Language Fundamentals

Introduction

- Python is a general purpose high level programming language.
- Python was developed by Guido Van Rossam in 1989 while working at National Research Institute at Netherlands.
- But officially Python was made available to public in 1991. The official Date of Birth for Python is : Feb 20th 1991.
- Python is recommended as first programming language for beginners.

Eg1: To print Helloworld:

Java:

```
1) public class HelloWorld
2) {
3)   p s v main(String[] args)
4)   {
5)      SOP("Hello world");
6)   }
7) }
```

C:

```
1) #include<stdio.h>
2) void main()
3) {
4)   print("Hello world");
5) }
```

Python:

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

Eg2: To print the sum of 2 numbers

Java:

```
1)  public class Add
2)  {
3)     public static void main(String[] args)
4)     {
5)        int a,b;
6)        a =10;
7)        b=20;
8)        System.out.println("The Sum:"+(a+b));
9)     }
10) }
```

C:

```
1)  #include <stdio.h>
2)
3)  void main()
4)  {
5)     int a,b;
6)     a =10;
7)     b=20;
8)     printf("The Sum:%d",(a+b));
9)  }
```

Python:

```
1)  a=10
2)  b=20
3)  print("The Sum:",(a+b))
```

The name Python was selected from the TV Show
"The Complete
Monty
Python's
Circus", which was broadcasted in BBC from 1969 to 1974.

Guido developed Python language by taking almost all programming features from different languages

1. Functional Programming Features from C
2. Object Oriented Programming Features from C++
3. Scripting Language Features from Perl and Shell Script

4. Modular Programming Features from Modula-3

Most of syntax in Python Derived from C and ABC languages.

Where we can use Python:

We can use everywhere. The most common important application areas are

1. For developing Desktop Applications
2. For developing web Applications
3. For developing database Applications
4. For Network Programming
5. For developing games
6. For Data Analysis Applications
7. For Machine Learning
8. For developing Artificial Intelligence Applications
9. For IOT

...

Note:

Internally Google and Youtube use Python coding
NASA and Nework Stock Exchange Applications developed by Python.
Top Software companies like Google, Microsoft, IBM, Yahoo using Python.

Features of Python:

1. Simple and easy to learn:

Python is a simple programming language. When we read Python program,we can feel like reading english statements.
The syntaxes are very simple and only 30+ kerywords are available.
When compared with other languages, we can write programs with very less number of lines. Hence more readability and simplicity.
We can reduce development and cost of the project.

2. Freeware and Open Source:

We can use Python software without any licence and it is freeware.

Its source code is open,so that we can we can customize based on our requirement.

Eg: Jython is customized version of Python to work with Java Applications.

3. High Level Programming language:

Python is high level programming language and hence it is programmer friendly language. Being a programmer we are not required to concentrate low level activities like memory management and security etc..

4. Platform Independent:

Once we write a Python program,it can run on any platform without rewriting once again. Internally PVM is responsible to convert into machine understandable form.

5. Portability:

Python programs are portable. ie we can migrate from one platform to another platform very easily. Python programs will provide same results on any paltform.

6. Dynamically Typed:

In Python we are not required to declare type for variables. Whenever we are assigning the value, based on value, type will be allocated automatically.Hence Python is considered as dynamically typed language.

But Java, C etc are Statically Typed Languages b'z we have to provide type at the beginning only.

This dynamic typing nature will provide more flexibility to the programmer.

7. Both Procedure Oriented and Object Oriented:

Python language supports both Procedure oriented (like C, pascal etc) and object oriented (like C++,Java) features. Hence we can get benefits of both like security and reusability etc

8. Interpreted:

We are not required to compile Python programs explcitly. Internally Python interpreter will take care that compilation.

If compilation fails interpreter raised syntax errors. Once compilation success then PVM (Python Virtual Machine) is responsible to execute.

9. Extensible:

We can use other language programs in Python.
The main advantages of this approach are:

1. We can use already existing legacy non-Python code
2. We can improve performance of the application

10. Embedded:

We can use Python programs in any other language programs.
I.e we can embedd Python programs anywhere.

