**Python**

**Introduction:**

1. Python is a programming language.
2. It is high level programming language. (Programmer friendly and not machine friendly)
3. It is general purpose programming language. (used to develop programs for variety use)

**Who and when created Python:**

****

**Guido Van Rossum** – This person introduced top easy simple programming language. It is called as python language.

He was working in NRI institute in Netherland.

Official python released in 20 Feb 1991.

Java came in 1995

Python came in 1989

Python is old language than java language

**Why name is python**:

**Guido Van Rossum** was big fan of comedy show named **“The complete Monty Python & Flying Circus”.** From this name python word is taken.

**Why python is popular nowadays?**

Current Market trending requirement changes time to time. Now,

1. Everyone talks about simple language and easy quickly to understand
2. Write less code (concise code)
3. For ML, AI, IoT, NN python is used.
4. Library support
5. Data Type headache not in python

**Prerequisite for python:**

It is easy programming language. If we want to learn python, we don’t need any programming background.

**Python flexibility:**

Dynamically Typed programming language.

**All rounder Features of Python:**

C – Functional programming language

Java - OOP

Perl – Scripting language (group of lines executed one by one)

Python borrowed–

1. Functional programming features from C language
2. OOP features from C++
3. Scripting language from Perl
4. Modular programming (Out dated) features from Modula3

So, Python is all-rounder with above features.

Most of syntax of python taken from C and ABC languages.

**Where We can use Python:**

1. Desktop application
2. Web application
3. Networking application(Client server, Chat)
4. Games application
5. Data analysis application
6. ML, AI, NN, IoT application

**Which software companies uses Python:**



**Features of Python:**

1. **Simple and Easy to learn** – less words and reader friendly.

For Ex. Condition check

Java –> int a = (2>3)?5:6;

Python –> x = 5 if 2>3 else 6

Read file from disk

Python –> print(open(‘FILE\_NAME.txt’).read())

1. **Open source** - Freely available(No license fee)

PSF – python software foundation – charitable trust is maintaining python

1. **High level programming language** –
   * 1. easily learn, write and understand.
     2. No need to take care for low level activities like security, garbage collection, memory management etc.
2. **Platform Independent** – PVM (Python virtual machine) for each platform. Similar to java, Write once, run anywhere.
3. **Portability** - The ability of software to be transferred from one machine or system to another.
4. **Dynamically Typed –**  No Statically typed restriction for variables
5. **Procedure & object oriented**
6. **Interpreted –** No compilation required
7. **Extensible –** use of other language non-python code in python code
8. **Embedded –** Use of python code in any other languages. E.g. use of python is Jython
9. **Extensive Library support –** existing functionality given as module. Developer no need to implement functionality.

**Limitations of Python:**

1. Mobile application development – not suitable – no library supports as of now
2. Enterprise application development –not suitable as lots of things like messaging security etc
3. Python is interpreted – performance wise not up to the mark as compared to other language. To improve performance, Now JIT(executed as group of lines) is added. For performance, use PyPy.

**Python Flavors:**

1. Cython – Python with C language support
2. Jython – Python with java language support libraries
3. Iron Python – Python with C#. .NET support
4. Ruby - Python with Ruby language support
5. Anaconda Python- When we work with large volumes of data like ML, data science. All libraries available with Anaconda Python.
6. Stackless – for concurrent application
7. PyPy – Python for speed. Uses JIT inbuilt.

**Python Versions:**

1. Python 0.90 – Feb 20th 1991
2. Python 1.0 – Jan 1994
3. Python 2.0 – October 2000 – Py2K
4. Python 3.0 – December 2008 – Py3k –more powerful as new market requirement.

From 2020 onwards no python support for python 2.0.

**Python Identifiers:**

Name is python program is identifier. Class name, variable or method name used for identification purpose

For ex. x=2. Here x is identifier

**Rules to define python identifiers:**

1. Only [a-z, A-Z, 0-9, \_] allowed, otherwise syntax error. For ex. Sachin#1, Sachin&1 are invalid identifiers
2. Identifier should not start with digit. For ex. 1Sachin is invalid identifier.
3. Identifiers are case sensitive. For ex. Sachin and Sachin are not equal identifiers
4. No length limit for python identifier.
5. Reserved keywords are not used to define identifiers. For Ex. if=10 is invalid

**Reserved Keywords in Python:**

Python has total 33 reserved keywords. Python is very small programming language. Every reserved keyword represents some meaning.

