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:

```
public class Hello
{
  public static void main(String args[])
  {
    System.out.println("Hello World!!");
  }
}
```

Eg2: To print Helloworld C:

```
#include<stdio.h>
 void main()
 {
    printf("Hello World!!");
  }
```

Eg2: To print Helloworld Python:

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

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

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 etc.

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. i.e. we can migrate from one platform to another platform very easily. Python programs will provide same results on any platform

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 because 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:
        - ♣ We can use already existing legacy non-Python code
        - ♣ We can improve performance of the application

10. **Embedded**: We can use Python programs in any other language programs. i.e we can embedded 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.

Identifiers

- A Name in Python Program is called Identifier.
- It can be Class Name OR Function Name OR Module Name OR Variable Name.

  Eg: 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 (_)

2. Identifier should not start with digit
   - 123total ρ
   - total123 √

3. Identifiers are case sensitive. Of course Python language is case sensitive language.
   - total=10
   - TOTAL=999
   - print(total) #10
   - print(TOTAL) #999

4. We cannot use reserved words as identifiers
5. There is no length limit for Python identifiers. But not recommended to use too lengthy identifiers.
6. Dollor ($) Symbol is not allowed in Python.

Example:

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 Word

In Python some words are reserved to represent some meaning or functionality. Such types 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

Datatypes in Python

- 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 contains the following inbuilt data types

1) int

2) float

3) complex

4) bool

5) str

6) bytes

7) bytearray

8) range

9) list

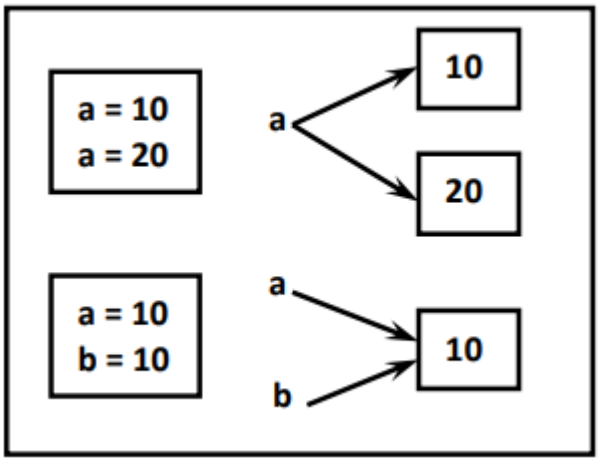
10) tuple

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

1) int Data Type:

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

Eg: a = 10 type(a) #int

We can represent int values in the following ways

1) Decimal form
2) Binary form
3) Octal form
4) Hexadecimal form

I) Decimal Form (Base-10):

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

II)Binary Form (Base-2):

- The allowed digits are: 0 & 1
- Literal value should be prefixed with 0b or 0B
- Eg: a = 0B1111, a = 0B123

III) 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

IV) 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

Base Conversions

Python provide the following in-built functions for base conversions

1. bin()
2. oct()
3. hex()

1) bin():

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

bin(15) → '0b1111'

bin(0o11) → '0b1001'

bin(0X10) 6) → '0b10000'

2) oct():

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

oct(10) → '0o12'

oct(0B1111) → '0o17'

oct(0X123) → '0o443'

3) hex():

We can use hex() to convert from any base to hexadecimal

hex(100) → '0x64'

hex(0B111111) → '0x3f'

hex(0o12345) → '0x14e5'

2. Float Data Type:

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

  Eg: f = 1.234

- We can also represent floating point values by using exponential form (Scientific Notation)

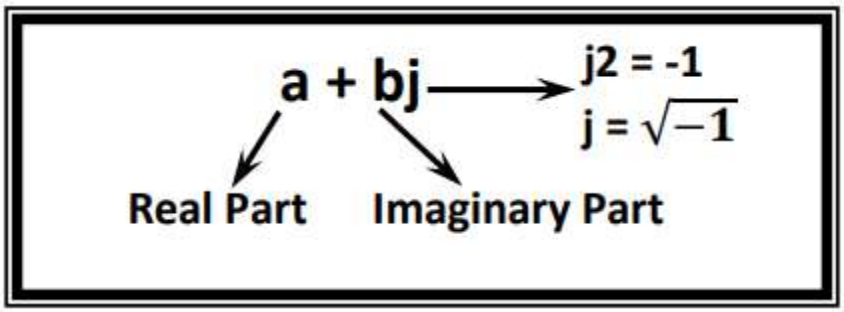
  Eg: f = 1.2e3 ➔ instead of 'e' we can use 'E' print(f) 1200.0

- 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 hexadecimal forms. But we can represent float values only by using decimal form.

