DataTypes(or)DataStructures(or)CollectionTypes
A collection is an object that represents group of objects.

Advantages of Collection Types:
1) Reduces programming effort
2) Increases programming speed and quality

There are five collection types in python:
1) list 2) tuple 3) set 4) frozenset 5) dict

List:
**List can be created by using square brackets
**List can also be created by using list() function
**List can have homogeneous elements or heterogeneous elements
**List allows duplicates
**List supports both positive and negative indexing
**List supports slice operations also
**List is mutable
**List supports deleting an element
**List is an iterable object
**Elements of a list can be iterated by using for loop or while loop
**Insertion order is preserved in a list

Examples:
1) List with homogeneous elements:
```
a=[10,20,15,25,30]
print(a)
print(type(a))
```

2) List with heterogeneous elements:
```
a=[10, 10.2, “abc”, True]
print(a)
```

3) List with duplicate elements(Indexing & Slicing Operations):
```
a=[10,20,10,50,88,31,89,88]
print(a)
print(a[1])
print(a[-3])
print(a[1:3])
print(a[-4:-1])
```

4) Modifying and Deleting elements:
```
a=[10,20,30,13,56,93]
print(a)
a[1]=22
print(a)
del a[2]
print(a)
```

5) Iterating elements of a list by using for loop:

```
a=[10,33,63,98,13,67]
for b in a:
     print(b)
```

6) Iterating elements of a list by using while loop:

```
a=[10, 98, 32, 81, 38, 90]
i=0
while i<len(a):
   print(a[i])
```

7) List in a list(Nested list):

```
a=[[10,11,12],[20,21,22],[30,31,32]]
for b in a:
    print(b)
    for c in b:
        print(c)
```

8) Searching for an element in a list:

```
a=[10, 98, 32, 81, 38, 90]
b=32
if b in a:
     print("Element found")
else:
     print("Element not found")
```

9) Unpacking elements from list:

```
a=[10,20,30]
b,c,d=a
print(a)
print(b)
print(c)
print(d)
```

10) Converting a string into a list:

```
a="welcome"
b=list(a)
print(a)
```

List comprehensions:

The concept of generating elements into a list by writing some logic in a list is known as list comprehension.

Examples:

1)
```
a=[i for i range(10)]
print(a)
```

2)
```
a=[i*i for i in range(10,20)]
print(a)
```

Functions:

1) len() function: It returns the length of the list
2) max() function: It returns the maximum value in a list
3) min() function: It returns the minimum value in a list
4) sum() function: It returns sum of the elements in a list
5) sorted() function: It returns sorted list

Example:

```
a=[10,23,83,43,92,13]
print(len(a))
print(max(a))
print(min(a))
print(sum(a))
print(sorted(a))
```

Methods:

1) append() method: It is used append an element at the end of the list.
2) count() method: It is used to count the no. of times appears a specified element in a list.
3) index() method: It is used to get the lowest index of specified element.
4) copy() method: It is used to create a copy of list.
5) remove() method: It is used to remove specified element.
6) extend() method: It is used to extend a list with another list.
7) sort() method: It is used to sort elements in a list.
8) reverse() method: It is used to reverse elements in a list.
9) clear() method: It is used to remove all elements from list.
10) pop() method: It is used to remove an element at the specified index.
11) insert() method: It is used to insert an element at the specified index.

Example:

```
a=[10,39,53,39,81,35]
print(a)
a.append(33)
print(a)
print(a.count(39))
print(a.index(81))
b=a.copy()
print(b)
a.remove(53)
print(a)
a.extend(b)
print(a)
a.sort()
print(a)
a.reverse()
print(a)
a.pop(2)
print(a)
a.insert(1,91)
print(a)
a.clear()
```

```
print(a)
```

Tuple:

- Tuple can be created by using parenthesis
- Tuple can also be created by using tuple() function
- Tuple can also be created by assigning multiple values
- Tuple can have homogeneous elements or heterogeneous elements
- Insertion order is preserved in a tuple
- Tuple allows duplicates
- Tuple supports both positive and negative indexing
- Tuple supports slice operations also
- Tuple is immutable
- Tuple does not support to delete an element
- Tuple can be deleted
- Tuple is an iterable object

Tuple Functions:

i) len() ii) max() iii) min() iv) sum() v) sorted()

Methods of Tuple:

i) count() ii) index()

Differences between List and Tuple

List	Tuple
1) It is mutable	1) It is immutable
2) Deleting an element is possible	2) Deleting an element is not possible
3) Many methods are present in a list	3) Few methods are present in a tuple
4) List cannot be used as a key in a dict	4) Tuple can be used as a key in a dict

Set:

- Set can be created by using curly braces
- Set can also be created by using set() function
- {} empty curly braces treated as dict
- {10} atleast one element if we write then it will be treated as Set
- Set can have homogeneous elements or heterogeneous elements
- Set does not allow duplicates
- If we write duplicates, automatically ignores
- In a set insertion order is not preserved where as in a list and
- tuple insertion order is preserved.
- Set does not support indexing
- Set does not support slice operations
- Set is mutable because we can add an element by using add()
method and we can remove an element by using remove() method.
- Set is an iterable object
- Set in a set is not possible(Nested Set is not possible)