11. Extensive Library:

Python has a rich inbuilt library.
Being a programmer we can use this library directly and we are not responsible to implement the functionality.

etc...

Limitations of Python:

1. Performance wise not up to the mark b'z it is interpreted language.
2. Not using for mobile Applications

Flavors of Python:

1.CPython:
It is the standard flavor of Python. It can be used to work with C lanugage Applications

2. Jython or JPython:
It is for Java Applications. It can run on JVM

3. IronPython:
It is for C#.Net platform

4.PyPy:
The main advantage of PyPy is performance will be improved because JIT compiler is available inside PVM.

5.RubyPython
For Ruby Platforms

6. AnacondaPython
It is specially designed for handling large volume of data processing.

...

Python Versions:

Python 1.0V introduced in Jan 1994
Python 2.0V introduced in October 2000
Python 3.0V introduced in December 2008

Note: Python 3 won't provide backward compatibility to Python2
I.e there is no guarantee that Python2 programs will run in Python3.

Current versions

Python 3.6.1 **Python 2.7.13**

Identifiers

A name in Python program is called identifier.
It can be class name or function name or module name or variable name.

a = 10

Rules to define identifiers in Python:

1. The only allowed characters in Python are

- alphabet symbols(either lower case or upper case)
- digits(0 to 9)
- underscore symbol(_)

By mistake if we are using any other symbol like $ then we will get syntax error.

- cash = 10 ✓
- ca$h =20 ✗

2. Identifier should not starts with digit

- 123total ✗
- total123 ✓

3. Identifiers are case sensitive. Of course Python language is case sensitive language.

- total=10
- TOTAL=999
- print(total) #10
- print(TOTAL) #999

Identifier:

1. Alphabet Symbols (Either Upper case OR Lower case)

2. If Identifier is start with Underscore (_) then it indicates it is private.

3. Identifier should not start with Digits.

4. Identifiers are case sensitive.

5. We cannot use reserved words as identifiers
   Eg: def=10 ✗

6. There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.

7. Dollor ($) Symbol is not allowed in Python.

Q. Which of the following are valid Python identifiers?

1) 123total ✗
2) total123 ✓
3) java2share ✓
4) ca$h ✗
5) _abc_abc_ ✓
6) def ✗
7) if ✗

Note:

1. If identifier starts with _ symbol then it indicates that it is private
2. If identifier starts with __(two under score symbols) indicating that strongly private identifier.
3. If the identifier starts and ends with two underscore symbols then the identifier is language defined special name,which is also known as magic methods.

Eg: __add__

Reserved Words

In Python some words are reserved to represent some meaning or functionality. Such type of words are called Reserved words.

There are 33 reserved words available in Python.

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

Note:

1. All Reserved words in Python contain only alphabet symbols.

2. Except the following 3 reserved words, all contain only lower case alphabet symbols.

- True
- False
- None

Eg: a= true ✗
a=True ✓

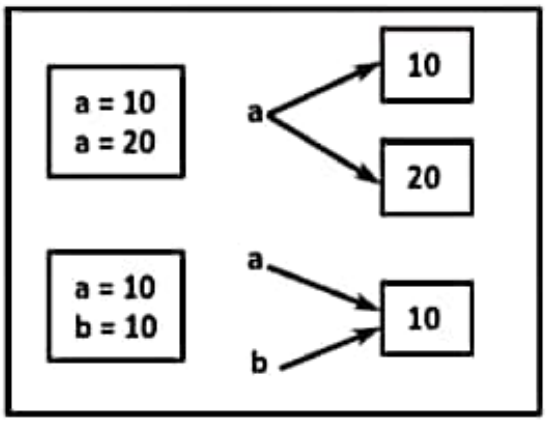
```
>>> import keyword
>>> keyword.kwlist
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else',
'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or',
'pass', 'raise', 'return', 'try', 'while',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

Data Types

Data Type represent 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 contains the following inbuilt data types

1. int
2. float
3.complex
4.bool
5.str
6.bytes
7.bytearray
8.range
9.list
10.tuple
11.set
12.frozenset
13.dict
14.None



Note: Python contains several inbuilt functions

1.type()
to check the type of variable

2. id()
to get address of object

3. print()
to print the value

In Python everything is object

int data type:

We can use int data type to represent whole numbers (integral values)

Eg:

```
a=10
type(a) #int
```

Note:

In Python2 we have long data type to represent very large integral values.
But in Python3 there is no long type explicitly and we can represent long values also by using int type only.

We can represent int values in the following ways

1. Decimal form
2. Binary form
3. Octal form
4. Hexa decimal form

1. Decimal form(base-10):

It is the default number system in Python
The allowed digits are: 0 to 9
Eg: a =10

2. Binary form(Base-2):

The allowed digits are : 0 & 1
Literal value should be prefixed with 0b or 0B

Eg: a = 0B1111
a =0B123
a=b111

3. Octal Form(Base-8):

The allowed digits are : 0 to 7
Literal value should be prefixed with 0o or 0O.

Eg: a=0o123
a=0o786

4. Hexa Decimal Form(Base-16):

The allowed digits are : 0 to 9, a-f (both lower and upper cases are allowed)
Literal value should be prefixed with 0x or 0X

Eg:
a =0XFACE
a=0XBeef
a =0XBeer

Note: Being a programmer we can specify literal values in decimal, binary, octal and hexa decimal forms. But PVM will always provide values only in decimal form.

```
a=10
b=0o10
c=0X10
d=0B10
print(a)10
print(b)8
print(c)16
print(d)2
```

Base Conversions

Python provide the following in-built functions for base conversions

1.bin():

We can use bin() to convert from any base to binary

Eg:

```
1)    >>> bin(15)
2)  '0b1111'
3)  >>> bin(0o11)
4)  '0b1001'
5)  >>> bin(0X10)
6)  '0b10000'
```

2. oct():

We can use oct() to convert from any base to octal

Eg:

```
1) >>> oct(10)
2) '0o12'
3) >>> oct(0B1111)
4) '0o17'
5) >>> oct(0X123)
6) '0o443'
```

3. hex():

We can use hex() to convert from any base to hexa decimal

Eg:

```
1) >>> hex(100)
2) '0x64'
3) >>> hex(0B111111)
4) '0x3f'
5) >>> hex(0o12345)
6) '0x14e5'
```

float data type:

We can use float data type to represent floating point values (decimal values)

Eg: f=1.234
type(f) float

We can also represent floating point values by using exponential form (scientific notation)

Eg: f=1.2e3
print(f) 1200.0
instead of 'e' we can use 'E'

The main advantage of exponential form is we can represent big values in less memory.

***Note:
We can represent int values in decimal, binary, octal and hexa decimal forms. But we can represent float values only by using decimal form.

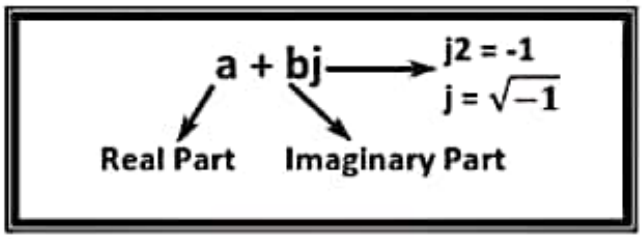
Eg:

```
1)  >>> f=0B11.01
2) File "<stdin>", line 1
3)   f=0B11.01
```

```
4)          ^
5)  SyntaxError: invalid syntax
6)
7)  >>> f=0o123.456
8)   SyntaxError: invalid syntax
9)
10) >>> f=0X123.456
11)  SyntaxError: invalid syntax
```

Complex Data Type:

A complex number is of the form



a and b contain intergers or floating point values

Eg:

3+5j
10+5.5j
0.5+0.1j

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

```
1)  >>> a=0B11+5j
2)  >>> a
3)  (3+5j)
4)  >>> a=3+0B11j
5)   SyntaxError: invalid syntax
```

Even we can perform operations on complex type values.

