14/6/19 Unit-1 Introduction

History of Python:

→ Python is a general purpose interpreted, interactive, object oriented and high level programming language.

→ It was created by Guido Van Rossum who was a Dutch person from Netherlands in the year 1991.

→ He started implementing Python from 1989 and finished and released the first working version of python in 1991.

→ He created it when he was working in National Research Institute of mathematics and Computer science.

→ He named it has "python" being a big fan of 'Monty Python's Flying Circus' comedy show boroadcasted in BBC from 1969 to 1974.

→ It was derived from many other languages including ABC (All Basic Codes), MODULA-3, C, C++, ALGOL-68, Small talk and Unix Shell etc.

Python Version List:

Python Version	Released Date
Python 1.0	Jan 1994
Python 1.5	Dec 31st, 1997

Python 1.6	September 5th, 2000
Python 2.0	October 16th, 2000
Python 2.1	April 17th, 2001
Python 2.2	December 21st, 2001
Python 2.3	July 29th, 2003
Python 2.4	November 30th, 2004
Python 2.5	September 19th, 2006
Python 2.6	October 1st, 2008
Python 2.7	July 3rd, 2010
Python 3.0	December 3rd, 2008
Python 3.1	June 27th, 2009
Python 3.2	February 20th, 2011
Python 3.3	September 29th, 2012
Python 3.4	March 16th, 2014
Python 3.5	September 13th, 2015
Python 3.6	December 23rd, 2016.
Python 3.7	June 27th, 2018

Need of Python Programming:

The following are the factors to use Python

1→ Software Quality: Python code is designed to be readable and hence reusable and maintainable.

2→ Developer Productivity: Python boosts developer productivity many times beyond compiled or statically typed languages such as C, C++ and JAVA. Python code is typically 1/3rd to 1/5th the size of equivalent C, C++, JAVA programs.

3→ Program Portability; Most python programs run unchanged on all major computer platforms. Porting python code between Linux, Windows and Mac

4→ Support Libraries; So Python comes with a large collection of prebuilt and portable functionality known as Standard library. This library supports an array of application level programming task from text pattern matching to network scripting.

5→ Component Integration: Python scripts can easily communicate with other parts of an application, using a variety of integration mechanism.

Such Integrations allow python to be used as a product customization and extension tool.

Today python code can import C and C++ libraries.

6 → Enjoyment : It is easy to use. It gives programmer flexibility to write programs

Applications of Python Programming:

Python Programming most common applications are as follows:

1. System programming
2. GUI (Graphical User Interface)
3. Internet scripting
4. Component Integrating
5. Database programming
6. Rapid Prototyping
7. Numeric and Scientific Programming
8. Gaming, Images, Datamining
9. Robotics
10. Excel Sheets
11. Artificial Intelligence etc.

→ System Programming : Python's built-in interfaces to operating system services make it ideal for writing portable, maintainable system-administration tools and utilities (sometimes called shell tools). Python program can search files and directory trees, launch other programs, do parallel processing with processes and threads and so on.

→ GUIs: Python simplicity and rapid turnaround also make it a good match for graphical user interface programming on the desktop. Python comes with a standard object-oriented interface to the TK GUI API called tkinter (Tkinter in 2.x) that allows Python programs to implement portable GUIs with a native look and feel. Python / tkinter GUIs run unchanged on Microsoft Windows, X Windows (on unix and Linux) and the Mac OS (both classic and OS X)

→ Internet scripting: Python comes with standard Internet modules that allow Python programs to perform a wide variety of networking tasks, in client and server modes. Scripts can communicate over sockets, extract form information sent to server-side CGI (common gateway interface) scripts, transfer files by FTP (FTP) file transfer protocol, parse and generate XML and JSON documents, send, receive, compose, and parse email, and fetch web pages by URLs (uniform resource locator).

