**Built in Object Types in Python**

* All programming languages support a number of built-in variable types.These are the building blocks used for all the other variable types available within the language.
* By combining or extending the basic variable,we can create some complex variables.
* Python variables are actually objects.by using objects at the basic storage level the overall language is object based.This has a number of advantages.You can use the same methods and functions to operate on different types of objects.
* Python supports a wide variety of built-in object types that helps to make your programs easier to write.

Python supports five main built in object types



**Numeric:**

Numeric data type having the data which has numeric value i.e the numeric value which can be integer,float,complex number.

* Integers belongs to the class ‘int’
* Float belongs to the class ‘float’
* Complex belongs to the class ‘complex’
1. **Integer:**

Integers in python are basic data types used to represent whole numbers.

Integers are objects of the class int.

It can have numbers either positive, negative or zero

Python has no restriction on how long an integer value can take.

Ex:10,20,99,-23,0,-100 etc.

1. **Float:**

Float data type is used to represent the numbers with decimal points.

Float numbers are objects of the class float.

It is accurate up to 15 decimal points.

Internally python stores floating- point values as C doubles,giving as much precision as possible.

Ex: 3.05,1.674,3.17

1. **Complex:**

They are used to represent numbers with both a real and imaginary part.

Complex numbers are objects of the class complex.

The form of the complex number is x+iy. here x is real part and y is imaginary part

Ex: 2+5j, 5+5j

Python has a function named type function which is used to know the data type of any variable.

Example:

INPUT:

#Examples of numeric data types

a = 10

print("The datatype of a is :", type(a))

b = 3.14

print("The datatype of b is : ", type(b))

c = 5 + 9j

print("The datatype of c is : ", type(c))

**Output:**

The datatype of a is : <class 'int'>

The datatype of b is : <class 'float'>

The datatype of c is : <class 'complex'>

**BOOLEAN:**

Boolean datatype provides only two values

 1.True

 2.False

Booleans are objects of the class bool.

The first letter in both True and False must be capitalized otherwise the python interpreter will throw an error.

Booleans are essential in program flow and decision making.

These values are used to determine the given statement true or false.

* True can be represented by any non-zero value or the letter 'T'
* False can be represented by the 0 or the letter 'F'.

Example:

INPUT:

#Examples of boolean data type

a =True

print("The datatype of a is:", type(a))

b =False

print("The datatype of b is:", type(b))

Output:

The datatype of a is: <class 'bool'>

The datatype of b is: <class 'bool'>

Sequence data types

**List** :

 list is a sequence data type which is used to store a collection of items which are of the same or different data types and the items are accessible using the index.

Lists are mutable i.e they can be modified after the list is created.

Lists are ordered i.e when you add elements to a list, they are placed in a particular sequence, and that sequence is preserved until you change it.

Use case : Data Analysis, Managing collections of data

**Creation of list**

This can be done in 2 ways

1. Using Square Brackets

By using square brackets with elements separated by commas.

Example :

lis=[] # empty list

list\_1=[1,2,3,4,5,6,7,8,9]

print(list\_1) #prints [1, 2, 3, 4, 5, 6, 7, 8, 9]

1. Using list constructor

By passing an iterable to the list constructor.

Example :

l=list ((10,20,30,40,50,60,70,80,90,))

print(l) # prints [10, 20, 30, 40, 50, 60, 70, 80, 90]

**Tuple** :

Tuple is used to store a collection of items and the items are accessible using the index.

Tuples are immutable i.e once they are created they can’t be modified.

Use Case : Representing fixed collections of items

**Creation of tuple**

This can be done in 2 ways

1. Using parentheses

By using parentheses with elements separated by commas

Example :

My\_tuple=() # empty tuple

print(type(My\_tuple)) # prints <class 'tuple'>

My\_tuple1=(44,55,66,77,88,99,22,11,66)

print(My\_tuple1) # prints (44, 55, 66, 77, 88, 99, 22, 11, 66)

1. Using tuple constructor

By passing an iterable to the tuple constructor.

Example :

My\_tuple=tuple((23,34,56,79,89))

print(My\_tuple) # prints (23, 34, 56, 79, 89)

**String** :

Strings are a sequence of characters which are enclosed in single or double quotes.

Strings are immutable i.e once they are created they can’t be modified.

Use Case : Text processing, Data representation, Regular Expressions, File I/O

**Creation of strings**

You can create strings by using the single or double quotes.

Example :

My\_string= 'Hello World'

print(type(My\_string)) # prints <class 'str'>

My\_string1="Welcome to Python"

print(type(My\_string1)) #prints <class 'str'>

**SET DATATYPE:**

* Python Set is the unordered collection of the unique elements i.e. set will not maintain any specific order of elements and each is unique.
* It is iterable and has only unique elements.Set data structure is used when we want each element to be unique in a large collection of data.
* Set is mutable i.e they can be modified after the set is created.

**Creation of set**

This can be done in 2 ways

1. Using Curly Braces

Example :

s={1,2,3,4,5,6,7}

print(type(s)) # prints <class 'set'>

print(s) #prints {1, 2, 3, 4, 5, 6, 7}

1. Using set constructor

By passing an iterable to the set constructor.

Example :

s=set((11,22,33,44,55,66,77))

print(type(s)) # prints <class 'set'>

print(s) #prints {33, 66, 11, 44, 77, 22, 55}

**Dictionary**
Dictionary is a mutable, unordered collection of key-value pairs.

it provides a way to map unique keys to corresponding values.

Use Case : Mapping relations, grouping the data

**Creation of Dictionary**

This can be done in 2 ways

1. Using Curly Braces with colon

Example :

My\_dict1={} # creation of empty dictionary

print(type(My\_dict1)) # prints <class 'dict'>

My\_dict={"Name":"Vinod", "Age": 34 , "City": "London"}

print(type(My\_dict)) # prints <class 'dict'>

print(My\_dict) # prints {'Name': 'Vinod', 'Age': 34, 'City': 'London'}

1. Using dict constructor

By passing key value pairs as arguments to an iterable to the dictionary constructor.

Example :

My\_dict=dict(Name = "Vinod", Age= 34 ,City ="London")

print(type(My\_dict)) # prints <class 'dict'>

print(My\_dict) # prints {'Name': 'Vinod', 'Age': 34, 'City': 'London'}