Python – An easy to learn interpreter based high-end scripting language

Python 2.x – Legacy and no further development but has life-time support

Python 3.x – current and Feature of the language

**Why 2 versions?**

Creator of python (Guido van Rossum) thought of cleaning up python (Like every developer, after few years they wish to rewrite their framework/program with current updated skill set and make it as Art) and he did that too.

**Which one to use Python 2 or 3?**

It depends, still lot of tools, operating systems supports python 2 and it is safe to use it and python 3 is a feature so there no regret in using that as well.

The best approach would be take any one of the as primary supported and try to write a code which is suitable to other version too. For example, if you take python 3.x as your primary language, try to use the syntax and methods which is also supported in python 2. If any critical situation, give priority to primary supported language.

**Platform Windows or Linux or Mac?**

Python supports all and why to restrict so it’s better to write a code which is compatible with all the platforms.

**IDEs**

There are 1000’s of IDEs out there which supports python and we will see few of them here.

1. VS Code
2. Pycharm
3. Eclipse with PyDev
4. Notepad++ etc.

It’s depending on your choice. I would prefer VS Code or Pycharm.

**Installation of Python**

It is always suggested to use latest version of python and you can have multiple versions of python in your machine. I would suggest to install both Python 2.7.x and python 3.x.x versions and make sure you installing in right path.

My choice would be For Python 2.7.x, it will be C:\Python27 and for Python 3.x.x, it will be C:\Python3x

**What is source control?**

When you work with a team, keeping source code and maintain them in a single common place is more important. There are many tools to do that, starting from MS Visual source, CVS, SVN to today’s leading GIT and Mercury.

GIT Would be the choice of most of the companies but you have to get familiar with what used in your company.

**Let’s talk a little about GIT and GIT Hub**

GIT is a revision control system (Source code management system or Configuration management system – lot of names ☺ ) just a tool which smartly maintain and merge source code and maintain histories. Also helps to avoid confusions when multiple developer works on the same source code.

GIT Hub just a host service which provides free and commercial services to host your projects online. It uses git as its revision control system.

GIT is a big topic, which we can cover based on need and while using it. Let’s go back to Python and environment setup.

**Setting up Environment**

As VS Code is light weight one, let’s start with that,

* Download and install VS code from <https://code.visualstudio.com/>
* Download and install Python 2.7 latest from <https://www.python.org/downloads/> [Make sure you select the installation path.]
* Download and install Python 3.6 latest from <https://www.python.org/downloads/> [Make sure you select the installation path.]
* Download and install GIT from <https://git-scm.com/download> based on your platform

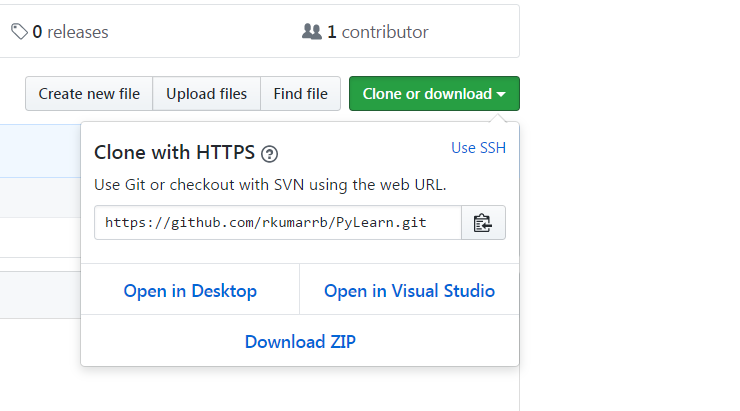
I will walk through basic steps of how to use GIT with VS Code but refereeing below links would give you some idea

<https://code.visualstudio.com/docs/editor/versioncontrol>

We are almost done, just one more thing, cloning GIT Hub project

**Cloning GIT Hub project**

1. In the git hub project website, click on Clone/Download button



1. Copy the URL
2. Open Git bash and type ``git clone <url>``
3. Alternatively we can use any of the GIT client and Source tree is one of the best app for windows. [Source Tree Download URL : [https://www.sourcetreeapp.com/]

We have a place to develop and maintain the code now. So environment setup is done.

**Python**

Python is an interpreter based language hence we don’t need to compile the code to execute and it can be executed interactively

Basic arithmetic operations, string operations are mostly like other languages but little better here. So nothing much to discuss over there.

|  |
| --- |
| ***Note:***  Before to All, one of the most important this is, in python ***“Everything is an Object”,*** and almost everything has attributes and methods. All functions have a built-in attribute \_\_doc\_\_, which returns the doc string defined in the function's source code. The sys module is an object which has (among other things) an attribute called path. And so forth.  In other words, But everything is an object in the sense that it can be assigned to a variable or passed as an argument to a function  *It means even a function or class can be an argument to another function.* |

Whereas data types like List is little important to go through,

**List**

Python is not having arrays instead they have list. String is an ~~array~~ list of characters so that’s the best example for list.