→ Component Integration: Python scripts can easily communicate with other parts of an application, using a variety of integration mechanisms. Such integrations allow Python to be used as a product customization and extension tool. Today, Python code can invoke C and C++ libraries

→ Database Programming: For traditional database demands, there are Python interfaces to all commonly used relational database systems - Sybase, Oracle, Informix, ODBC, MySQL, PostgreSQL

SQLite, and more. The Python world has also defined a portable database API for accessing SQL database systems from Python scripts, which looks the same on a variety of underlying database systems.

→ Rapid Prototyping: To python programs, components written in Python and C look the same. Because of this, its possible to prototype systems in Python initially and then move selected components to a compiled language such as C or C++ for delivery. Unlike some prototyping tools, Python doesn't require a complete rewrite once the prototype has solidified. It increases efficiency and reduce time and cost.

→ Numeric and Scientific Programming: Python has a module called "NumPy" to support all numerical programming. Python also has "SciPy" to support all the scientific for both numeric and scientific applications.

Using NumPy, a developer can perform the following operations.

1. Mathematical and logical operations on arrays.
2. Fourier transforms and routines for shape manipulation
3. Operations related to linear algebra. NumPy has inbuilt functions for linear algebra and number generation.

→ Gaming, Images, Data Mining, Robots, Excel....

Python is commonly applied in more domains than can be covered here. For Example, you'll find tools that allow you to use Python to do.

→ Game programming and multimedia with pygame, cgkit, pyglet, PySoy, Panda3D and others
→ Serial port communication on Windows, linux and more with the PySerial extension
→ Image processing with PIL and its newer Pillow fork, PyOpenGL, Blender, Maya and more
→ Robot control programming with the PyRo toolkit
→ Natural language analysis with the NLTK package
→ Instrumentation on the Raspberry Pi and Arduino boards.
→ Mobile computing with ports of Python to the Google Android and Apple iOS platforms
→ Excel spreadsheet function and macro programming with the PyXLL or DataNitro add-ins
→ Media file content and metadata tag processing with PyMedia, ID3, PIL/Pillow and more
→ Artificial Intelligence with the PyBrain neural net library and the Milk machine learning toolkit
→ Expert system programming with PyCLIPS, PyKe, Pyrolog and pyDatalog

→ Network monitoring with Zenoss, written in and customized with Python
→ Python-scripted design and modeling with PythonCAD, PythonOCC, FreeCAD, and others
→ Document processing and generation with Report Lab, Sphinx, Cheetah, PyPDF, and so on
→ Data visualisation with Mayavi, matplotlib, VTK, VPython and more
→ XML parsing with the xml library package, the xmlrpclib module, and third-party extensions
→ JSON and CSV file processing with the json and csv modules

Features of Python Programming:

1. Open Source: Python is available easily open source software. Anyone can use source code that doesnot cost anything.
2. Easy to Learn: Python is a popular scripting language, clear and easy syntax, No type declarations, automatic memory management, high level datatypes and operations, design to read and write fast
3. High Level Language: High level language refers to the higher level of concept from machine language (Assembly language) Python is an example of high level language like 'c, C++, Perl and JAVA

with low-level optimisation (Reduce or compress)

4. Portable: High level languages are portable which means they are able to run across all major hardware and software platforms with few (or) no change in source code. Python is portable and can be used on Windows, Linux, Solaris, free BSD, Amiga, OS/2 and many more.

5. Object Oriented: Python is a full featured object oriented programming language, with features such as classes, objects, inheritance and overloading.

6. Interactive: Python ~~is an in~~ has an interactive console where you get a python prompt and interact with the interpreter directly to write and test our programs. This is useful for mathematical programming.

7. Interpreted: Python programs are interpreted, takes source code as input and then compiles each statement and executes it immediately. No need to compiling or linking.