Contains only alphabets

Except 3 words(True, False, None), all are small case

switch, do-while not available in python

int, char, float not available in Python

Below is list of reserved keywords in python:

>>> import keyword

>>> keyword.kwlist

['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

More details for given keywords:

**print**

print to console

**while**

controlling the flow of the program

**for**

iterate over items of a collection in order that they appear

**break**

interrupt the (loop) cycle, if needed

**continue**

used to interrupt the current cycle, without jumping out of the whole cycle.

New cycle will begin.

**if**

used to determine, which statements are going to be executed.

**elif**

stands for else if.If the first test evaluates to False,

then it continues with the next one

**else**

is optional. The statement after the else keyword is executed,

unless the condition is True

**is**

tests for object identity

**not**

negates a boolean value

**and**

all conditions in a boolean expression must be met

**or**

at least one condition must be met.

**import**

import other modules into a Python script

**as**

if we want to give a module a different alias

**from**

for importing a specific variable, class or a function from a module

**def**

used to create a new user defined function

**return**

exits the function and returns a value

**lambda**

creates a new anonymous function

**global**

access variables defined outside functions

**try**

specifies exception handlers

**except**

catches the exception and executes codes

**finally**

is always executed in the end. Used to clean up resources.

**raise**

create a user defined exception

**del**

deletes objects

**pass**

does nothing

**assert**

used for debugging purposes

**class**

used to create new user defined objects

**exec**

executes Python code dynamically

**yield**

is used with generators

**Data Types in Python**

Python is dynamically typed programming language. So, not required to specify type explicitly. Type will be assigned automatically as per given value to the variable. Data type means Type of data stored in variable.

For ex.

>>> a=5

>>> print(type(a))

<class 'int'>

>>> b=5.5

>>> print(type(b))

<class 'float'>

>>> x='Sachin'

>>> print(type(x))

<class 'str'>

**Numeric Types:**int, float, complex

**Sequence Types:**list, tuple, range

**Text Sequence Type:**str

**Binary Sequence Types:**bytes, bytearray, memoryview

**Set Types:**set, frozenset

**Mapping Types: —**dict

**Other Built-in Types:**

Modules, Classes and Class Instances, Functions, Methods, Code Objects, Type Objects, the Null Object (None), the Ellipsis Object, the NotImplemented Object, Boolean Values (True and False), Internal Objects.

Everything in python is object.

**Q. How to know address of the object?**

>>> x=5

>>> id(x)

140704347509920

.

**Data Types in details:**

int, float, complex, bool, str are fundamental data types in python.

1. **int**

Numbers without decimal point.

For ex. 10, 100 are int in python

Note – No long data type in Python 3. It is available in python 2.

**int type available in below forms:**

**Decimal** (base 10) – allowed digits are 0-9. For ex. 10, 120 etc.

**Binary** (base 2) – allowed digits 0 and 1 – for ex. 5 -> **0b**0101,

>>> x=0b1010

>>> print(x)

10

**Octal** (base 8) – allowed values from 0-7. For ex. 0o111

>>> x=0o111

>>> print(x)

73

**Hexadecimal** (Base 16) – Allowed values from 0-9 and A-F. For ex. 0xABC

>>> x=0xABC

>>> print(x)

2748

Base conversion functions:

Python provides in built function for base conversions:

**bin()** – to convert number to binary format.

For ex.

>>> bin(10)

'0b1010'

>>> bin(0x111)

'0b100010001'

>>> bin(0o111)

'0b1001001'

**hex()** – to convert number to hexadecimal

>>> hex(10)

'0xa'

>>> hex(0o111)

'0x49'

**oct()** -to convet number to octal

>>> oct(10)

'0o12'

>>> oct(0x111)

'0o421'

1. **float**

The number with decimal point is float number. Floating values we represent only in decimal form. Octal, binary, hexadecimal forms not allowed for float types.

For ex. x= 111.11, x=1.3e4(Scientific float values)

1. **Complex number**

The number in the form (a+bj) is a real number.