```
1)  >>> a=10+1.5j
2)  >>> b=20+2.5j
3)  >>> c=a+b
4)  >>> print(c)
5)  (30+4j)
6)  >>> type(c)
7)  <class 'complex'>
```

Note: Complex data type has some inbuilt attributes to retrieve the real part and imaginary part

```
c=10.5+3.6j

c.real==>10.5
c.imag==>3.6
```

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

4.bool data type:

We can use this data type to represent boolean values.
The only allowed values for this data type are:
True and False

Internally Python represents True as 1 and False as 0

```
b=True
type(b)  =>bool
```

Eg:

```
a=10
b=20
c=a<b
print(c)==>True


True+True==>2
True-False==>1
```

str type:

str represents String data type.

A String is a sequence of characters enclosed within single quotes or double quotes.

```
s1='durga'
s1="durga"
```

By using single quotes or double quotes we cannot represent multi line string literals.

```
s1="durga
```

soft"

For this requirement we should go for triple single quotes(''') or triple double quotes(""")

```
s1='''durga
      soft'''

s1="""durga
      soft"""
```

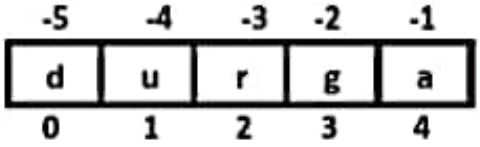
We can also use triple quotes to use single quote or double quote in our String.

''' This is " character'''
' This i " Character '
We can embed one string in another string

'''This "Python class very helpful" for java students'''

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




```
1) >>> s="durga"
2) >>> s[0]
3) 'd'
4) >>> s[1]
5) 'u'
6) >>> s[-1]
7) 'a'
8) >>> s[40]
```

IndexError: string index out of range

```
1)  >>> s[1:40]
2)  'urga'
3)  >>> s[1:]
4)  'urga'
5)  >>> s[:4]
6)  'durg'
7)  >>> s[:]
8)  'durga'
9)  >>>
10)
11)
12) >>> s*3
13) 'durgadurgadurga'
14)
15) >>> len(s)
16) 5
```

Note:

1. In Python the following data types are considered as Fundamental Data types

- int
- float
- complex
- bool
- str

2. In Python,we can represent char values also by using str type and explicitly char type is not available.

Eg:

```
1)  >>> c='a'
2)  >>> type(c)
3)  <class 'str'>
```

3. long Data Type is available in Python2 but not in Python3. In Python3 long values also we can represent by using int type only.

4. In Python we can present char Value also by using str Type and explicitly char Type is not available.

Type Casting

We can convert one type value to another type. This conversion is called Typecasting or Type coersion.
The following are various inbuilt functions for type casting.

1. int()
2. float()
3. complex()
4. bool()
5. str()

1.int():

We can use this function to convert values from other types to int

Eg:

```
1)  >>> int(123.987)
2)  123
3)  >>> int(10+5j)
4)  TypeError: can't convert complex to int
5)  >>> int(True)
6)  1
7)  >>> int(False)
8)  0
9)  >>> int("10")
10) 10
11) >>> int("10.5")
12) ValueError: invalid literal for int() with base 10: '10.5'
13) >>> int("ten")
14) ValueError: invalid literal for int() with base 10: 'ten'
15) >>> int("0B1111")
16) ValueError: invalid literal for int() with base 10: '0B1111'
```

Note:

1. We can convert from any type to int except complex type.
2. If we want to convert str type to int type, compulsary str should contain only integral value and should be specified in base-10

2. float():

We can use float() function to convert other type values to float type.

```
1)  >>> float(10)
2)  10.0
3)  >>> float(10+5j)
4)  TypeError: can't convert complex to float
5)  >>> float(True)
6)  1.0
7)  >>> float(False)
8)  0.0
9)  >>> float("10")
10) 10.0
11) >>> float("10.5")
12) 10.5
13) >>> float("ten")
14) ValueError: could not convert string to float: 'ten'
15) >>> float("0B1111")
16) ValueError: could not convert string to float: '0B1111'
```

Note:

1. We can convert any type value to float type except complex type.

2. Whenever we are trying to convert str type to float type compulsary str should be either integral or floating point literal and should be specified only in base-10.

3.complex():

We can use complex() function to convert other types to complex type.

Form-1: complex(x)

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

Eg:

```
1)  complex(10)==>10+0j
2)  complex(10.5)===>10.5+0j
3)  complex(True)==>1+0j
4)  complex(False)==>0j
5)  complex("10")==>10+0j
6)  complex("10.5")==>10.5+0j
7)  complex("ten")
8)      ValueError: complex() arg is a malformed string
```

Form-2: 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.

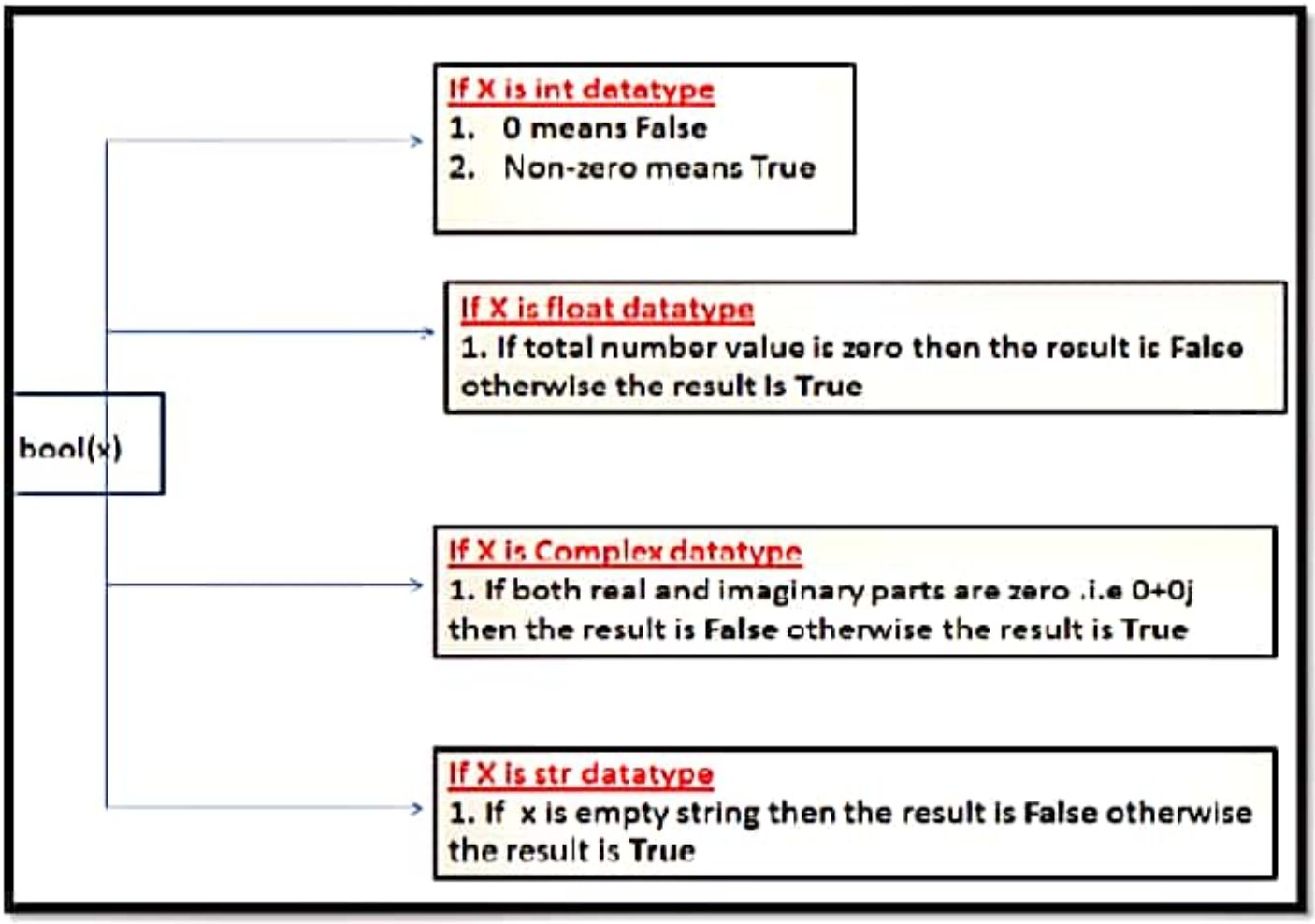
Eg: complex(10,-2)==>10-2j
complex(True,False)==>1+0j

4. bool():

We can use this function to convert other type values to bool type.

Eg:

```
1) bool(0)==>False
2) bool(1)==>True
3) bool(10)===>True
4) bool(10.5)===>True
5) bool(0.178)==>True
6) bool(0.0)==>False
7) bool(10-2j)==>True
8) bool(0+1.5j)==>True
9) bool(0+0j)==>False
10) bool("True")==>True
11) bool("False")==>True
12) bool("")==>False
```



5. str():

We can use this method to convert other type values to str type

Eg:

```
1) >>> str(10)
2) '10'
3) >>> str(10.5)
4) '10.5'
5) >>> str(10+5j)
6) '(10+5j)'
7) >>> str(True)
8) 'True'
```

Fundamental Data Types vs Immutability:

All Fundamental Data types are immutable. i.e once we creates 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-chageable behaviour is called immutability.

In Python if a new object is required, then PVM wont create object immediately. First it will check is 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.

Eg:

```
1) >>> a=10
2) >>> b=10
3) >>> a is b
4) True
5) >>> id(a)
6) 1572353952
7) >>> id(b)
8) 1572353952
9) >>>
```

```
>>> a=10
>>> b=10
>>> id(a)
1572353952
>>> id(b)
1572353952
>>> a is b
True
```

```
>>> a=10+5j
>>> b=10+5j
>>> a is b
False
>>> id(a)
15980256
>>> id(b)
15979944
```

```
>>> a=True
>>> b=True
>>> a is b
True
>>> id(a)
1572172624
>>> id(b)
1572172624
```

```
>>> a='durga'
>>> b='durga'
>>> a is b
True
>>> id(a)
16378848
>>> id(b)
16378848
```

bytes Data Type:

bytes data type represens a group of byte numbers just like an array.

Eg:

```
1)  x = [10,20,30,40]
2)      b = bytes(x)
3)      type(b)==>bytes
4)      print(b[0])==> 10
5)      print(b[-1])==> 40
6)      >>> for i in b : print(i)
7)
8)        10
9)        20
10)       30
11)       40
```

Conclusion 1:

The only allowed values for byte data type are 0 to 256. By mistake if we are trying to provide any other values then we will get value error.

Conclusion 2:

Once we creates bytes data type value, we cannot change its values,otherwise we will get TypeError.

Eg:

```
1)  >>> x=[10,20,30,40]
2)  >>> b=bytes(x)
3)  >>> b[0]=100
4)   TypeError: 'bytes' object does not support item assignment
```

bytearray Data type:

bytearray is exactly same as bytes data type except that its elements can be modified.

Eg 1:

```
1)  x=[10,20,30,40]
2)  b = bytearray(x)
3)  for i in b : print(i)
4)  10
```

```
5)  20
6)  30
7)  40
8)   b[0]=100
9)   for i in b: print(i)
10) 100
11) 20
12) 30
13) 40
```

Eg 2:

```
1)  >>> x =[10,256]
2)  >>> b = bytearray(x)
3)  ValueError: byte must be in range(0, 256)
```

list data type:

If we want to represent a group of values as a single entity where insertion order required to preserve and duplicates are allowed then we should go for list data type.

1. Insertion order is preserved
2. heterogeneous objects are allowed
3. duplicates are allowed
4. Growable in nature
5. values should be enclosed within square brackets.

Eg:

```
1)  list=[10,10.5,'durga',True,10]
2)       print(list) # [10,10.5,'durga',True,10]
```

Eg:

```
1)  list=[10,20,30,40]
2)  >>> list[0]
3)  10
4)  >>> list[-1]
5)  40
6)  >>> list[1:3]
7)  [20, 30]
8)  >>> list[0]=100
9)  >>> for i in list:print(i)
10) ...
11) 100
12) 20
13) 30
```

```
14) 40
```

list is growable in nature. i.e based on our requirement we can increase or decrease the size.

