3...

Data Structures in Python

Chapter Outcomes...

- Write Python program to use and manipulate lists for the given problem.
- Write Python program to use and manipulate tuples for the given problem.
- Write Python program to use and manipulate sets for the given problem.
- Write Python program to use and manipulate dictionaries for the given problem.

Learning Objectives...

- To learn Concepts of Lists like Defining, Accessing, Deleting, Updating and so on
- To understand Basic List Operations and Built-in List Functions
- To Study Concept of Tuples with Accessing, Deleting, Updating Values in Tuples
- To study Basic Tuple Operations and Built-in Tuple Functions
- To understand Concepts of Sets with Accessing, Deleting, Updating Values in Sets
- To understand Basic Set Operations and Built-in Set Functions
- To know Concepts of Dictionaries with Accessing, Deleting, Updating Values in Dictionary
- To study Basic Dictionary Operations and Built-in Dictionaries Functions

3.0 INTRODUCTION

- A data structure is a specialized format for organizing and storing data, so that various operations can be performed on it efficiently and easily.
- Any data structure is designed to organize data to suit a specific purpose so that it can be accessed and worked with in appropriate and systematic manner/way. There are four data structures in Python namely, list, tuple, dictionary and set.
- A data structure that stores an ordered collection of items in Python is called a list. In other words, a list holds a sequence of items or elements. Similar to a list, the tuple is an ordered sequence of items. A set is an unordered collection of unique items in Python. A dictionary in Python is an unordered collection of key value pairs.
- The data types that are most used in Python are strings, tuples, lists and dictionaries. These are collectively called data structures.

3.1 LISTS

- A list in Python is a linear data structure. The elements in the list are stored in a linear order one after other. A list is a collection of items or elements; the sequence of data in a list is ordered.
- The elements or items in a list can be accessed by their positions i.e. indices. The index in lists always starts with 0 and ends with n-1, if the list contains n elements.

Defining List:

- In Python lists are written with square brackets. A list is created by placing all the items (elements) inside a square brackets [ ], separated by commas.
- **Syntax** for define lists is: `<list_name> = [value1, value2, … valueN]`
  Here, list_name is the name of the list and value1, value2, …. valueN are list of values assigned to the list.
  Example: `Emp = [20, "Amar", 'M', 50, 40]`
- Lists are mutable or changeable. The value of any element inside the list can be changed at any point of time. Each element or value that is inside of a list is called an item.
- A list data type is a flexible data type that can be modified to the requirement, which makes it a mutable data type. It is also a dynamic data type. Lists can contain any sort of object. It can be numbers, strings, tuples and even other lists.

3.1.1 Creating a List

- The simplest method to create a list by simply assigning a set of values to the list using the assignment operator (=).
- We can create a list by placing a comma separated sequence of values in square brackets[].

Example: Creating an empty list.

```
>>> l1=[]          # Empty list
>>> l1             # display l1
[]
>>> l1=list()
>>> l1
[]
```

Example: Creating a list with any integer elements.

```
>>> l2=[10,20,30]   # List of Integers
>>> l2
[10, 20, 30]
>>> l2=list([10,20,30])
>>> l2
[10, 20, 30]
```

Example: Creating alist with string elements.

```
>>> l3=["Mango","Orange","Banana"]    # List of Strings
>>> l3
['Mango', 'Orange', 'Banana']
>>> l3=list(["red","yellow","green"] )
>>> l3
['red', 'yellow', 'green']
```

Example: Creating alist with mixed data.

```
>>> l4=[1,"Two",11.22,'X']  #List of mixed data types
>>> l4
[1, 'Two', 11.22, 'X']
```

- You can convert other data types to lists using Python's list() constructor.

Example: Create a list using inbuilt range() function.

```
>>> l5=list(range(0,5))
>>> l5
```

```
[0, 1, 2, 3, 4]
```

Example: Create a list with in-built characters A, B and C.

```
>>> l6=list("ABC")
>>> l6
['A', 'B', 'C']
```

3.1.2 Accessing Values in List

- Accessing elements liters from a list in Python is a method to get values that are stared in the list at a particular location or index.
- To access values in lists, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example: accessing list values.

```
>>> list1 = ["one","two",3,10,"six",20]
>>> list1[0]            # positive indexing
'one'
>>> list1[-2]           # negative indexing
'six'
>>> list1[1:3]          # get element from mth index to n-1 index
['two', 3]
>>> list1[3:]           # get element from mth index to last index
[10, 'six', 20]
>>> list1[:4]           # get element from 0th index to n-1 index
['one', 'two', 3, 10]
>>>
```

- Accessing elements from a particular list in Python at once allows the user to access all the values from the lists. This is possible by writing the list name in the print statement.

3.1.3 Deleting Values in List

- Python provides many ways in which the elements in a list can be deleted.

1. The pop() method in Python is used to remove a particular item/element from the given index in the list. The pop() method removes and returns the last item if index is not provided. This helps us implement lists as stacks (first in, last out data structure).

Example: for pop() method.

```
>>> list= [10, 20, 30, 40]
>>> list
[10, 20, 30, 40]        # pop with index
30
>>> list
[10, 20, 40]
>>> list.pop()          # pop without index
40
>>> list
[10, 30]
```

2. We can delete one or more items from a list using the keyword del. It can even delete the list entirely. But it does not store the value for further use.

Example: Using del.

```
>>> list= [10, 20, 30, 40]
```

```
>>> list
[10, 20, 30, 40]
>>> del (list[1])       # del() with index
>>> list
[10, 30, 40]
>>> del list[2]         # del with index
>>> list
[10, 30]
>>> del list            # del without index
>>> list
<class 'list'>
```

3. The remove() method in Python issued to remove a particular element from the list. We use the remove() method if we know the item that we want to remove or delete from the list (but not the index).

Example: For remove() method.

```
>>> list=[10,"one",20,"two"]    # heterogeneous list
>>> list.remove(20)             # remove element 20
>>> list
[10, 'one', 'two']
>>> list.remove("one")          # remove element one
>>> list
[10, 'two']
>>>
```

3.1.4 Updating Lists (Change or Add Elements to a List)

- List are mutable, meaning their elements can be changed or updated unlike string or tuple.
- Mutability is the ability for certain types of data to be changed without entirely recreating it. Using mutable data types can allow programs to operate quickly and efficiently.
- Multiple values can be added into list. We can use assignment operator (=) to change an item or a range of items.
- We can update items of the list by simply assigning the value at the particular index position. We can also remove the items from the list using remove() or pop() or del statement.

Example: For updating lists.

```
>>> list1= [10, 20, 30, 40, 50]
>>> list1
[10, 20, 30, 40, 50]
>>> list1[0]=0                  # change 0th index element
>>> list1
[0, 20, 30, 40, 50]
>>> list1[-1]=60                # change last index element
>>> list1
[0, 20, 30, 40, 60]
>>> list1[1]=[5,10]             # change 1st index element as sublist
>>> list1
[0, [5, 10], 30, 40, 60]
>>> list1[1:1]=[3,4]            # add elements to a list at the desired location
>>> list1
```

```
[0, 3, 4, [5, 10], 30, 40, 60]
```

- Following table shows the list methods used for updating list.

Sr. No.	Method	Syntax	Argument Description	Return Type
1.	append()	list.append(item)	The item can be numbers, strings, another list, dictionary etc.	Only modifies the original list. It does not return any value.
2.	extend()	list1.extend(list2)	The extend() method takes a list and adds it to the end.	Only modifies the original list. It doesn't return any value.
3.	insert()	list.insert(index, element)	The index position where an element needs to be inserted element - the is the element to be inserted in the list.	It does not return anything; returns None.