8. Extendable: Python Interpreter is easily extended and can add a new built in function. Modules written in C, C++, JAVA code

9. Libraries: Database, web services, networking, 3D Graphics etc, graphical user interface.

10. Supports: Support from online python community.

→ Keywords or Reserve Words in Python

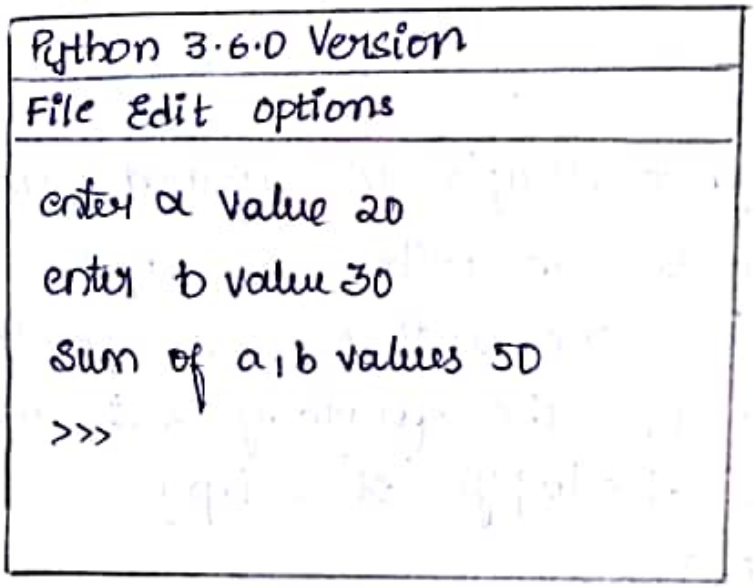
The following identifiers are used as reserve words and cannot be used as ordinary identifiers

false	class	finally	is	return
none	continue	for	lambda	try
true	def	from	nonlocal	while
and	del	global	not	with
as	el	if	or	yield
assert	else	import	pass	
break	except	in	raise	

Basics of python programming using REPL shell:

REPL (Read-Eval-Print-Loop) is a simple interactive computer programming environment The term REPL is usually used to refer to a ~~list~~ LISP (List processing) interactive environment but can be applied to command line shells and similar environments for programming languages like python, Ruby etc.

Python IDLE Interactive Mode: Python IDLE in later comes with an interactive interpreter. When we type python in our shell or command prompt the python interpreter becomes active with >>> prompt and waits for our commands.



Now we can type any valid python expression at the prompt (>>>). Python reads the typed expression, evaluates it and prints the result

Example: >>>print ("Hello")
Hello
>>>

Running Python scripts in IDLE:

Go to file menu, click on new file (Ctrl+N)

Write the code and save as extension.py

Program:

```
a= int (input ("enter a value"))
b = int (input ("enter b value"))
```

```
c = a+b
print ("Sum of a,b values is", c)
```

→ Run the program by pressing F5 (or)

Run → Run Module

Output: Enter a value 10

Enter b value 20

Sum of a, b values is 30

Running Python scripts in command prompt:

Before going to run python run python folder in the command prompt open the text editor. Type the following text and save it as hello.py 'or' hai.py

```
print ("Hello")
```

In the command prompt run this program by calling python. hi.py

Make sure that you change the directory where you saved the file before doing it.

```
C:\>

C:\>users>admin>cd desktop

C:\>users>admin>desktop>Python hello.py

hello.
```

Variables in Python:

A variable is a memory location where a programmer can store a value

Example: roll_no, amount, etc.

Value is either string, numeric etc

Example: "Ram", 25, 43.7

→ Variables are created when first assigned. Variables must be assigned before being reference. The values stored in a variable can be accessed or updated later

→ No declaration required. The type (int, float, string etc) of the variable is determined by python.

→ The interpreter allocates memory on the basis of the datatype of a variable

Rules of a Variable:

1. Must being with an alphabet / letter (a to z. or A to z or underscore)
2. Other characters can be letters/numbers or underscore
3. K sensitive
4. Can be reasonable length
5. There are some reserved words which you cannot use as a variable name because

Python Assignments:

The assignment statement creates new variables and gives them values.

Basic Assignment statement in python is

Syntax:

<variable> = <expression>

where we equal to sign is used to assign the value (right side) to a variable name (left side)

Example: a = 20

Print (a)

O/P :- 20

Example: a = "welcome"

Print(a)

O/P :- welcome

Example: a = 20

Print ("a")

O/P :- a

Multiple Assignments:

The basic assignment statement works for a single variable and a single expression. We can also assign a single value to more than one variables.