3. 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 hexadecimal form.

- But imaginary part should be specified only by using decimal form.

  a=0B11+5j, → print(a) → 3+5j

- Even we can perform operations on complex type values.

```
>>> a=10+1.5j
>>> b=20+2.5j
>>> c=a+b
>>> print(c)
>>>(30+4j)
```

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

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

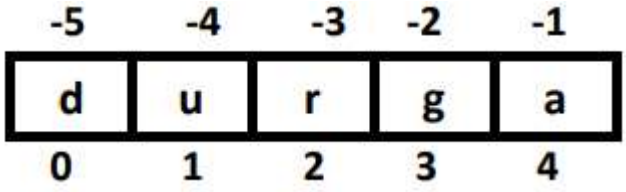
True+True → 2
True-False → 1

5. str Data Type:

- str represents String data type.
- A String is a sequence of characters enclosed within single quotes or double quotes.
  - ♣ s1='Bishal'
  - ♣ s1="Bishal"
- By using single quotes or double quotes we cannot represent multi line string literals.
  - ♣ s1="Bishal Patel"
- For this requirement we should go for triple single quotes (''') or triple double quotes (""")
  - ♣ s1='''Bishal Patel'''
  - ♣ s1="""Bishal Patel"""
- 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:

1) slice means a piece
2) [ ] operator is called slice operator, which can be used to retrieve parts of String.
3) In Python Strings follows zero based index.
4) The index can be either +ve or -ve.
5) +ve index means forward direction from Left to Right
6) -ve index means backward direction from Right to Left




```
>>> s="durga"
>>> s[0]
'd'
>>> s[1] 5) 'u'
>>> s[-1]
'a'
>>> s[40]
```
IndexError: string index out of range

```
>>> s[1:40]
'urga'
>>> s[1:]
'urga'
>>> s[:4]
'durg'
>>> s[:]
'durga'
>>> s*3
'durgadurgadurga'
>>> len(s)
5
```

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()

int():

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

```
>>> int(123.987)
123
>>> int(10+5j)
TypeError: can't convert complex to int

>>> int(True)
1
>>> int(False)
0
>>> int("10")
10
>>> int("10.5")
ValueError: invalid literal for int() with base 10: '10.5'

>>> int("ten")
ValueError: invalid literal for int() with base 10: 'ten'

>>> int("0B1111")
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, compulsory str should contain only integral value and should be specified in base-10.

float():

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

```
>>> float(10)
10.0
>>> float(10+5j)
TypeError: can't convert complex to float

>>> float(True)
1.0
>>> float(False)
0.0
>>> float("10")
10.0
>>> float("10.5")
10.5
>>> float("ten")
ValueError: could not convert string to float: 'ten'
```

```
>>> float("0B1111")
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 compulsory str should be either integral or floating point literal and should be specified only in base-10.