- Let use see above methods in detail:

1. append() Method: The append() method adds an element to the end of a list. We can insert a single item in the list data time with the append().

Example: For append() method.

```
>>> list1=[10,20,30]
>>> list1
[10, 20, 30]
>>> list1.append(40)           # add element at the end of list
>>> list1
[10, 20, 30, 40]
```

2. extend() Method: The extend() method extends a list by appending items. We can add several items using extend() method.

Example: Program for extend() method.

```
>>>list1=[10, 20, 30, 40]
>>>list1
[10, 20, 30, 40]
>>> list1.extend([60,70])      #add elements at the end of list
>>> list1
[10, 20, 30, 40, 60, 70]
```

3. insert() Method: We can insert one single item at a desired location by using the method insert() or insert multiple items by squeezing it into an empty slice of a list.

Example: Program for insert() method.

```
>>> list1=[10, 20]
>>>list1
[10,20]
>>> list1.insert(1,30)
>>> list1
[10, 30, 20]
```

4. The concatenation operation is Python programming used combine the elements/items or two lists. We can also use + operator to combine two lists.

Example: Using + operator to combine with list.

```
>>> list1=[10,20,30]
>>> list1
[10, 20, 30]
>>> list1 + [40,50,60]          # using + to combine two lists
[10, 20, 30, 40, 50, 60]
>>> list2=["A","B"]
>>> list1 + list2
[10, 20, 30, 'A', 'B']
```

5. Sometimes, there is a need or requirement to repeat all the items of the lists as precise number of times. The * operator repeats a list for the given number of times.

Example: Using * operator with list.

```
>>> list2=['A', 'B']
>>>list2
['A', 'B']
>>> list2 *2                    #using * to repeat a list
['A', 'B', 'A', 'B']
```

6. **sort() Method:** It arranges the list items in ascending order or descending order. The sort () is called by a list items and sort the list by default in ascending order.

Example: For sort() method.

```
>>> list=[4,2,5,1,7,3]
>>> list.sort()
>>> list
[1, 2, 3, 4, 5, 7]
```

3.1.5 Basic List Operations (indexing and slicing)

- The operations on list include indexing and slicing.

3.1.5.1 Indexing

- An individual item in the list can be referenced by using an index, which is an integer number that indicates the relative position of the item in the list.
- There are various ways in which we can access the elements of a list some as them are given below:

1. **List Index:** We can use the index operator [] to access an item in a list. Index starts from 0. So, a list having 5 elements will have index from 0 to 4.

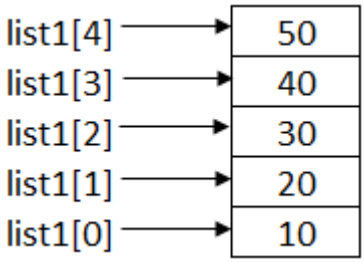

Fig. 3.1: List has Five Elements with Index 0 to 4

Example: For list index in list.

```
>>> list1=[10,20,30,40,50]
>>> list1[0]
```

```
10
>>> list1[3:]    # list[m:] will return elements indexed from m^th index to last index
[40, 50]
>>>list1[:4]     #  list[:n]  will  return  elements  indexed  from  first  index  to
                    n-1^th index
[10, 20, 30, 40]
>>> list1[1:3]       # list[m:n] will return elements indexed from m to n-1.
[20, 30]
>>> list1[5]
Traceback (most recent call last):
 File "<pyshell#71>", line 1, in <module>
  list1[5]
IndexError: list index out of range
```

2. **Negative Indexing:** Python allows negative indexing for its sequences. The index of –1 refers to the last item, –2 to the second last item and so on.

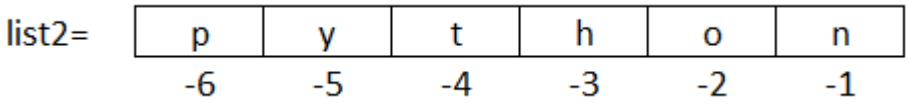

Fig. 3.2: List with Negative Index

Example: For negative indexing in list.

```
>>> list2=['p','y','t','h','o','n']
>>> list2[-1]
'n'
>>> list2[-6]
'p'
>>> list2[-3:]
['h', 'o', 'n']
>>> list2[-7]
Traceback (most recent call last):
 File "<pyshell#76>", line 1, in <module>
  list2[-7]
IndexError: list index out of range
```

3.1.5.2 List Slicing

- Slicing is an operation that allows us to extract elements from units. The slicing feature used by Python to obtain a specific subset or element of the data structure using the colon (:) operator.
- The slicing operator returns a subset of a list called slice by specifying two indices, i.e. start and end.

  Syntax: `list_variable[start_index:end_index]`
- This will return the subset of the list starting from start_index to one index less than that of the end index.

Example: For slicing list.

```
>>> l1=([10,20,30,40,50])
>>> l1[1:4]
[20, 30, 40]
>>>l1[2:5]
[30,40,50]
```

List Slicing with Step Size:

- Step is the integer value which determines the increment between each index for slicing.

Syntax: `list_name[start_index:end_index:step_size]`

Example: For slicing operation in list.

```
>>> l1=['Red', 1, 'yellow', 2, 'Green', 3, 'Blue', 4]
>>>l1
['Red', 1, 'yellow', 2, 'Green', 3, 'Blue', 4]
>>> l2=l1[0:6:2]
>>> l2
['Red', 'yellow', 'Green']
```

Example: Complex example of list slicing.

```
>>> l1                 # list of five elements
[10, 20, 30, 40, 50]
>>> l1[::-1]           # display list in reverse order
[50, 40, 30, 20, 10]
>>> l1[-1:0:-1]        # start index with -1 and end index with o and step size with -1
[50, 40, 30, 20]
```

Traversing a List:

- Traversing a list means accessing all the elements or items of the list. Traversing can be done by using any conditional statement of Python, but it is preferable to use for loop.

Example 1:

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

Output:

```
10
20
30
40
```

Example 2:

```
list1=[[1,2,3,4],['A','B','C','D'],['@','#','$','&']]
for i in list1:
  for j in i:
    print(j,end=' ')
  print('\n')
```

Output:

```
1 2 3 4
A B C D
@ #$ &
```

3.1.6 Built-in Functions and Methods for list

- Python provide certain method and function work with list. Following table shows different functions with their description and example.

Sr. No.	Function	Description	Example
1.	`len(list)`	It returns the length of the list.	`>>> list1`

			[1, 2, 3, 4, 5] >>> len(list1) 5

2.	max(list)	It returns the item that has the maximum value in a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> max(list1)
			5

3.	sum(list)	Calculates sum of all the elements of list.	>>>list1 [1, 2, 3, 4, 5] >>>sum(list1) 15

4.	min(list)	It returns the item that has the minimum value in a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> min(list1)
			1

5.	list(seq)	It converts a tuple into a list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list(list1)
			[1, 2, 3, 4, 5]

Methods in Lists:

Sr. No.	Function	Description	Example

1.	list.append(item)	It adds the item to the end of the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list1.append(6)
			>>> list1
			[1, 2, 3, 4, 5, 6]

2.	list.count(item)	It returns number of times the item occurs in the list.	>>> list1
			[1, 2, 3, 4, 5, 6, 3]
			>>> list1.count(3)
			2

3.	list.extend(seq)	It adds the elements of the sequence at the end of the list.	>>> list1
			[1, 2, 3, 4, 5]
			>>> list2
			['A', 'B', 'C']
			>>>
			list1.extend(list2)
			>>> list1
			[1, 2, 3, 4, 5, 'A', 'B', 'C']

