - ambiguous indentation
  - $ tabnanny.py –v script.py
- 'math', 'sys', 're', 'os', 'os.path', 'logging', 'collections', 'struct', 'decimal', 'datetime', 'time', 'temfile', 'random', 'shutil', 'gob', subprocess', …
- 'os' module
  - system functions
- 'CTypes' module
  - calling the functions of dlls/shared librries
- SciPy package
  - 'NumPy' module
    - N-dim array
  - 'SciPy' module
    - Scientific computing
- 'matplotlib'
  - numerical ploting

Python modules

- 'PIL/Pillow'
- 'IPython'
  - Interactive Python
- Modules for TensorFlow
  - 'numpy'
    - a numerical processing package
  - 'dev'
    - enables adding extensions to Python
  - 'pip'
    - to install and manage certain Python packages
  - 'wheel'
    - manage Python compressed packages in the wheel (.whl) format

Torch and PyTorch
* A tensor library like Numpy with strong GPU support.

References

- Python tutorial
  - http://www.tutorialspoint.com/python/
  - http://www-h.eng.cam.ac.uk/help/languages/python/