Ø Tuple can be stored in a Set
Ø List cannot be stored in a Set
Ø Set can be stored in a List
Ø Set can be stored in a Tuple

Set Functions:
i) len() ii) max() iii) min() iv) sum() v) sorted()

Set Methods:
i) add() ii) remove() iii) clear() iv) copy()

Set supports mathematical operations like union, intersection, difference & symmetric difference
1) a|b (or) a.union(b) => Returns a set that has elements from both the sets.
2) a&b (or) a.intersection(b) => Returns a set that has elements which are common in both the sets.
3) a-b (or)a.difference(b) => Returns a set that has elements in a but not in b
4) iv) a^b (or) a.symmetric_difference(b) => Returns a set with elements either in a or in b but not both

Example:
```
a={10,20,30,40,50}
b={40,50,60,70,80}
print(a|b)
print(a.union(b)
print(a&b)
print(a.intersection(b))
print(a-b)
print(a.difference(b))
print(a^b)
print(a.symmetric_difference(b))
```

frozenset:
Ø frozenset is a immutable version of a set
Ø frozenset can be created by using frozenset() function
Ø We cannot add an element and we cannot remove an element because frozenset is immutable set
Ø frozenset can be used as a key in a dictionary

Example:
```
a={10,20,30,40,50}
b=frozenset(a)
print(a)
print(type(a))
print(b)
print(type(b)
a.add(83)
print(a)
b.add(83) =>Error, because immutable set
print(b)
```

Dict:
Ø Dictionary can be created by using curly braces
Ø Dictionary can also be created by using dict() function
Ø Dictionary maintains data as a key/value pairs
Ø Keys & values are separated with ":" symbol
Ø Here keys are immutable
Ø Here values are mutable
Ø Dictionary does not allow duplicate keys
Ø Values may be duplicated
Ø If we write duplicate keys, automatically ignores
Ø Insertion order is not preserved
Ø Keys & values can be homogeneous or heterogeneous
Ø Dictionary does not support indexing
Ø Dictionary does not support slice operations
Ø Here values can be accessed by using keys
Ø Dictionary is mutable because new key & value pair can be added and existed value can be modified
Ø Dictionary is an iterable object
Ø List cannot be used as a key
Ø List can be used as a value
Ø Tuple can be used as a key
Ø Tuple can be used as a value
Ø Set cannot be used as a key
Ø Set can be used as a value

Dictionary Functions:
i) len() ii) max() iii) min() iv) sum() v) sorted()

Dictionary Methods:
i) copy() ii) clear() iii) items() iv) keys() v) values() vi) get(key)

Example:
```
a={101:5000.00, 102:5500.00, 103=6000.00}
print(a)
a[104]=9000.00
print(a)
print(a[103])
```
Iterating keys:
```
a={101:5000.00, 102:5500.00, 103=6000.00}
for i in a:
    print(i)
```

Iterating Values:
```
a={101:5000.00, 102:5500.00, 103=6000.00}
for i in a.values():
    print(i)
```

Iterating both keys & values:
```
a={101:5000.00, 102:5500.00, 103=6000.00}
```

```
for i in a.items():
    print(i)

Nested dictionary:
a={“venkatesh”:{“C”:99, “Java”:88, “Python”:90}}
print(a)

Searching a key:
a={101:5000.00, 102:5500.00, 103=6000.00}
b=102
if b in a:
    print(“Given key present in a dictionary”)
else:
    print(“Given key not present in a dictionary”)

Unpacking key & values from dictionary:
a={101:5000.00, 102:5500.00, 103=6000.00}
b,c,d=a.items()
print(b)
print(c)
print(d)

Dictionary comprehensions:
a={i:i*i for i in range(10)}
print(a)

================================================================================
============================
Functions and Modules::


Functions:
A group of statements into a single logical unit is called as
function.
Functions are used to perform the task.
Function is not called automatically.
Function body is executed whenever we call the function
Function can be called any no. of times.

Syntax:
def function_name():
       =============
       =============

Advantages of Functions:
1) Modularity
2) Reusability

Example:
def display():
```

```
      print("Welcome")
display()
display()
display()
```

Functions are divided into 4 categories:
1) Functions with arguments with return value.
2) Functions with arguments and without return value.
3) Functions without arguments and with return value.
4) Functions without arguments and without return value.

Examples:

```
def add(a, b):
       return a+b
def sub(a, b):
       print(a-b)
def mul():
       a=5
       b=10
       return a*b
def div():
       a=5
       b=10
       print(a//b)
```

==========

Parameters:
The variables that are declared in a function declaration is called parameters.
Parameters of a function can be accessed with in the same function only.