4.	list.index(item)	It returns the index number of the item. It item appears one than one time, it returns the lowest index number.	>>>
			list1=[1,2,3,4,5,3]
			>>> list1
			[1, 2, 3, 4, 5, 3]
			>>> list1.index(3)

			2

5.	`list.insert(index,item)`	It inserts the given item onto the given index number while the elements in the list take one right shift.	`>>> list1`
			`[1, 2, 3, 4, 5, 3]`
			`>>> list1.insert(2,7)`
			`>>> list1`
			`[1, 2, 7, 3, 4, 5, 3]`

6.	`list.pop(item=list[-1])`	It deletes and returns the last element of the list.	`>>> list1`
			`[1, 2, 7, 3, 4, 5, 3]`
			`>>> list1.pop()`
			`3`
			`>>> list1.pop(2)`
			`7`

7.	`list.remove(item)`	It deletes the given item from the list.	`>>> list1`
			`[1, 2, 3, 4, 5]`
			`>>> list1.remove(3)`
			`>>> list1`
			`[1, 2, 4, 5]`

8.	`list.reverse()`	It reverses the position (index number) of the items in the list.	`>>> list1`
			`[1, 2, 3, 4, 5]`
			`>>> list1.reverse()`
			`>>> list1`
			`[5, 4, 3, 2, 1]`

9.	`list.sort()`	Sorts items in the list.	`>>> list1`
			`[1, 3, 2, 4, 5]`
			`>>> list1.sort()`
			`>>> list1`

9.	`list.sort([func])`	It sorts the elements inside the list and uses compare function if provided.	`>>> list1`
			`[1, 3, 2, 5, 4]`
			`>>> list1.sort()`
			`>>> list1`
			`[1, 2, 3, 4, 5]`

3.2 TUPLES

- A tuple also a linear data structure. A python tuple is a sequence of data values called on items or elements. A tuple is a collection itms which is ordered and unchangeable.
- A tuple is a data structure that is an immutable or unchangeable, ordered sequence of elements/items. Because tuples are immutable, their values cannot be modified.
- Tuples heterogeneous data structure and used for grouping data. Each element or value that is inside of a tuple is called an item.
- A tuople is an immutable data type. An immutable data structure means that we cannot add or remove items from the tuple data structure.
- In Python tuples are written with round brackets () and values in tuples can also be accessed by their index values, which are integers starting from 0.
- Tuples are the sequence or series values of different types separated by commas (,). Tuples are just like lists, but you can not change their values.

Difference between Tuples and Lists:

1. A values in the list can be replaced with another any time after its creation, whereas in tuples, the values in it cannot be replaced with another, once tuples are created.
2. Lists allows us to add new items to it, but tuple does not allow us to add new items, once it is created.
3. We generally use tuple for heterogeneous (different) datatypes and list for homogeneous (similar) data types.
4. Since, tuple are immutable, iterating through tuple is faster than with list. So there is a slight performance boost.
5. Tuples that contain immutable elements can be used as key for a dictionary. With list, this is not possible.
6. Tuples can be used as values in sets whereas lists can not.
7. Some tuples can be used as dictionary keys (specifically, tuples that contain immutable values like strings, numbers, and other tuples). Lists can never be used as dictionary keys, because lists are not immutable.
8. Tuples have structure and list have order.

Immutable (Value cannot be modified) **Mutable (values can be modified)**

Sr. No.	Strings str="hi"	Tuples tuples=(5,4.0,'a')	List list=[5,4.0','a']
1.	Sequence Unicode character.	Ordered sequence.	Order sequence.
2.	Values cannot be modified.	Same as list but it is faster than list because it is immutable.	Value can be changed dynamically.
3.	It is a sequence of character.	Values stored in alpha numeric.	Values stored in alpha numeric.
4.	Access values from string.	Access values from tuples.	Access values from list.
5.	Adding values in not possible.	Adding values is not possible.	Adding values is possible.
6.	Removing values is not possible.	Removing values is not possible.	Removing values is possible.

3.2.1 Creating Tuple

- To create tuple, all the items or elements are placed inside parentheses ( ) separated by commas and assigned to a variable.
- The **syntax** for defining a tuple in Python is: `<tuple_name> = (value1, value2, ... valueN)`.
  Here help name indicate name as the tuple and value1, value2,...valueN are the values assigned to the tuple.
  Example: `Emp (20, "Amar", 'M', 50, 40)`
- A tuple in Python is an immutable data type which means a tuple once created cannot be altered/modified. Tuples can have any number of different data items (integer, float, string, list etc).
- The simplest method Python is simply set of values to the tuple using assignment operator (=). For example: t( ) # creates an empty tuple with t name.

Example: For creating tuples.

```
>>> tuple1=(10,20,30)                    # A tuple with integer values
>>> tuple1
(10, 20, 30)
>>> tuple2=(10,"abc",11.22,'X')          # A tuple with different data types
>>> tuple2
```

```
(10, 'abc', 11.22, 'X')
>>> tuple3=("python",[10,20,30],[11,"abc",22.33]) # Nested tuple
>>> tuple3
('python', [10, 20, 30], [11, 'abc', 22.33])
>>> tuple4=10,20,30,40,50                        # Tuple can be created without
                                                 parenthesis
>>> tuple4
(10, 20, 30, 40, 50)
>>> type(tuple4)
<class 'tuple'>
```

Note: In case generating a tuple with a single element, make sure to add a comma after the element otherwise it would not be considered as tuple.

```
>>> tuple=10
>>> type(tuple)
<class 'int'>
>>> tuple="hello"
>>> type(tuple)
<class 'str'>
>>> tuple=("hello",)
>>> type(tuple)
<class 'tuple'>
```

Tuple Assignment:

- It allows the assignment of values to a tuple of variables on the left side of the assignment from the tuple of values on the right side of the assignment.
- The number of variables in the tuple on the left of the assignment must match the number of elements/items in the tuple on the right of the assignment.

Example 1: For tuple assignment.

```
>>> vijay=(11,"Vijay","Thane")
>>> (id,name,city)=vijay
>>> print(id)
11
>>> print(name)
Vijay
```

Example 2:

```
>>> language=('python','java')
>>> language
('python', 'java')
>>> (a,b)=language
>>> a
'python'
>>> b
'java'
```

- Similarly swapping of two values of variables can be solve by using tuple assignment:

Swapping the values of two variables using traditional method:

```
>>> a=10
```

Swapping the values of two variables using tuple assignment:

```
>>> x=10
```

```
>>> b=20
>>> print(a,b)
10 20
>>> temp=a
>>> a=b
>>> b=temp
>>> print(a,b)
20 10
```

```
>>> y=20
>>> print(x,y)
10 20
>>> x,y=y,x
>>> print(x,y)
20 10
```

3.2.2 Accessing Values in Tuple

- Accessing items/element from the tuple is a method to get values stored in the tuple from a particular location or index.
- To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index.

Example: For accessing values in tuples.

```
>>> tuple=(10,20,30,40,50)
>>> tuple[1]         #access value at specific index
20
>>> tuple[1:4]       #tuple[m:n] will return elements from mth index to n-1th index.
(20, 30, 40)
>>> tuple[:2]        #tuple[:n] will return elements from 0th index to n-1th index.
(10, 20)
>>> tuple[2:]        #tuple[m:] will return elements from mth index to last index.
(30, 40, 50)
>>> tuple[-1]        #access value at last index
50
```

- Accessing elements/items from a particular tuple at once allows the user to access all the values from the tuple only by writing a single statement. This is possible by writing the tuple name in the print statement.

3.2.3 Deleting Tuples

- Tuples are unchangeable, so we cannot remove items from it, but we can delete the tuple completely. To explicitly remove an entire tuple, just use the del statement.

