Comments:

* Comments are used to write description about application logics to understand the logics easily.
* The main objective comment is code maintenance will become easy.
* The comments are non-executable code.

There are two types of comments in python

Single line comment: Write the description in single line and it starts with #

Multiline comments: Write the description in more than one line starts with

“”” ends with : “”””(triple quotes)

In Python while writing the comments we can write triple double quotes or triple single quotes (“””) or (‘’’)

Python style rules and conventions:

Statement termination: Python does not use any symbol to terminate a statement. When you end a physical code-line by pressing Enter key, the statement is considered terminated by default.

Maximum line length: Line length should be maximum 79 characters.

Lines and indentation: Blocks of code are denoted by line indentation, which is enforced through 4 spaces per indentation level.

# Data type: -

* Data type represents the type of data present inside a variable.
* In Python, we are not required to specify the type explicitly. Based on value provided, the type will be assigned automatically. Hence Python is dynamically typed language.

Python has 14 data types.

int, float, complex, bool, str, bytes, bytearray, range, list, tuple, set, frozenset, dict, None.

Data type related functions:

1. type(variable): - It is used to identify the data type.

>>>a=10

>>>type(a)

<class ‘int’>

1. id(variable): - To print address of variable.

>>>id(a)

18268367 (real address)

Note: - In Python everything is treated as object only.

# int data type: -

It is used to accept whole numbers (integral values).

It does not have a decimal point.

Ex: a=10

type(a)

#<class ‘int’>

In Python, integral values can be 4 types:

1. Decimal form (base 10)
2. Binary form (base 2)
3. Octal form (base 8)
4. Hexa decimal form (base 16)

**Decimal form:** - It is the default number system in Python. The allowed digits are 0 to 9.

a=106 a=100

**Binary form:** - The allowed digits are 0 and 1. If you provide binary number then number must be prefixed with ‘0b’ or ‘0B’. a=0B1111 a=0b101

**Octal form:** - The allowed digits are 0 to 7. If you provide octal number then number must be prefixed with ‘0o’ or ‘0O’. a=0o124 a=0O156

**Hexadecimal form:** - The allowed digits are 0 to 9, A to F (a to f). If you provide hexadecimal number then number must be prefixed with ‘0x’ or ‘0X’.

a=0xA1 a=0X2B1 Python always provide the value decimal only.

Ex:

>>>a=120

>>>a

120

>>>a=0b101

>>>a

5

>>>a=0x10A

>>>a

257

Base Conversions: -

**bin():** - This function is used to convert other integer formats to binary format.

Ex:

>>>bin(15)

0b1111

>>>bin(0o11)

0b1001

>>>bin(0xA)

0b1010

**oct():** - This function is used to convert other integer formats to octal format.

Ex:

>>>oct(10)

0o12

>>>oct(0b1111)

0o17

>>>oct(0x123)

0o443

**hex():** - This function is used to convert other integer formats to hexadecimal format.

Ex:

>>>hex(100)

0x64

>>>hex(0b111111)

0x3f

>>>hex(0o12345)

0x14e5

float data type:

It allows fractional numbers.

>>>a=12.34

>>>type(a)

<class ‘float’>

We can provide values exponent form also.

>>>f=1.2e3 (1.2\*103=1.2\*1000=1200.0)

>>>f

1200.0

complex data type:

Complex number in python is made up of two floating point values, one is real part and another is imaginary part. For accessing different parts of variable (object) x; we will use x.real and x.image. Imaginary part of the number is represented by "j" instead of "i", so 1+0j denotes zero imaginary part.

A complex number is of the form: a+bj

a🡪real part

b🡪imaginary part jj2=-1 j=sqrt(-1)

a and b contain integers or floating point values.

Ex:

>>>a=10+20j

>>>type(a)

<class ‘complex’>

>>>a.real

10

>>>a.imag

20

>>>10+5.5j

>>>2.4+3.6j

* In the real part if we use int value then we can specify that either by decimal, binary, octal or hexa decimal form.
* But imaginary part should be specified only by using decimal form.

>>>a=0b11+5j

>>>a

(3+5j)

>>>a=3+0b11j

SyntaxError: Invalid syntax

We can perform operations on complex type values.

>>>a=10+1.5j

>>>b=20+2.5j

>>>c=a+b

>>>c

(30+4j)

>>>type(c)

<class ‘complex’>

We can use complex type generally in scientific applications and electrical engineering applications.

bool data type: - It represent Boolean values. This data type allows only True or False. Internally Python represents as True=1 and False=0.

>>>b=True

>>>type(b)

<class ‘bool’>

>>>a=10

>>>b=20

>>>a>b

False

>>>True+True

2

str data type: - To represent the sequence of characters. String values enclosed in single quotes or double quotes.