```
1)  >>> list=[10,20,30]
2)  >>> list.append("durga")
3)  >>> list
4)  [10, 20, 30, 'durga']
5)  >>> list.remove(20)
6)  >>> list
7)  [10, 30, 'durga']
8)  >>> list2=list*2
9)  >>> list2
10) [10, 30, 'durga', 10, 30, 'durga']
```

Note: An ordered, mutable, heterogenous collection of eleemnts is nothing but list, where duplicates also allowed.

tuple data type:

tuple data type is exactly same as list data type except that it is immutable.i.e we cannot chage values.

Tuple elements can be represented within parenthesis.

Eg:

```
1)  t=(10,20,30,40)
2)  type(t)
3)  <class 'tuple'>
4)  t[0]=100
5)  TypeError: 'tuple' object does not support item assignment
6)  >>> t.append("durga")
7)  AttributeError: 'tuple' object has no attribute 'append'
8)  >>> t.remove(10)
9)  AttributeError: 'tuple' object has no attribute 'remove'
```

Note: tuple is the read only version of list

range Data Type:

range Data Type represents a sequence of numbers.
The elements present in range Data type are not modifiable. I.e range Data type is immutable.

Form-1: range(10)
generate numbers from 0 to 9

Eg:
```
r=range(10)
for i in r : print(i)    0 to 9
```

Form-2: range(10,20)

generate numbers from 10 to 19

```
r = range(10,20)
for i in r : print(i)    10 to 19
```

Form-3: range(10,20,2)

2 means increment value

```
r = range(10,20,2)
for i in r : print(i)   10,12,14,16,18
```

We can access elements present in the range Data Type by using index.
```
r=range(10,20)
r[0]==>10
r[15]==>IndexError: range object index out of range
```

We cannot modify the values of range data type

Eg:
```
r[0]=100
TypeError: 'range' object does not support item assignment
```

We can create a list of values with range data type

Eg:

```
1) >>> l = list(range(10))
2) >>> l
3) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

set Data Type:

If we want to represent a group of values without duplicates where order is not important then we should go for set Data Type.

1. insertion order is not preserved
2. duplicates are not allowed
3. heterogeneous objects are allowed
4. index concept is not applicable
5. It is mutable collection
6. Growable in nature

Eg:

```
1)  s={100,0,10,200,10,'durga'}
2)  s # {0, 100, 'durga', 200, 10}
3)  s[0] ==>TypeError: 'set' object does not support indexing
4)
5)  set is growable in nature, based on our requirement we can increase or decrease the size.
6)
7)  >>> s.add(60)
8)  >>> s
9)  {0, 100, 'durga', 200, 10, 60}
10) >>> s.remove(100)
11) >>> s
12) {0, 'durga', 200, 10, 60}
```

frozenset Data Type:

It is exactly same as set except that it is immutable.
Hence we cannot use add or remove functions.

```
1)  >>> s={10,20,30,40}
2)  >>> fs=frozenset(s)
3)  >>> type(fs)
4)  <class 'frozenset'>
5)  >>> fs
6)  frozenset({40, 10, 20, 30})
7)  >>> for i in fs:print(i)
8)  ...
9)  40
10) 10
11) 20
12) 30
13)
14) >>> fs.add(70)
15) AttributeError: 'frozenset' object has no attribute 'add'
16) >>> fs.remove(10)
17) AttributeError: 'frozenset' object has no attribute 'remove'
```

dict Data Type:

If we want to represent a group of values as key-value pairs then we should go for dict data type.

Eg:
d={101:'durga',102:'ravi',103:'shiva'}

Duplicate keys are not allowed but values can be duplicated. If we are trying to insert an entry with duplicate key then old value will be replaced with new value.

Eg:

```
>>> d={101:'durga',102:'ravi',103:'shiva'}
>>> d[101]='sunny'
>>> d
{101: 'sunny', 102: 'ravi', 103: 'shiva'}

We can create empty dictionary as follows
d={ }
We can add key-value pairs as follows
d['a']='apple'
d['b']='banana'
print(d)
```

Note: dict is mutable and the order wont be preserved.