Example: Delete entire tuple.

```
>>> t1
(10, 20)
>>> del t1
>>> t1
Traceback (most recent call last):
 File "<pyshell#65>", line 1, in <module>
  t1
NameError: name 't1' is not defined
```

- If we want to remove elements from a tuple, we can use index slicing to leave out a particular index.

Example: Leave out elements from tuple.

```
a = (1, 2, 3, 4, 5)
b = a[:2] + a[3:]   #element 3 is leave out
print(b)
```

Output:

```
(1, 2, 4, 5)
```

- Or we can convert tuple into a list, remove the item and convert back to a tuple.

Example:

```
tuple1 = (1, 2, 3, 4, 5)
list1 = list(tuple1)  #convert tuple to list
del list1[2]
b = tuple(list1)    #convert list to tuple
print(b)
```

Output:

```
(1, 2, 4, 5)
```

3.2.4 Updating Tuple

- Tuples are immutable which means we cannot update or change the values of tuple elements. We are able to take portions of existing tuples to create new tuples.

Example: For updating tuple.

```
>>> tuple1
(10, 20, 30)
>>> tuple2
('A', 'B', 'C')
>>> tuple1[1]=40                        # get error as tuples are immutable
Traceback (most recent call last):
 File "<pyshell#39>", line 1, in <module>
  tuple1[1]=40
TypeError: 'tuple' object does not support item assignment
>>> tuple3=tuple1+tuple2               # concatenating two tuples
>>> tuple3
(10, 20, 30, 'A', 'B', 'C')
```

- If the element of tuple is itself a mutable data type like list, its nested items can be changed as shown in following example:

```
>>> tuple1 = (1,2,3,[4,5])
>>> tuple1[3][0]= 14
>>> tuple1[3][1]=15
>>> tuple1
(1, 2, 3, [14, 15])
```

- In order to change a value, we can convert tuple into list and then change the values and revert back list into tuple can solve the purpose of update as shown in following example.

Example:

```
tuple1 = (1, 2, 3, 4, 5)
list1 = list (tuple1)
list1[2]=20
b = tuple (list1)
print(b)
```

Output:

```
(1, 2, 20, 4, 5)
```

- Like updation or addition operations we can also perform deletion operation an tuple.

3.2.5 Tuple Operations

- A set of operations can be apply to Python tuple. The operations described below are supported by most sequence types, both mutable and immutable. Here we will consider different operations in context of immutable sequence.

1. Concatenation and Repetition:

- We can use + operator to combine two tuples. This is also called concatenation operation.
- We can also repeat the elements in a tuple for a given number of times using the * operator.

Example: For tuple operations using + and * operators.

```
>>> t1=(10,20)
>>> t2=(30,40)
>>> t1+t2
(10, 20, 30, 40)
>>> t1*2
(10, 20, 10, 20)
```

2. Membership Function:

- We can test if an item exists in a tuple or not, using the keyword in. The in operator evaluates to true if it finds a variable in the specified sequence and false otherwise.

Example: For membership function in tuple.

```
>>> tuple
(10, 20, 30, 40, 50)
>>> 30 in tuple
True
>>> 25 in tuple
False
```

3. Iterating through a Tuple:

- Iteration over a tuple specifies the way which the loop can be applied to the tuple.
- Using a for loop we can iterate through each item in a tuple. Following example uses a for loop to simply iterates over a tuple.

Example: For iterating items in tuple using for loop.

```
>>> tuple=(10,20,30)
>>> for i in tuple:
    print(i)        #use two enter to get the output
```

Output:

```
10
20
30
>>>
```

3.2.6 Build-in Tuple Functions

- Following table built-in tuple function in Python programming.

Sr. No.	Function	Description	Example

1.	`cmp(tuple1, tuple2)`	Compares elements of both tuples.	`>>> tup1=(1,2,3)`
			`>>> tup2=(1,2,3)`
			`>>> cmp(tup1,tup2)`
			`0`

2.	`len(tuple)`	Gives the total length of the tuple.	`>>> tup1`
			`(1, 2, 3)`
			`>>> len(tup1)`
			`3`

3.	`max(tuple)`	Returns item from the tuple with max value.	`>>> tup1`
			`(1, 2, 3)`
			`>>> max(tup1)`
			`3`

4.	`min(tuple)`	Returns item from the tuple with min value.	`>>> tup1`
			`(1, 2, 3)`
			`>>> min(tup1)`
			`1`

5.	`count()`	Returns the number of times a specified value occurs in a tuple	`>>> tup1`
			`(1, 2, 3, 2, 4)`
			`>>> tup1.count(2)`
			`2`

6.	`zip(tuple1,tuple2)`	It zips elements from two tuples into a list of tuples.	`>>> tup1=(1,2,3)`
			`>>> tup2=('A','B','C')`
			`>>> tup3=zip(tup1,tup2)`
			`>>> list(tup3)`
			`[(1, 'A'), (2, 'B'), (3, 'C')]`

7.	`index()`	Searches the tuple for a specified value and returns the position of where it was found	`>>> tup1`
			`(1, 2, 3)`
			`>>> tup1.index(3)`
			`2`

8.	`tuple(seq)`	Converts a list into tuple.	`>>>tuple1 = (1, 2, 3, 4, 5)`
			`>>>list1 = list (tuple1)`
			`>>>list1`
			`[1,2,3,4,5]`

3.3 SETS

- The set data structure in Python programming is implemented to support mathematical set operations.
- Set is an unordered collection of unique items. Items stored in a set are not kept in any particular order.
- A set data structure in python programming includes an unordered collection of items without duplicates. Sets are unindexed that means we cannot access set items by referring to an index.
- Sets are changeable (mutable) i.e., we can be changed or update a set as and when required. Type of elements in a set need not be the same, various mixed up data type values can also be passed to the set.
- The sets in python are typically used for mathematical operations like union, intersection, difference and complement etc.

3.3.1 Accessing Values in Sets

- A set is used to contain an unordered collection of items. There are two ways for creation of sets in python:
  1. Set is defined by values separated by comma inside braces { }. Items in a set are not ordered.

Syntax for defining set is: `<set_name>={value1, value2,….,valueN}`

Example: For creating sets with { }.

```
>>> a={1,3,5,4,2}
>>> print("a=",a)
a= {1, 2, 3, 4, 5}
>>> print(type(a))
<class 'set'>
>>> colors = {'red', 'green', 'blue,’red’'}
>>> colors
{'blue', 'red', 'green'}
>>>
```

- A set does not contain any duplicate values or elements. We can perform set operations like union, intersection on two sets. Set have unique values. They eliminate duplicates.

Example:

```
>>> a={1,2,2,3,3,3}
>>> a
{1, 2, 3}
>>>
```

2. Set can be create by calling a type constructor called set().

Example: Creating sets with set().

```
>>> s=set('abc')
>>> s
{'c', 'b', 'a'}
>>> s=set(range(1,3))
>>> s
{1, 2}
>>>
```

3.3.2 Deleting Values in Set

- There are different ways for deletion of sets in Python programming. Some of them are given below:
  1. Remove elements from a set by using discard() method.
  2. Remove elements from a set by using remove() method. The only difference between remove and discard method is that If specified item is not present in a set then remove() method raises KeyError.
  3. pop() method removes random item from a set and returns it.
  4. clear() method remove all items from the set.

Example: For deleting values in sets.

```
>>> b={"a","b","c"}
>>> b
{'c', 'b', 'a'}
>>> b.discard("b")
>>> b
{'c', 'a'}
>>> b.remove("a")
>>> b
{'c'}
```

```
>>> b={"a","b","c"}
>>> b.pop()
'c'
>>> b.clear()
>>> b
set()
```

3.3.3 Updating Set

- We can update a set by using add() method or update() method.