Where a – real part (either int or float values)

b – imaginary part (in decimal form only)

For ex. **x= 11+5j**

>>> x=11+5j

>>> x.real

11.0

>>> x.imag

5.0

1. **perform arithmetic (+,-,\*,/)operation on complex numbers**

x=2+3j and y=4+5j

>>> **x=2+3j**

>>> **y=4+5j**

>>> x+y

(6+8j)

>>> x-y

(-2-2j)

>>> x\*y

(-7+22j)

>>> x/y

(0.5609756097560976+0.0487804878048781j)

1. **bool**

Allowed values True and False.

For ex. a=4, b=5

>>> a=4

>>> b=5

>>> c=a>b

>>> print(c)

False

>>> type(c)

<class 'bool'>

True internally represented as 1 and False as 0.

For ex.

>>> print(True+True)

2

>>> print(True+False+True+True)

3

1. **str**

We can use single or double quotes is considered as String type. In Python, no char type. It is represented as str type only

For ex. x=”Sachin”

>>> x="Sachin"

>>> type(x)

<class 'str'>

1. **Triple quote**(single or double) used to define multiline string literals. It is used to use single and double quotes as a normal characters in string. It is used for doc string.

>>> x='''sachin

... rane

... pune'''

>>> print(x)

sachin

rane

pune

1. **Index**

>>> x='sachin'

>>> print(x[1])

a

>>> print(x[6]) # when index value more than expected one

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: string index out of range

Python supports positive and negative(reverse direction) index

>>> print(x[-1])

n

1. **Slice operator**

Use to extract substring from given string. Returns substring with begin index and end\_index-1

For ex. x=’sachin rane’

>>> x='sachin rane'

>>> x[1:4]

'ach'

>>> x[2:6]

'chin'

>>> x[2:] #when end index not provided, it gives whole string from given begin index

'chin rane'

>>> x[2:20] #when end index is big than string length, it gives whole string from given begin index

'chin rane'

>>> x[:6] #when begin index not provided, it gives sub string from 0 index to end index-1

'sachin'

>>> x[:] #When no begin and end index provided, it gives whole string

'sachin rane'

>>> x[6:3] #when end index lower than begin index, it returns empty string

''

**Convert ‘sachin rane’ to ‘Sachin Rane’ using slice operator**

>>> x='sachin rane'

>>> result = x[0].**upper()**+x[1:7]+x[7].**upper()**+x[8:]

>>> print(result)

Sachin Rane

**Skip last 1 char from given string**

>>> print(x[:**len(x)**-1])

sachi

**Convert first and last char capital letters for given string**

>>> x='sachin'

>>> print(x[0].upper()+x[1:len(x)-1]+x[len(x)-1:].upper())

SachiN

**Multiply string with given number of times**

>>> print('sachin'\*5)

Sachinsachinsachinsachinsachin

**Type casting in python**

Convert one type to another type

int() – to convert other types to int **excluding complex**

float() - to convert other types to float **excluding complex**

complex() - to convert other types to complex

bool() - to convert other types to bool

str() - to convert other types to str

**For ex. for int conversion**

>>> print(int(5.5))

5

>>> int(True)

1

>>> int(False)

0

>>> int('5')

5

>>> int('5a') #When no integral value of string

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '5a'

>>> int('0b0101') #when value provided is not of base 10

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

ValueError: invalid literal for int() with base 10: '0b0101'

**For float conversion**

>>> print(float(5))

5.0

>>> print(float(0b0101))

5.0

>>> print(float(0xabc))

2748.0

>>> print(float(True))

1.0

>>> print(float(False))

0.0

>>> print(float('10.5'))

10.5

**For complex conversion:**

>>> complex(10)

(10+0j)

>>> complex(10.5)

(10.5+0j)

>>> complex(0b111)

(7+0j)

>>> complex(0x111)

(273+0j)

>>> complex(0o111)

(73+0j)

>>> complex('10')

(10+0j)

>>> complex('10.5')

(10.5+0j)

>>> complex(10,11)

(10+11j)

>>> complex(10.5,11.5)

(10.5+11.5j)

>>> complex('10','11')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: complex() can't take second arg if first is a string

>>> complex(10,'11')

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: complex() second arg can't be a string

>>> complex('10',11)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: complex() can't take second arg if first is a string

>>> complex(True)

(1+0j)

>>> complex(False)

0j

**For bool conversion:**

If argument to bool() is zero ‘0’ then result is False. For all non zero argument, result is True.