In generally List will be given in comma separated format in python.

Simple list would be like below,

A = [1, 2, 3, 4, 5]

And A[0] would return 1 and A[1] would return 2 and so on. Python also supports ***reverse*** (negative) notation means, A[1] would return last element i.e 5. It is not necessary to get the size of list to get last element.

List also supports the concept of ***slicing,*** (Just like take substring operations in other languages but little better)

***Slicing***

Syntax would be ListVariable[Start:End]

So A[1,3] would return [2,3,4]

Both start and end are optional, if start is not given it takes from first element and if end is not given it goes till last element.

A[3:] would return [4, 5] and A[:2] would return [1, 2, 3]

Even here reverse notation is supported.

A[-3:] would return [3, 4, 5]

***Adding element and Concatenation***

Simple + sign would do concatenation like below

A + [6, 7, 8, 9, 10] would return [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

And there is a function called append which would add element to list.

A.append(6) would be [1, 2, 3, 4, 5, 6]

When you use list in append it would result as nested list

A.append([6, 7]) would result [1, 2, 3, 4, 5, [6, 7]]

To find the length of the list, can use len(listvariable). That’s all about basic of List we should know about it.

*Note: String is immutable list and list is mutable.*

**Control Flows**

Like any other languages python also supports if-else, for-break-continue, while conditions.

For loop alone is little different which works only with Range of elements or objects. It is not following traditional initialization, condition, increment format of for loop(C, Java, C# languages using this style)

There is a small variable called pass which is simply do nothing.

There isn’t much new with Python so let’s move on.

**Functions**

Functions in the pythons are generally declared with keyword **def** and code block starts with colon(:) and indented.

***Note: In Python everything is an “object”. It means that function, class and everything can be assigned to a variable and utilized like a normal object.***

*Normal function definition would be like,*

|  |
| --- |
| def functionname (arguments):  codeblock\_1  codeblock\_2  ……. |

Python function arguments can be mentioned with representation of positional or keyword arguments.

For example,

|  |
| --- |
| def addnumbers(num1, num2, num3, num4=0):  return num1 + num2 + num3 + num4 |

In the above function, num1 to 3 are mandatory arguments and num4 is an optional argument. Return statement will return the value.

The above function can be called like below,

*Way 1: Positional*

addnumbers(1, 2, 3, 4) and it would return 10.

*Way 2: Keywords*

addnumbers(num1=1, num3=2, num2=3, 4) in this example, if you noticed, num 3 and num2 are not in the same order as its declared, with keyword call its possible.

*Way 3: using list or sequence*

argslist = [1, 2, 3, 4]

addnumbers(\*argslist) it’s just like positional but arguments can be sent using list.

*Way 4: using keywords*

argsdict = {num1:1, num2:2, num3:3, num4:4}

addnumbers(\*\*argsdict) it’s just like keyword method but arguments can be shared with dictionary.

Functional definition we can also use variable number of arguments using list (\*) or keywords (\*\*)

***Arguments as list***

For Example:

|  |
| --- |
| def addnumbers(\*nums):  s = 0  for n in nums:  s = s + n  return s |

and it can be called like,

addnumbers(1, 2, 3, 4) would return 10

addnumbers(1, 2, 3, 4, 5, 6, 7) would return 28 and likewise any number of arguments can be given.

***Arguments as dictionary (keyword arguments)***

For Example:

|  |
| --- |
| def addnumbers(\*\*kwargs):  return kwargs[“num1”] + kwargs[“num2”] + kwargs[“num3”] + kwargs[“num4”] |

And it can be called like,

addnumbers(num1=1, num2=1, num3=3, num4=4)

***Return and Yield statement***

In python single or multiple values can be returned,

Return single value,

|  |
| --- |
| def fun():  return 1 |

Return multiple value,

|  |
| --- |
| def fun():  return 1, 2, 3 |

and it can be received as,

a, b, c, d = fun()

and if some of the return values wanted to be skipped, underscore can be used.

\_, b, c, \_ = fun()

Yield is like return but it return a generator, for example

|  |
| --- |
| def fun():  yield 1  yield 2  yield 3 |

the above function return 3 values 1, 2 and 3

***Creating function alias***

Function can be assigned to a variable

for example, the above function can be assigned to a variable called that as a function

alterfun = fun

alterfun() would also return 1, 2, and 3.

**Class:**

Class can be created using the keyword “class”. Class is a blueprint of an object contains methods (actions) and properties.

***\_\_init\_\_ and \_\_del\_\_***

Class can contain any number of functions including special functions like \_\_init\_\_ and \_\_del\_\_. Init is a constructor and whereas del is destructor.

Init will be executed whenever the class is initiated and del will be executed whenever object of the class is closed.

***self***

Yet to write.