Python supports two types of parameters:
1) Non default parameters
2) Default parameters

Non default parameters:
The parameters that are declared without assigning values are called non default parameters.
At the time of calling a function we should pass values of the non default parameters of the function.

Example:

```
def add(a, b):
      c=a+b
      print(c)
add(10, 20)
add(30, 83)
```

Default Parameters:

The parameters that are declared by assigning values in a
function declaration are called default parameters.
At the time of calling a function, we need not to pass the values.

Example:

```
def add(a=10, b=20):
      c=a+b
      print(c)
add()
add(30)
add(40, 50)
```

==========

Arguments:
The values that are passed to a function at the time of calling
a function are called arguments.

Types of arguments:
1) Non keyword arguments
2) Keyword arguments
3) Arbitrary arguments

Non keyword Arguments:
The arguments that are passed without assigning to variables
are called non keyword arguments.

Example:

```
def sub(a, b):
     print(a-b)
sub(10, 2)
sub(2, 10)
```

Keyword arguments:
The arguments that are passed with assigning to variables
are called non keyword arguments.

Example:

```
def sub(a, b):
     print(a-b)
sub(a=10, b=2)
sub(b=2, a=10)
```

Arbitrary arguments:
The arguments that are prefixed with * are called arbitrary
arguments.
Arbitrary arguments are by default tuple type.
It allows to pass 0 to any number of arguments to a function.

Example:

```
def add(*a):
```

```
      for i in a:
           print(i)
add()
add(10)
add(33, 45)
add(31, 13, 54)
```

Lambda Function (or) Anonymous Function:
A function that has no name is known as lambda function or anonymous function.

Syntax: variable=lambda arguments : expression

Examples:

```
1) fun=lambda a : a*a*a
    b=fun(10)
    print(b)

2) add=lambda a, b : print(a+b)
    add(10, 20)
```

==========

Modules:
In python, every file treated as a module.
A module can contains variables, functions, classes,.. etc.,
Python supports two types of import statements to access modules data.
1) Normal import
2) From import

1) Normal import:
In normal import we can access variables and functions by using module name

Example:
test.py

```
a=10
def add(a, b):
     print(a+b)
```

demo.py

```
import test
print(test.a)
test.add(10, 20)
```

In normal import, instead of module name we can use any alias name also

Example:

```
test.py
a=10
def add(a, b):
     print(a+b)

demo.py
import test as t
print(t.a)
t.add(10, 20)
```

2) From import:
In from import we can access variables and functions directly

Example:

```
test.py
a=10
def add(a, b):
     print(a+b)

demo.py
from test import a, add
print(a)
add(10, 20)
```

In from import, we can import all variable and functions by using * symbol

Example:

```
test.py
a=10
def add(a, b):
     print(a+b)

demo.py
from test import *
print(a)
add(10, 20)
```

Module search path:
By default python interpreter will search in the following locations:
1) Current directory
2) sys.path
3) pythonpath variable in Environment variables.

Packages:
A package is a folder or directory, which contains modules.
A package can also contain sub packages.
To access modules of package, we use packagename.modulename

==========

ExceptionHandling ::

There are three types of errors:
1) Compile time errors
2) Run time errors
3) Logical errors

Compile time errors are also called syntax errors.
Run time errors are also called exceptions.

Example of syntax error:
```
i=0
if i==0
   print(i)
```
In the above example we have missed the : symbol after if statement. It is called as syntax error.

Example of exception:
```
a=10
b=0
print(a//b)
```
In the above example ZeroDivisionError occurs. It is called as an exception.

This type of exceptions can be handled explicitly to display user friendly error messages.

Handling Exceptions:
Syntaxes:
1) try with except block:
```
    try:
         ========
    except ExceptionClassName:
        ========
```

2) try with multiple except blocks:
```
    try:
         ==========
    except ExceptionClassName1:
        ===========
    except ExceptionClassName2:
        ===========
```

3) Multiple exceptions in a single block:
```
    try:
         ==========
```

```
    except(ExceptionClassName1, ExceptionClassName2, ........)
         ==========

4) except block without exception class name
    try:
        ==========
    except:
        ==========

Examples:
1)
try:
    a=int(input("Enter First Number: "))
    b=int(input("Enter Second Number: "))
    c=a//b
    print(c)
except ZeroDivisionError:
    print("Enter Second Number Except Zero")

2)
try:
    a=int(input("Enter First Number: "))
    b=int(input("Enter Second Number: "))
    c=a//b
    print(c)
except ValueError:
    print("Enter Numbers Only")
except ZeroDivisionError:
    print("Enter Second Number Except Zero")

3)
try:
    a=int(input("Enter First Number: "))
    b=int(input("Enter Second Number: "))
    c=a//b
    print(c)
except(ZeroDivisionError, ValueError):
    print("Enter two numbers and second number except zero")

4)
try:
    a=int(input("Enter First Number: "))
    b=int(input("Enter Second Number: "))
    c=a//b
    print(c)
except:
    print("Enter two numbers and second number except zero")
```