Note:

1. In general we can use bytes and bytearray data types to represent binary information like images,video files etc
2. In Python2 long data type is available. But in Python3 it is not available and we can represent long values also by using int type only.
3. In Python there is no char data type. Hence we can represent char values also by using str type.

DURGASOFT, # 202, 2nd Floor, HUDA Maitrivanam, Ameerpet, Hyderabad - 500038,
☎ 040 – 64 51 27 86, 80 96 96 96 96, 92 46 21 21 43 | www.durgasoft.com

Summary of Datatypes in Python3

Datatype	Description	Is Immutable	Example

Int	We can use to represent the whole/integral numbers	Immutable	>>> a=10
			>>> type(a)
			<class 'int'>

Float	We can use to represent the decimal/floating point numbers	Immutable	>>> b=10.5
			>>> type(b)
			<class 'float'>

Complex	We can use to represent the complex numbers	Immutable	>>> c=10+5j
			>>> type(c)
			<class 'complex'>
			>>> c.real
			10.0
			>>> c.imag
			5.0

Bool	We can use to represent the logical values(Only allowed values are True and False)	Immutable	>>> flag=True
			>>> flag=False
			>>> type(flag)
			<class 'bool'>

Str	To represent sequence of Characters	Immutable	>>> s='durga'
			>>> type(s)
			<class 'str'>
			>>> s="durga"
			>>> s='''Durga Software Solutions
			... Ameerpet'''
			>>> type(s)
			<class 'str'>

bytes	To represent a sequence of byte values from 0-255	Immutable	>>> list=[1,2,3,4]
			>>> b=bytes(list)
			>>> type(b)
			<class 'bytes'>

bytearray	To represent a sequence of byte values from 0-255	Mutable	>>> list=[10,20,30]
			>>> ba=bytearray(list)
			>>> type(ba)
			<class 'bytearray'>

range	To represent a range of values	Immutable	>>> r=range(10)
			>>> r1=range(0,10)
			>>> r2=range(0,10,2)

list	To represent an ordered collection of objects	Mutable	>>> l=[10,11,12,13,14,15]
			>>> type(l)
			<class 'list'>

tuple	To represent an ordered collections of objects	Immutable	>>> t=(1,2,3,4,5)
			>>> type(t)
			<class 'tuple'>

set	To represent an unordered collection of unique objects	Mutable	>>> s={1,2,3,4,5,6} >>> type(s)

			<class 'set'>

frozenset	To represent an unordered collection of unique objects	Immutable	>>> s={11,2,3,'Durga',100,'Ramu'}
			>>> fs=frozenset(s)
			>>> type(fs)
			<class 'frozenset'>

dict	To represent a group of key value pairs	Mutable	>>> d={101:'durga',102:'ramu',103:'hari'} >>> type(d) <class 'dict'>

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.
It is something like null value in Java.

Eg:

```
def m1():
  a=10

print(m1())
  None
```

Escape Characters:

In String literals we can use esacpe characters to associate a special meaning.

```
1)  >>> s="durga\nsoftware"
2)  >>> print(s)
3)  durga
4)  software
5)  >>> s="durga\tsoftware"
6)  >>> print(s)
7)  durga  software
8)  >>> s="This is " symbol"
9)    File "<stdin>", line 1
10)    s="This is " symbol"
11)                ^
12) SyntaxError: invalid syntax
13) >>> s="This is \" symbol"
14) >>> print(s)
15) This is " symbol
```

The following are various important escape characters in Python

1) \n==>New Line
2) \t===>Horizontal tab
3) \r ==>Carriage Return
4) \b===>Back space
5) \f===>Form Feed
6) \v==>Vertical tab
7) \'===>Single quote
8) \"===>Double quote
9) \\===>back slash symbol

....

Constants:

Constants concept is not applicable in Python.
But it is convention to use only uppercase characters if we don't want to change value.

MAX_VALUE=10

It is just convention but we can change the value.