Same applies for float as well.

If string argument to bool() is empty then result is False otherwise result is True.

>>> bool(0)

False

>>> bool(0.0)

False

>>> bool(0.01)

True

>>> bool(-1)

True

>>> bool(1.1)

True

>>> bool(0+0j)

False

>>> bool(2+3j)

True

>>> bool(2+0j)

True

>>> bool("2")

True

>>> bool("0")

True

>>> bool("")

False

>>> bool("False")

True

>>> bool("True")

True

**For str conversion:**

>>> str(5)

'5'

>>> str(5.5)

'5.5'

>>> str('5')

'5'

>>> str(5+6j)

'(5+6j)'

>>> str(True)

'True'

>>> str(False)

'False'

>>> str(0b111)

'7'

**Fundamental data types and immutability:**

Immutability – Not changeable.

Once object created, we can’t do any changes in that object

**If we try to modify the object, new object will be created.**

>>> x=5

>>> id(x)

140704347509920

>>> x=x+1 # changing x reference from 5 to 6. 5 will be eligible for gc if no reference available.

>>> id(x)

140704347509952

**If both variables points to same object and we modify one variable reference, new object will be created for modified variable.**

>>> x=5

>>> y=x # both x and y points to same object 5.

>>> y=x+1 # changing y reference to 6

>>> print(x) # still x value is same.

5

>>> print(y) # y value changes and refers to new object created.

6

>>> id(x) # x refers to address of 5

140704347509920

>>> id(y) # y refers to address of 6

140704347509952

**Advantages of Immutability of objects in python:**

1. Memory utilization
2. Performance – python checks if object is already available in memory. If not available, then create new one. This avoids costly object create operations. Searching object in memory is less complex than creating new object in memory.

**Collection Data types:**

Fundamental data types – int, float, bool, complex, str – they only hold one value at a time.

Collection data type – list - hold multiple values (group of objects)

Below are the collection data types in python:

list – [10,20,30]

tuple – (10,20,30)

Set – {10,20,30}

dict – {111:’Sachin’, 222:’Rane’}

1. **list**

insertion ordered group of heterogeneous objects where duplicate objects allowed.

Indexing and slicing is applicable for list.

List is automatically growing(append()) and shrinking(remove()).

List is mutable

For ex.

>>> l=[10, 'sachin',20,30]

>>> type(l)

<class 'list'>

**Access elements of list :**

**Get first element of list**

>>> l=[10, 'sachin',20,30]

>>> l[0]

10

Get last element of list

>>> l[-1]

30

Get any element of list for given index

>>> l[2]

20

If index provided more than list length-1

>>> l[4]

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IndexError: list index out of range

Slice of elements of list

>>> l[1:3]

['sachin', 20]

Empty list

>>>l=[]

Add element to list

>>> l=[]

>>> l.append(10)

>>> l.append(100)

>>> print(l)

[10, 100]

Remove element from list

>>> l.remove(10)

>>> print(l)

[100]

Mutable (change element values)

>>> l=[10,20,30]

>>> l[0]=222

>>> print(l)

[222, 20, 30]

1. **Tuple**

Read only version of list. It is immutable in nature.

For ex.

>>> t=(10,20,30)

>>> type(t)

<class 'tuple'>

Tuple object does not support object assignment

>>> t[0]=1212

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: 'tuple' object does not support item assignment

Tuple object does not support append and remove operations.

>>> t.append(1000)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'tuple' object has no attribute 'append'

>>> t.remove(3232)

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

AttributeError: 'tuple' object has no attribute 'remove'

Empty tuple

>>>t=()

Single int valued tuple

>>> t=(10)

>>> type(t)

<class 'int'>

To overcome this problem and to tell PVM as this is tuple and not type of int, we need to add comma after first element.

>>> type(t)

<class 'tuple'>

1. **Set**

Group of objects with single entity where duplicates not allowed and order not important.