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:

```
complex(10)==>10+0j
complex(10.5)===>10.5+0j
complex(True)==>1+0j
complex(False)==>0j
complex("10")==>10+0j
complex("10.5")==>10.5+0j
complex("ten")
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:

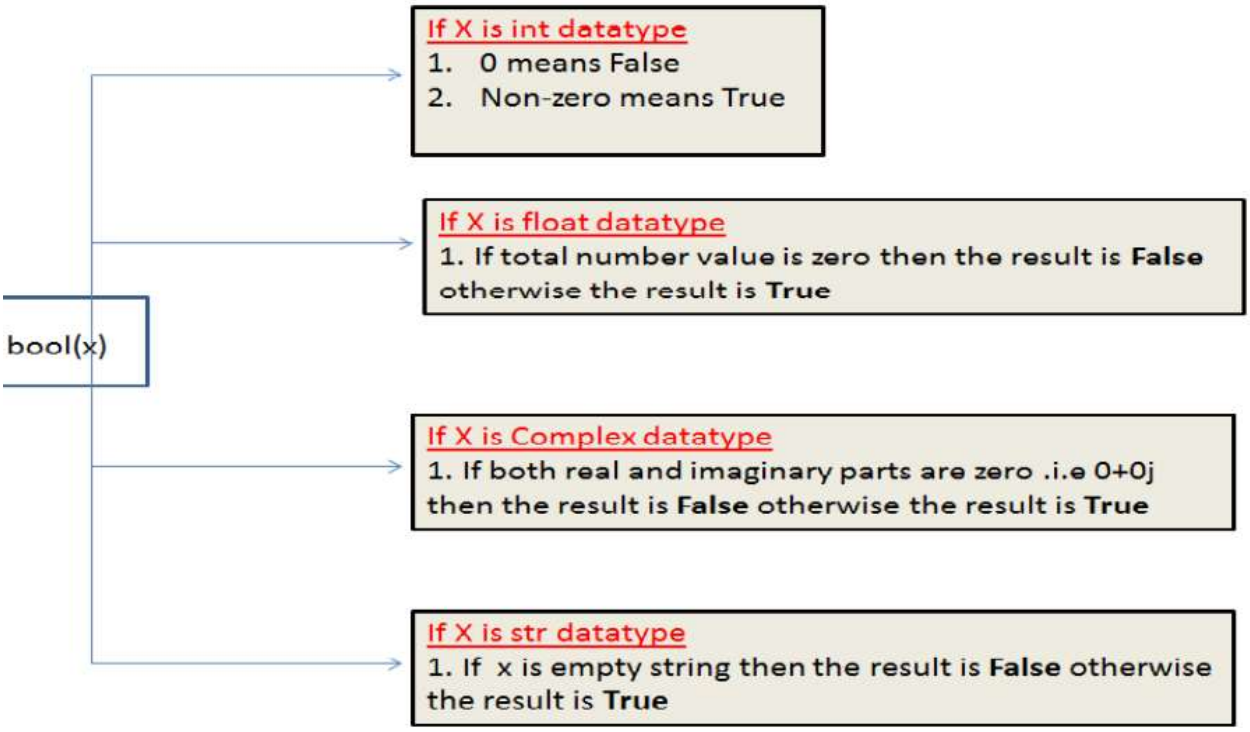
complex(10, -2) → 10-2j
complex(True, False) → 1+0j

bool():

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

bool(0) → False
bool(1) → True
bool(10) → True
bool(10.5) → True
bool(0.178) → True
bool(0.0) → False
bool(10-2j) → True
bool(0+1.5j) → True
bool(0+0j) → False
bool(1+0j) → True

bool("True") → True
bool("False") → True
bool("") → False



str():

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

```
>>> str(10)
'10'
>>> str(10.5)
'10.5'
>>> str(10+5j)
'(10+5j)'
>>> str(True)
'True'
```

Fundamental Data Types vs Immutability:

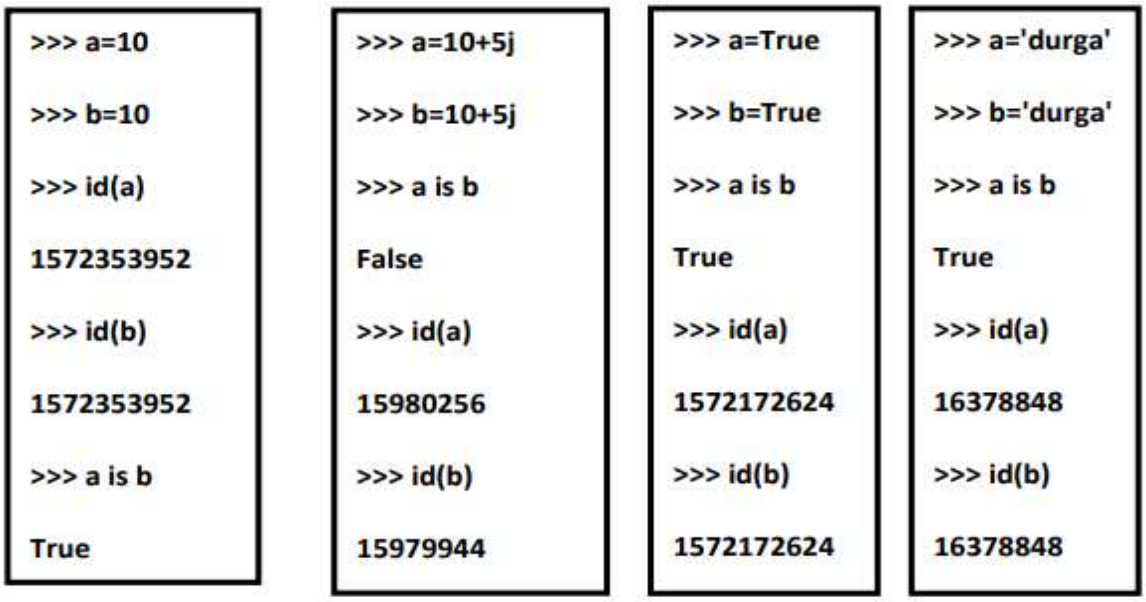
All Fundamental Data types are immutable. i.e. once we create an object, we cannot perform any changes in that object. If we are trying to change then with those changes a new object will be created. This non-changeable behavior is called immutability.

In Python if a new object is required, then PVM won't 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.

Example:

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



6. bytes Data type

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