Syntax: <var1> = <var2> = <var3> = ... = <varn>

= <expre>

Example: x = y = z = 1

```
Print (x)
Print (y)
Print (z)
```

Olp:
```
1
1
1
```

```
x, y, z = 1, 2, "abcd"
Print (x)
Print (y)
Print (z)
```

Program: Write a python programming swapping of two numbers

```
>>> x=20
>>> y=30
>>> print (x)
    20
>>> print (y)
    30
>>> x, y = y, x
>>> Print (x)
    30
>>> print (y)
    20
```

24

Input / Output:

Input functions: The input function in python programming is input()

Syntax: input ([prompt])

where prompt is the string we wish to display on the screen. It is optional.

Example: a = input ("enter a value")

Here we can see that the entered value 10 is a string not a number.

To convert this into a number we can use int() or float()

Example: a = int (input ("enter a value"))

Output functions: We use print() to output data to the standard output device (screen)

Example: print ("hello") , o/p:- Hello.

Example:

```
a=5
print ("The value of a is", a)
```

o/p: The value of a is 5

Syntax:

```
print (* objects, sep = ' ', end = '\n',
       file = sys.stdout, flush = false)
```

Here object is the values to be printed

sep separator is used between the values. It defaults into a space character

→ After all values are printed, end is printed. It defaults into a new line

→ The file is the object where the values are printed and its default value is sys.stdout (screen)

Example:

```
Print (1,2,3,4, sep='*')
```

o/p = 1*2*3*4

Example:

```
Print (1,2,3,4 sep='#,' end='&')
```

o/p: 1#2#3#4&

Sometimes we would like to format our output to make it look attractive. This can be done by using str.format() ~~function~~ method

Example: ~~x=5 ; y=~~

```
x = 5
y = 10
Print ("The value of x is {}
        and the value of y is {}".
        format (x, y))
```

o/p: The value of x is 5 and the value of y is 10

Here the curly braces are used as place holders. We can specify the order in which it is printed by using numbers

Example: Print ("I love {0} and {1}".
format ('bread', 'butter'))

Output: I love bread and butter

Example: Print ("I love {1} and {0}".
format ('bread', 'butter'))

Output: I love butter and ~~br~~ bread

Example: Print ("Hello {name}, {greeting}".
format (greeting = 'good morning',
name = 'John'))

Output: Hello ram, good morning

Example: x = 12.345678
Print ('The value of x is %.3.2f', %x)

Output: The value of x is 12.345 or 12.35

25/6 **Indentation:**

Python uses white space (spaces and tabs) to define program blocks whereas other languages like C, C++, use braces ({ }) to indicate blocks of codes for class, functions or flow control.

→ The number of white spaces in the indentation is not fixed but all statements within the block must be the indented same amount.

→ In the following program the block statements have no indentation

Example:
```
a=5
b=3
if (a>b):
print ("a is greater")
else:
print ("b is greater")
```

O/P: Syntax error (or) Unexpected Indent

Correction:
```
a=5
b=3
if(a>b):
  print("a is greater")
else:
  print ("b is greater")
```

O/P: a is greater

Multiple Statements in a Single Line:

The semicolon (;) allows multiple statements on a single line given that neither statement starts a new code block

Example: a=5; b=3; c=a+b; print c

Comments in python

→ A hash (#) sign i.e not inside a string literal begins a comment

→ All characters after the (#) and upto the end of the physical line (or) part of the comment and the python interpreter ignores them

Example: #This is a single line comment

→ Following triple quoted string is also ignored by python interpreter and can be used as a multiline comment

Example:

```
"""
This is multiline comment where
we can use the multiline
comment used in python"""
```

Multiline statements:

Statements in python typically end with a new line. Python allow the use of the line continuation character (\) to denote that the line should continue

Example:
```
Total = item-one \
      + item_two \
      + item_three
```

Statements contained within the [ ], { }, ( ) donot need to use the line continuation character.

Example:
```
days = ['Mon', 'Tue', 'Wed', 'Thur',
        'fri', 'sat']
```