1. Using add() Method:

- We can add elements to a set by using add() method. The () method takes one argument which is the element we want to add in set.
- At a time only one element can be added to the set by using add() method, loops are used to add multiple elements at a time with the use of add() method.

Syntax: `A.add(element)`

Here, A is set which will be updated by given element.

Example: For add() method which updates sets.

```
>>> a={1,2,3,4}
>>> a.add(6)
>>> print(a)
{1, 2, 3, 4, 6}
>>> a={2,4,6,8}
>>> a.add(3)
>>> print(a)
{2, 3, 4, 6, 8}
>>>
```

2. Using update() Method:

- The update() adds elements from lists/strings/tuples/sets to the set.
- The update() method can accepts lists, strings, tuples as well as other sets as its arguments. In all of these cases, duplicate elements are avoided.

Syntax: `A.update(B)`

Here, B can be lists/strings/tuples/sets and A is set which will be updated without duplicate entries.

Example: For update() method which updates sets.

```
>>> a={1,2,3}
>>> b={'a','b','c'}
>>> a
{1, 2, 3}
>>> b
{'c', 'b', 'a'}
>>> a.update(b)
>>> a
{1, 2, 3, 'b', 'a', 'c'}
>>> a.update({3,4})
>>> a
{1, 2, 3, 'b', 4, 'a', 'c'}
```

3.3.4 Basic Set Operations

- Union, Intersection, Difference and Symmetric Difference are some operations which are performed on sets.

1. Union:

- Union operation performed on two sets returns all the elements from both the sets.
- The union of A and B is defined as the set that consists of all elements belonging to either set A or set B (or both).It is performed by using | operator.

Example: For union operation in sets.

```
>>> A={1,2,4,6,8}
>>> B={1,2,3,4,5}
>>> C=A | B
>>> C
{1, 2, 3, 4, 5, 6, 8}
```

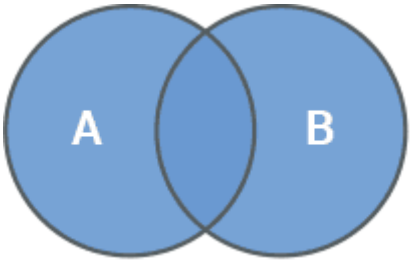

Fig. 3.3: Union Operation

2. Intersection:

- Intersection operation performed on two sets returns all the elements which are common or in both the sets.
- The intersection of A and B is defined as the set composed of all elements that belong to both A and B. It is performed by using & operator.

Example: For intersection operation in sets.

```
>>> A={1,2,4,6,8}
>>> B={1,2,3,4,5}
>>> C=A & B
>>> C
{1, 2, 4}
```

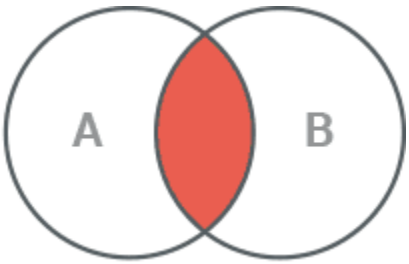

Fig. 3.4: Intersection operation

3. Difference:

- Difference operation on two sets set1 and set2 returns all the elements which are present on set1 but not in set2. It is performed by using – operator.

Example: For difference operation in sets.

```
>>> A={1,2,4,6,8}
>>> B={1,2,3,4,5}
>>> C=A - B
>>> C
```

```
{8, 6}
```

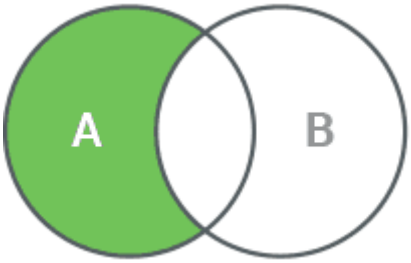

Fig. 3.5: Difference Operation

4. Symmetric Difference:

- Symmetric Difference operation performed by using ^ operation.

Example: For symmetric difference operation in sets.

```
>>> A={1,2,4,6,8}
>>> B={1,2,3,4,5}
>>> C=A ^ B
>>> C
{3, 5, 6, 8}
```

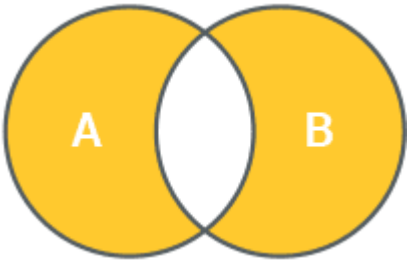

Fig. 3.6: Symmetric Difference Operation

3.3.5 Built-in Set Functions and Methods

Methods in set:

Method	Description	Syntax
`union()`	Return a new set containing the union of two or more sets	`set.union(set1,set2…)`
`intersection()`	Returns a new set which is the intersection of two or more sets	`set.intersection(set1,set2…)`
`intersection_update()`	Removes the items from this set that are not present in other sets	`set.intersection_update(set1,set2…)`
`difference()`	Returns a new set containing the difference between two or more sets	`set.difference(set1,set2…)`
`difference_update()`	Removes the items from this set that are also included in another set	`set.difference_update(set1,set2…)`
`symmetric_difference()`	Returns a new set with the symmetric	`set.symmetric_difference(set)`

	differences of two or more sets	
symmetric_difference_update()	Modify this set with the symmetric difference of this set and other set	set.symmetric_difference_update(set)
isdisjoint()	Determines whether or not two sets have any elements in common	set.isdisjoint(set)
issubset()	Determines whether one set is a subset of the other	set.issubset(set)
issuperset()	Determines whether one set is a superset of the other	set.issuperset(set)
add(item)	It adds an item to the set. It has no effect if the item is already present in the set.	set.add(set)
discard(item)	It removes the specified item from the set.	set.discard(set)
remove(item)	Remove an element from a set; it must be a member. If the element is not a member, raise a KeyError.	set.remove(set)
pop()	Remove and return an arbitrary set element that is the last element of the set. Raises KeyError if the set is empty.	set.pop(set)
update()	Updates the set with the union of itself and others.	set.update(set)

- Built-in functions like all(), any(), enumerate(), len(), max(), min(), sorted(), sum() etc. are commonly used with set to perform different tasks.

Following table lists built-in function of Set

Sr. No.	Function	Description
1.	all()	Return True if all elements of the set are true (or if the set is empty).
2.	any()	Return True if any element of the set is true. If the set is empty, return False.
3.	enumerate()	Return an enumerate object. It contains the index and value of all the items of set as a pair.
4.	len()	Return the length (the number of items) in the set.
5.	max()	Return the largest item in the set.
6.	min()	Return the smallest item in the set.
7.	sorted()	Return a new sorted list from elements in the set(does not sort the set itself).
8.	sum()	Retrun the sum of all elements in the set.