>>>x=’gangadhar’

>>>type(x)

<class ‘str’>

>>>x=”lokesh”

>>>type(x)

<class ‘str’>

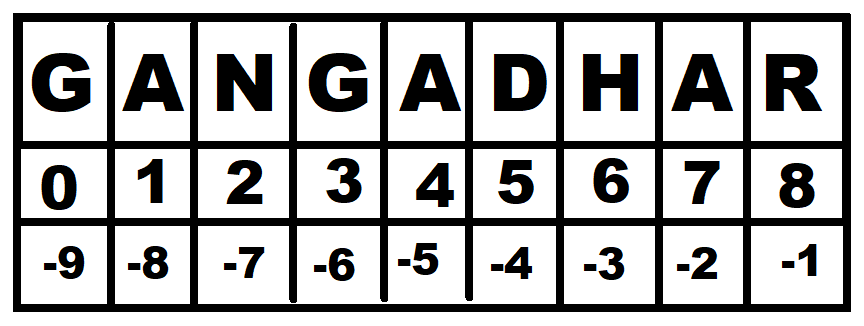
Also allows ''' ……. ''' or """ ……. """ to define multi line string literals.

>>>x="The ‘Python’ Course” >>>x=’The “Python” Course’

>>>x=’’’The ‘Python’ is developed by “Rossam” in 1989’’’

Slicing of strings:

* Slice means a piece.
* [] operator is called slice operator, which can be used to retrieve parts of string.
* In Python strings follows zero based index.
* The index can be either +ve or –ve.
* +ve index means forward direction from left to right.
* -ve index means backward direction from right to left.



Ex : s=”GANGADHAR”

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Command | Output | Command | Output | Command | Output |
| s[0] | G | s[-1] | R | S[-2] | A |
| S[1:4] | ANG | s[4:] | ADHAR | S[:4] | GANG |
| s[:] | GANGADHAR | s[::1] start:end:step | GANGADHAR | s[::2] | GNAHR |
| s[::-1] | RAHDAGNAG | 2\*s | GANGADHARGANGADHAR | 3\*s | GANGADHARGANGADHARGANGADHAR |

These are fundamental data types in Python.

# Type Conversions:

We can convert one type of value to another type. This conversion is called Type casting or Type conversion.

In Python, pre-defined functions to perform type casting.

* **int(): - To convert any other type to int except complex type.**

>>>int(123.45)

123

>>>int(True)

1

>>>int(‘12’)

12

If we want to convert str type to int, compulsory str should contain only integral value and should be specified in base 10.

>>>int("ten")

ValueError: invalid literal for int() with base 10: ‘ten’

>>>int("10.5")

ValueError: invalid literal for int() with base 10: ’10.5’

>>>int(10+5j)

TypeError: can’t convert complex to int

* **float(): - To convert any other type to float except complex type.**

>>>float(123)

123.0

>>>float(True)

1.0

>>>float(’12.34’)

12.34

If we want to convert str type to float, compulsory str should be either integral or floating point literal and should be specified in base 10.

>>>int("ten")

ValueError: could not convert string to float: ‘ten’

>>>int(10+5j)

TypeError: can’t convert complex to float

* **complex(): - To convert any other type to complex type.**

Form1: complex(x)

We can use this function to convert x into complex number with real part x and imaginary part 0.

>>>complex(10)

10+0j

>>>complex(True)

1+0j

>>>complex(12.5)

12.5+0j

>>>complex(‘10’)

10+0j

>>>complex(“ten”)

ValueError: complex() arg is malformed string.

Form2: complex(x,y)

We can use this method to convert x and y into complex number such that x will be real part and y will be imaginary part.

>>>complex(10,20)

10+20j

>>>complex(10.5,20.5)

10.5+20.5j

>>>complex(True,False)

1+0j

* **bool(): - To convert any other type to bool type.**

If int type 0 is False, non-zero is True.

>>>bool(0)

False

>>>int(122)

True

If float type, if total number evaluates 0 it returns False, otherwise it returns True.

>>>bool(12.34)

True

>>>bool(0.000)

False

>>>bool(0.123)

True

If complex type, if both real and imaginary part is 0 it returns False, otherwise it returns True.

>>>bool(0+0j)

False

>>>bool(10+20j)

True

>>>bool(0+10j)

True

If str type, if empty string then it becomes False otherwise it returns True.

>>>bool(‘gangadhar’)

True

>>>bool(‘’)

False

* **str(): - To convert any other type to str type.**

>>>str(10)

‘10’

>>>str(True)

‘True’

>>>str(12.34)

’12.34’

>>>str(0b1111)

‘15’

>>>str(10+5j)

‘10+5j’

# Fundamental data types vs immutability

* All fundamental data types are immutable. i.e once we create an object, we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behavior is called immutability.
* Every programming language object creation is very costly operation.
* In Python if a new object is required, then PVM won’t create object immeadiately. First it will check if any object available with the required content or not. If available then existing object will be reused. If it is not available then only a new object will be created. The advantage of this approach is memory utilization and performance will be improved.
* But the problem in this approach is, several references pointing to the same object, by using one reference if we are allowed to change the content in the existing object then the remaining references will be effected. To prevent this immutability concept is required. According to this once creates an object we are not allowed to change content. If we are trying to change with those changes a new object will be created.