```
x = [10,20,30,40]
b = bytes(x)
type(b) → bytes
print(b[0]) → 10
print(b[-1]) → 40
for i in b:
    print(i)
```

Output:

10
20
30
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 create bytes data type value, we cannot change its values, otherwise we will get TypeError.

Example:

```
>>> x = [10,20,30,40]
>>> b=bytes(x)
>>> b[0] =100
```

TypeError: 'bytes' object does not support item assignment

7) bytearray Data Type:

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

Eg 1:

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

Output:

```
10
20
30
40
```

```
b[0]=100
for i in b:
   print(i)
```

Output:

```
100
20
30
40
```

Eg 2:

```
>>> x =[10,256]
>>> b = bytearray(x)
```

ValueError: byte must be in range(0, 256)

8) 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]
print(list) # [10,10.5,'durga',True,10]
```

Eg 2:

```
list=[10,20,30,40]
>>> list[0]
10
>>> list[-1]
40
>>> list[1:3]
[20, 30]
>>> list[0]=100
>>> for i in list:
        print(i)
```

Output:

```
100
20
30
40
```

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

```
>>> list=[10,20,30]
>>> list.append("durga")
>>> list
[10, 20, 30, 'durga']
>>> list.remove(20)
>>> list
[10, 30, 'durga']
>>> list2=list*2
>>> list2
[10, 30, 'durga', 10, 30, 'durga']
```

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

9) Tuple Data Type:

- tuple data type is exactly same as list data type except that it is immutable. i.e. we cannot change values.
- Tuple elements can be represented within parenthesis.

Eg:

```
t = (10,20,30,40)
type(t)  → #  tuple
t[0]=100
TypeError: 'tuple' object does not support item assignment
>>> t.append("durga")
```

AttributeError: 'tuple' object has no attribute 'append'
>>> t.remove(10)
AttributeError: 'tuple' object has no attribute 'remove'

Note: tuple is the read only version of list

10) 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

Eg:

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

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

2 means increment value

Eg:

```
 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.

Eg:

```
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:

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

11) 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:

```
>>>s ={100,0,10,200,10,'durga'}
>>>s # {0, 100, 'durga', 200, 10}
>>>s[0] → TypeError: 'set' object does not support indexing
```

set is growable in nature, based on our requirement we can increase or decrease the size.

```
>>> s.add(60)
>>> s
{0, 100, 'durga', 200, 10, 60}
>>> s.remove(100)
>>> s
{0, 'durga', 200, 10, 60}
```

12) frozenset Data Type:

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

```
>>> s={10,20,30,40}
>>> fs=frozenset(s)
>>> type(fs)
>>> fs
frozenset({40, 10, 20, 30})
```

```
>>> for i in fs:
    print(i)
40
10
20
30

 >>> fs.add(70)
AttributeError: 'frozenset' object has no attribute 'add'
>>> fs.remove(10)
AttributeError: 'frozenset' object has no attribute 'remove'
```

13) 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 won't 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.

14) 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 escape characters to associate a special meaning.

```
>>> s="durga\nsoftware"
>>> print(s)
durga
software
>>> s="durga\tsoftware"
>>> print(s)
durga software
>>> s="This is " symbol"
File "<stdin>", line 1
s="This is " symbol"
SyntaxError: invalid syntax
>>> s="This is \" symbol"
>>> print(s)

      This is " symbol
```

The following are various important escape characters in Python

1) **\n**
2) **\t**
3) **\r**
4) **\b**
5) **\f**
6) **\v**
7) **\'**
8) **\"**
9) ****

....

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.