Example: For set functions.

```
A = {'red', 'green', 'blue'}
B = {'yellow', 'red', 'orange'}
print("Union ", A.union(B))
print("Intersection " ,A.intersection(B))
A.intersection_update(B)
print("Intersection Update ", A)
A = {'red', 'green', 'blue'}
print("Difference ",A.difference(B))
A.difference_update(B)
print("Difference Update ",A)
A = {'red', 'green', 'blue'}
print("Symmetric Difference", A.symmetric_difference(B))
A = {'red', 'green', 'blue'}
A.symmetric_difference_update(B)
print("Symmetric Difference Update",A)
A={'red'}
print(A.isdisjoint(B))
print(A.issubset(B))
print(A.issuperset(B))
```

Output:

```
Union {'yellow', 'orange', 'green', 'red', 'blue'}
Intersection {'red'}
Intersection Update {'red'}
Difference {'green', 'blue'}
Difference Update {'blue', 'green'}
Symmetric Difference {'orange', 'blue', 'yellow', 'green'}
Symmetric Difference Update {'blue', 'orange', 'yellow', 'green'}
False
True
False
```

3.4 DICTIONARIES

- The dictionary data structure is used to store key value pairs indexed by keys. A dictionary is an associative data structure, means that elements/items are stored in a non linear fashion.
- Python dictionary is an unordered collection of items or elements. Items stored in a dictionary are not kept in any particular order. The Python dictionary is a sequence as data values called as items or elements.
- While other compound data types have only value as an element, a dictionary has a key:value pair. Each value is associated with a key.
- Dictionaries are optimized to retrieve values when the key is known. A key and its value are separated by a colon (:) between them.
- The items or elements in a dictionary are separated by commas and all the elements must be enclosed in curly braces.

Syntax for define a dictionary is as follows

`<dictionary_name> = {key1:value1, key2:value2…keyN:valueN}`

Example: `Emp = {“ID”: 20, “Name” : “Amar”, “Gender”=Male, “SalaryPerHr”:200}`

- A pair of curly braces with no values in between is known as an empty dictionary. Dictionary items are accessed by keys, not by their position (index).
- The values in a dictionary can be duplicated, but the keys in the dictionary are unique. Dictionaries are changeable (mutable). We can change or update the items in dictionary as and when required.
- A dictionary can be used to store a collection of data values in a way that allows them to be individually referenced. However, rather than using an index to identify a data value, each item in a dictionary is stored as a key value pair.
- The key can be looked up in much the same way that we can look up a word in a paper-based dictionary to access its definition i.e., the word is the ' key' and the definition is its corresponding 'value'.
- Dictionaries can be nested i.e. a dictionary can contain another dictionary.

3.4.1 Creating of Dictionary

- The simplest method to create dictionary is to simply assign the pair of key:values to the dictionary using operator (=).
- There are two ways for creation of dictionary in python.
  1. We can create a dictionary by placing a comma-separated list of key:value pairs in curly braces {}. Each key is separated from its associated value by a colon:

Example: For creating a dictionary using { }.

```
>>> dict1={}                                    #Empty dictionary
>>> dict1
{}
>>> dict2={1:"Orange", 2:"Mango", 3:"Banana"} #Dictionary with integer keys
>>> dict2
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> dict3={"name":"vijay", 1:[10,20]}         #Dictionary with mixed keys
>>> dict3
{'name': 'vijay', 1: [10, 20]}
```

  2. Python provides a build-in function dict() for creating a dictionary.

Example: Creating directory using dict().

```
>>> d1=dict({1:"Orange",2:"Mango",3:"Banana"})
>>> d2=dict([(1,"Red"),(2,"Yellow"),(3,"Green")])
>>> d3=dict(one=1, two=2, three=3)
>>> d1
{1: 'Orange', 2: 'Mango', 3: 'Banana'}
>>> d2
{1: 'Red', 2: 'Yellow', 3: 'Green'}
>>> d3
{'one': 1, 'two': 2, 'three': 3}
```

3.4.2 Accessing Values in a Dictionary

- We can access the items of a dictionary by following ways:
  1. Referring to its key name, inside square brackets([]).

Example: For accessing dictionary items [ ] using.

```
>>> dict1={'name':'vijay','age':40}
>>> dict1['name']
'vijay'
>>> dict1['adr']
Traceback (most recent call last):
 File "<pyshell#79>", line 1, in <module>
```

```
  dict1['adr']
KeyError: 'adr'
>>>
```

Here, if we refer to a key that is not in the dictionary, you'll get an exception.This error can be avoided by using get() method.

2. Using get() method returns the value for key if key is in the dictionary, else None, so that this method never raises a KeyError.

Example: For accessing dictionary elements by get().

```
>>> dict1={'name':'vijay','age':40}
>>> dict1.get('name')
'vijay'
>>> dict1.get('adr')
```

3.4.3 Deleting Elements/Items From Dictionary

- We can remove a particular item in a dictionary by using the method pop(). This method removes as item with the provided key and returns the value.
- The method, popitem() can be used to remove and return an arbitrary item (key, value) form the dictionary.
- All the items can be removed at once using the clear() method. We can also use the del keyword to remove individual items or the entire dictionary itself.

Example: For deleting dictionary items/elements.

```
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> print(squares.popitem())        # remove an arbitrary item
(5, 25)
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16}
>>> squares.pop(2)                  # remove a particular item
4
>>> squares
{1: 1, 3: 9, 4: 16}
>>> del squares[3]                  # delete a particular item
>>> squares
{1: 1, 4: 16}
>>> squares.clear()                 # removes all items
>>> squares
{}
>>> del squares                     # delete a dictionary itself
>>> dquares
Traceback (most recent call last):
 File "<pyshell#29>", line 1, in <module>
  dquares
NameError: name 'dquares' is not defined
```

3.4.4 Updating Dictionary

- The dictionary data type is flexible data type that can be modified according to the requirements which makes it is a mutable data type. The dictionary are mutable means changeable. We can add new items or change the value of existing items using assignment operator.
- If the key is already present, value gets updated, else a new key:value pair is added to the dictionary.

Example: For updating dictionary items/elements.

```
>>> dict1
{'name': 'vijay', 'age': 40}
>>> dict1['age']=35                # updating values in dictionary
>>> dict1
{'name': 'vijay', 'age': 35}
>>> dict1['address']='thane'       # add new item in dictionary
>>> dict1
{'name': 'vijay', 'age': 35, 'address': 'thane'}
```

3.4.5 Basic Operations on Directory

- In previous sections we study basic operations of directory such as create, delete, update etc. other operations that can be applied to the element/items of the directory are explained below:

Dictionary Membership Test:

- We can test if a key is in a dictionary or not using the keyword in. Notice that membership test is for keys only, not for values.

Example: For directory membership test.

```
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> print(1 in squares)
True
>>> print(6 in squares)
False
>>> squares={1:1,2:4,3:9,4:16,5:25}
>>> print(9 in squares)
False
```

Traversing Dictionary:

- Using a for loop we can iterate though each key in a dictionary.

Example: For traversing dictionary.

```
>>> squares
{1: 1, 2: 4, 3: 9, 4: 16, 5: 25}
>>> for i in squares:
    print(i,squares[i])
```

Output:

```
1 1
2 4
3 9
4 16
5 25
```

Properties of Dictionary Keys:

- Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.
- There are two important points to remember about dictionary keys:
  1. More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins.

Example:

```
>>> dict={1:'Vijay',2:'Amar',3:'Santosh',2:'Umesh'}
>>> dict
{1: 'Vijay', 2: 'Umesh', 3: 'Santosh'}
```

2. Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like [‘key’] is not allowed.

Example:

```
>>> dict={[1]:'Vijay',2:'Amar',3:'Santosh'}
Traceback (most recent call last):
 File "<pyshell#2>", line 1, in <module>
  dict={[1]:'Vijay',2:'Amar',3:'Santosh'}
TypeError: unhashable type: 'list'
```

3.4.6 Dictionary Methods and Built-in Functions

- Python has a set of built-in methods that we can use on dictionaries. Some of them are given in following table.