>>>a1=10 a1

a3

10

>>>a2=10

>>>a3=10 a2

>>>a4=10

>>>id(a1)

>>>id(a2)

>>>id(a3)

>>>id(a4)

>>>a1 is a2(It returns True both objects refers same address otherwise False) True

Immutability means we cannot change the existing object. If you want to change the existing value then a new object will be created. complex and float data types does not support object reusability.

int type reusability objects can be available in range 0 to 256.

>>>a=256

>>>b=256

>>>a is b

True

>>>a1=257

>>>a2=257

>>>a1 is a2

False

float type, every time a new object will be created.

>>>a1=10.5

>>>a2=10.5

>>>a1 is a2

False

complex type, every time a new object will be created.

>>>a1=10+20j

>>>a2=10+20j

>>>a1 is a2

False

bool type is always supports reusability

>>>a1=True

>>>a2=True

>>>a1 is a2

True

str type is always supports reusability.

>>>a1=’gangadhar’

>>>a2=’gangadhar’

>>>a1 is a2

True

# list data type: -

If you want to represent a group of objects in a single entity then we use list type.

Insertion order is preserved.

Duplicates are allowed.

Heterogeneous objects are allowed.

Growable in nature.

Elements should be enclosed within ‘[ ]’ brackets.

Elements can be accessed through index.

It is mutable.

Ex:

>>>l=[10,20,30,40]

>>>print(l)

[10,20,30,40]

>>>type(l)

<class ‘list’>

>>>l=[10,10.5,’satish’,10]

>>>l[0]

10

>>>l[3]

10

>>>l[0:2]

[10,10.5,’satish’]

Ex:

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

>>>l.append('gangadhar')

>>>l

[10,20,30,’gangadhar’]

>>>l.remove(20)

>>>l

[10,30,’gangadhar’]

>>>l2=l\*2

>>>l2

[10,30,’gangadhar’,10,30,’gangadhar’]

# tuple data type: -

It is exactly same as list type, but it is immutable.

It is read only version of list. It is enclosed in paratheses ().

x=(10,20,30)

>>>t=(10,20,30,40)

>>>print(t)

(10,20,30,40)

>>>type(t)

<class ‘tuple’>

>>>t=(10,10.5,’gangadhar’,10)

>>>t[0]

10

>>>t[3]

10

>>>t[0:2]

[10,10.5,’gangadhar’]

>>>t[0]=100

TypeError: ‘tuple’ object does not support item assignment

>>>t.append(‘babu’)

AttributeError: ‘tuple’ object has no attribute ‘apppend’

>>>t.remove(10)

AttributeError: ‘tuple’ object has no attribute ‘remove’

# set data type: -

If you want to represent a group of objects in a single entity where insertion order not preserved, duplicates are not allowed.

Heterogeneous objects are allowed.

Growable in nature.

It is mutable.

It is enclosed in { }.

Ex:

>>>x={10,20,’gangadhar’}

>>>print(x)

>>>type(x)

<class ‘set’>

>>>x.add(50.6)

Will be added in random place.

>>>x

>>>x.remove(20)

>>>x

Index is not applicable.

Slice operator is not available.

>>>x[0]

TypeError: ‘set’ object does not support indexing.

# frozenset data type: -

It is exactly same as set type, but it is immutable.

Ex:

>>>x={10,20,’gangadhar’}

>>>type(x)

<class ‘set’>

>>>id(x)

23456789

>>>a=frozenset(x)

>>>id(a)

2346089

>>>x.add(50.6)

>>>id(x)

23456789

>>>a=frozenset(x)

>>>id(a)

246092

>>>type(a)

<class ‘frozenset’>

>>>a.add(70)

AttributeError: ‘frozenset’ object has no attribute ‘add’

>>>a.remove(10)

AttributeError: ‘frozenset’ object has no attribute ‘remove’

# dict data type: -

If you want to represent a group of key-value pairs then we use dict data type.

Values can be duplicated but key does not allow duplicates.

Ex:

>>>d={101:’gangadhar’,102:’lokesh’,103:’rajesh’}

>>>type(d)

<class ‘dict’>

>>>d[104]=’gangadhar’

>>>d

{101:’gangadhar’,102:’lokesh’,103:’rajesh’,104:’gangadhar’}

>>>d[103]=’lohit’

>>>d

{101:’gangadhar’,102:’lokesh’,103:’lohit’,104:’gangadhar’}

* **range data type: - It represents a sequence of numbers. It is immutable.**

range(number)

0 to number -1

Ex:

>>>r=range(10)

>>>type(r)

<class ‘range’>

>>>r[0]

0

>>>r[-1]

9

>>>r[2:8]

range(2,8)

>>>for i in r(2:8): print (i)

2

3

4

5

6

7

range(begin,end)  begin to end-1 range(10,20)  10 to 19

>>>for i in range(10,20):

print(i)

10 11 12 13 14 15 16 17 18 19

range(begin,end,step)

begin to end-1 change by step value range(1,101,5)  1,6,11,16…..

>>>for i in range(0,101,5):

print(i)

# None data type: -

None means nothing or No value associated. If the value is not available, then to handle such type of cases None introduced.

This object eligible for garbage collector.

>>>a=10

>>>a=None

>>>type(a)

<class ‘None Type’>