Sr. No.	Method	Description	Example

1.	clear()	Removes all the elements from the dictionary.	dict={1:'Vijay',2:'Amar',3:'Santosh'}
			>>> dict
			{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
			>>> dict.clear()
			>>> dict
			{}

2.	copy()	Returns a copy of the dictionary.	>>> dict={1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
			>>> X=dict.copy()
			>>> X
			{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}

3.	fromkeys()	The fromkeys() method creates a new dictionary with default value for all specified keys.
		If default value is not specified, all keys are set to None.
			>>> dict=dict.fromkeys(['Vijay','Meenakshi'], 'Author')
			>>> dict
			{'Vijay': 'Author', 'Meenakshi': 'Author'}
			>>>

4.	get()	Returns the value of the specified key.	>>> dict1={'name':'vijay','age':40}
			>>> dict1.get('name')
			'vijay'

5.	items()	Returns a list containing the a tuple for each key value pair.	dict{1: 'Vijay', 2: 'Amar', 3: 'Santosh'} >>> for i in dict.items(): print(i) (1, 'Vijay') (2, 'Amar') (3, 'Santosh')
6.	keys()	Returns a list containing the dictionary's keys.	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> dict.keys() dict_keys([1, 2, 3])
7.	pop()	Removes the element with the specified key.	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> print(dict.pop(2))

			Amar
8.	popitem()	Removes the last inserted key-value pair.	dict={1:'Vijay',2:'Amar',3:'Santosh'} >>> dict.popitem() (3, 'Santosh')

9.	setdefault()	Returns the value of the specified key. If the key does not exist: insert the key, with the specified value.	>>> dict
			{2: 'Amar', 3: 'Santosh'}
			>>> dict.setdefault(1,'Vijay')
			>>> dict
			{2: 'Amar', 3: 'Santosh', 1: 'Vijay'}

10.	update()	Updates the dictionary with the specified key-value pairs.	>>> dict1
			{2: 'Amar', 4: 'Umesh'}
			>>> dict2={1:'Vijay',3:'Santosh'}
			>>> dict2
			{1: 'Vijay', 3: 'Santosh'}
			>>> dict1.update(dict2)
			>>> dict1
			{2: 'Amar', 4: 'Umesh', 1: 'Vijay', 3: 'Santosh'}

11.	values()	Returns a list of all the values in the dictionary.	>>> dict
			{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
			>>> dict.values()
			dict_values(['Vijay', 'Amar', 'Santosh'])

Built-in Functions of Directory:

Sr. No.	Function	Description	Example

1.	all()	Return True if all keys of the dictionary are true (or if the dictionary is empty).	>>> dict
			{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
			>>> all(dict)
			True

2.	any()	Return True if any key of the dictionary is true. If the dictionary is empty, return False.	>>> dict={}
			>>> any(dict)
			False

3.	len()	Return the length (the number of items) in the dictionary.	>>> dict
			{1: 'Vijay', 2: 'Amar', 3: 'Santosh'}
			>>> len(dict)
			3

4.	sorted()	Return a new sorted list of keys in the dictionary.	>>> dict1
			{2: 'Amar', 1: 'Vijay', 4: 'Umesh', 3: 'Amar'}
			>>> sorted(dict1)
			[1, 2, 3, 4]

5.	cmp()	Compares two dictionaries based on key and values.	>>> dict1 {2: 'Amar', 1: 'Vijay', 4: 'Umesh', 3: 'Vikas'}

			>>> dict2 {2: 'Amol', 1: 'Vijay', 4: 'Ravi', 3: 'Vikas'} print "Return Value: %d" % cmp (dict1, dict2)
6.	type()	Returns the type of the passed variable.	dict = {'Name': 'Zara', 'Age': 7}; print "Variable Type: %s" % type (dict)

Strings:

- A string is a linear data structure in Python. Strings are the graph as characters.
- In Python, a string type object is a sequence (left-to-right order) of characters. Strings start and end with single or double quotes Python strings are immutable means string can not be modified according to requirement.
- Single and double quoted strings are same and we can use a single quote within a string when it is surrounded by double quote and vice versa.

Example: For string.

```
>>> a='Hello Python'
>>> a
'Hello Python'
>>> a="Hello Python"
>>> a
'Hello Python'
>>> type(a)
<class 'str'>
```

str Class:

- Strings are objects of the str class. We can create a string using the constructor of str class

```
S1=str()            # creates an empty string object
S2=str("Hello")     # creates a string object for Hello
```

- An alternative wat to create a string object is by assigning a string value to a variable.

Example: For str class.

```
S1=""               # create an empty string
S2="Hello"          # equivalent to S2=str("Hello")
```

str Function:

- The str function is used to convert a number into a string.

Example: For str().

```
>>> a=10
>>> type(a)
<class 'int'>
>>>str(a)          # converting int number to string
'10'
```

Python functions for string:

Sr. No.	Function	Example
1.	len() function return the number of characters in a string.	>>> a="PYTHON" >>>len(a) 6
2.	min() function return the smallest character in a string.	>>> a="PYTHON"

		>>> min(a) 'H'

3.	max() function return the largest character in a string.	>>> a="PYTHON"
		>>> max(a)
		'Y'

Deleting Entire String:

- Deletion of entire string is possible with the use of del keyword.

Example: For deletion of string.

```
>>> a="PYTHON"
>>> a
'PYTHON'
>>> del a
>>> a
Traceback (most recent call last):
 File "<pyshell#9>", line 1, in <module>
  a
NameError: name 'a' is not defined
```

+ operator (String Concatenation):

- The concatenation operator (+) is used to join two strings.

Example: For + operator in string.

```
>>> "Hello" + "Python"
'HelloPython'
>>> s1="Hello"
>>> s2="Python"
>>> s1+s2
'HelloPython'
```

*** Operator (String Multiplication)**:

- The multiplication (*) operator is used to concatenate the same string multiple times, it is called repetition operator.

Example: for * operator in string.

```
>>> s1="Hello "
>>> s2=3*s1
>>> s2
'Hello HelloHello '
>>> "**"*5
'**********'
>>> a="**"
>>> a*5
'**********'
>>>
```

String Traversal (Traversing string with for loop and while loop):

- Traversal is a process in which we access all the elements of the string one by one using for and while loop.

Example: Traversing using for loop.

```
>>> s=”Pythin Programming”
```

```
>>> for ch in s:
    print(ch,end="")
```

Output:

```
Python Programming
>>> for ch in range(0,len(s),2):
    print(s[ch],end="")
```

Example: Traversing using while loop.

```
>>> s=”Pythin Programming”
>>> index=0
>>> while index<len(s):
    print(s[index],end="")
    index=index+1
```

Output:

```
Python Programming
```

Immutable Strings:

- Strings are immutable which means that we cannot change any element of a string. If we want to change an element of a string, we have to create a new string.

Example: For immutable string.

```
>>>str="Python"
>>>str
'Python'
>>>str[0]="H"
Traceback (most recent call last):
 File "<pyshell#33>", line 1, in <module>
str[0]="H"
TypeError: 'str' object does not support item assignment
```

- Here, when we try to change the 0^{th} index of string to a character “H”, but the python interpreter generates an error. The solution to this problem is to generate a new string rather than change the old string.

Example:

```
>>>str="Python"
>>> str1='H'+str[1:]
>>> str1
'Hython'
```

- Consider the following two similar strings:

```
Str1=”Python”
Str2=”Python”
```

Here, Str1 and Str2 have the same content. Thus python uses one object for each string which has the same content. Both Str1 and St2 refers to the same string object, whereas Str1 and Str2 have the same ID number.

Example:

```
>>> Str1="Python"
>>> Str2="Python"
>>> id(Str1)
54058464
>>> id(Str2)
```

```
54058464
```

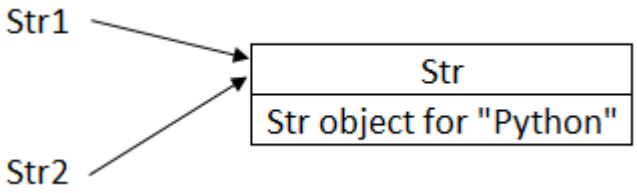

Fig. 3.7

String Indices and Accessing String Elements:

- Strings are arrays of characters and elements of an array can be accessed using indexing. Indices start with 0 from left side and –1 when starting from right side.

```
S1="Hello Python"
```

Character	H	e	l	l	O		P	y	t	h	o	n
Index (from left)	0	1	2	3	4	5	6	7	8	9	10	11
Index (from right)	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

Example:

```
>>> s1="Hello Python"
>>> print(s1[0])          # print first character
H
>>> print(s1[11])         # print last character
n
>>> print(s1[-12])        # print first character
H
>>> print(s1[-1])         # print last character
n
>>> print(s1[15])         # out of index range
Traceback (most recent call last):
 File "<pyshell#65>", line 1, in <module>
  print(s1[15])
IndexError: string index out of range
>>>
```

'in' and 'not in' Operator in Strings:

- The 'in' operator is used to check whether a character or a substring is present in a string or not. The expression returns a Boolean value.

Example: For in and out operators.

```
>>> s1="Hello Python"
>>> "Hello" in s1
True
>>> "java" in s1
False
>>> "Hello" not in s1
False
>>> 'H' in s1
True
>>> 'B' in s1
False
```

```
>>> 'Py' in s1
True
>>>
```

String Slicing:

- A piece or subset of a string is known as slice. To cut a substring from a string is called string slicing.
- Slice operator is applied to a string with the use of square braces([]). Operator [n:m] will give a substring which consists of letters between n and m indices, including letter at index n but excluding that at m, i.e. letter from nth index to (m-1)th index.
- Here two indices are used separated by a colon (:). A slice 3:7 means indices characters of 3rd, 4th, 5th and 6th positions. The second integer index i.e. 7 is not included. You can use negative indices for slicing.

Syntax:

```
stringname[start_index:end_index]
stringname[start_index:end_index:step_size]
```

Example:

```
>>s1=”Hello Python”
>>> s1[0:4]
'Hell'
>>> s1[:2]
'He'
>>> s1[2:]
'llo Python'
>>> s1[0:len(s1):2]
'HloPto'
>>> s1[-1:0:-1]
'nohtyPolle'
>>> s1[:-1]
'Hello Pytho'
```

String Comparison:

- Operators such as ==,<,>,<=,>= and != are used to compare the strings.

Example: For string competition.

```
>>> S1="abcd"
>>> s2="abcd"
>>> s1>s2
True
```

- **Note:** Python compares the numeric value of each character. The ASCII value of ‘a’ is 97 and ASCII numeric value of ‘A’ is 65. It means 97>65, thus it returns True. However, character by character comparison goes on till the end of the string.

```
>>> s1="abcd"
>>> s2="abcd"
>>> s1==s2
True
>>>s1="abcd"
>>> s2="defg"
>>> s1>s2
False
```

Special Characters in String:

- The backslash (\) character is used to introduce a special character or escape character. It converts difficult-to-type characters into a string.

Sr. No.	Escape Character	Meaning	Example
1.	\ newline	Ignored	>>>print("line1\ line2\ line3") line1line2line3
2.	\\	Backslash (\)	>>>print("This is a backslash (\\) mark") This is a backslash (\) mark
3.	\’	Single Quote (‘)	>>>print("These are \'single quote\'") These are 'single quote'
4.	\”	Double Quote (“)	>>>print("These are \"double quote\"") These are "double quote"
5.	\a	ASCII Bell (BELL)	>>> print("\a")
6.	\b	ASCII Backspace (BS)	>>>print("Hello \b World!") Hello World!
7.	\f	ASCII Formfeed (FF)	>>>print("Hello \f World!") Hello World!
8.	\n	ASCII Linefeed (LF), Newline	>>>print("This is a first line \n This is a second line") This is a first line This is a second line
9.	\r	ASCII Carriage return (CR)	>>>print("Hello \r World!") Hello World!
10.	\t	ASCII Horizontal tab (TAB)	>>>print("This is tav \t key") This is tav key
11.	\v	ASCII Vertical tab (VT)	>>>print("Hello \v World!") Hello World!
12.	\ooo	ASCII Character with octal value ooo	>>> print("\110\145\154\154\157\40\127\157\162\154\144\41") Hello World!
13.	\xhhh…	ASCII Character with hex value hh…	>>> print("\x48\x65\x6c\x6c\x6f\x20\x57\x6f\x72\x6c\x64\x21") Hello World!

String Formatting Operator:

- The string in python have a unique built-in operation: the % operator (modulo). This is also called the string formatting operator. This operator is unique to strings and makes up for the pack of having functions from C's printf() family.

  Example:

  ```
  >>>print("My name is %s and weight is %d kg!"%('Vijay',60))
  ```

```
My name is Vijay and weight is 60 kg!
```

Sr. No.	Format Symbol	Conversion
1.	%c	Character.
2.	%s	string conversion via str() prior to formatting.
3.	%i	signed decimal integer.
4.	%d	signed decimal integer.
5.	%u	unsigned decimal integer.
6.	%o	octal integer.
7.	%x	hexadecimal integer (lowercase letters).
8.	%X	hexadecimal integer (UPPERcase letters).
9.	%e	exponential notation (with lowercase 'e').
10.	%E	exponential notation (with UPPERcase 'E').
11.	%f	floating point real number.
12.	%g	the shorter of %f and %e.
13.	%G	the shorter of %f and %E.

String Formatting Functions:

- Python includes the following built-in functions to manipulate strings.

Sr. No.	Method	Description	Example

1.	`capitalize()`	Makes the first letter of the string capital.	`>>> s1="python programming"`
			`>>> s1.capitalize()`
			`'Python programming'`

2.	`center(width, fillchar)`	Returns a space padded string with the original string centered to a total width columns.	`>>> s1="python programming"`
			`>>> print(s1.center(30,'*'))`
			`******python programming******`

3.	`count(str, beg, end)`	Counts the number of times str occurs in the string or in a substring provided that starting index is beg and ending index is end.	`>>> s1="python programming"`
			`>>> s1.count('o')`
			`2`
			`>>> s1.count('o',5,18)`
			`1`

4.	`decode()`	Decodes the string using the codec registered for encoding.	`>>> s1`
			`'python Programming'`
			`>>> s1=s1.encode()`
			`>>> s1`
			`b'python Programming'`
			`>>> s1=s1.decode()`
			`>>> s1`
			`'python Programming'`

5.	`encode()`	Returns encoded string version of string; on error, default is to raise a ValueError unless errors is given with 'ignore' or 'replace'.	`s1="python Programming"`
			`>>> s1.encode()`
			`b'python Programming'`

6.	`endswith(suf ix,`	Determines if string or a	`>>> s1="python programming is`

	beg, end)	substring of string (if starting index beg and ending index end are given) ends with suffix; returns true if so and false otherwise.	easy to learn."
			>>> s1.endswith('learn')
			False
			>>> s1.endswith('learn.')
			True
			>>> s1.endswith('is',7,26)
			False
			>>> s1.endswith('easy',7,26)
			True

7.	expandstab(tabsize)	It returns a copy of the string in which tab characters ie. '\t' are expanded using spaces, optionally using the given tabsize (default 8).	>>> s1="python\tprogramming"
			>>> s1
			'python\tprogramming'
			>>> s1.expandtabs()
			'python programming'
			>>> s1.expandtabs(16)
			'python programming'

8.	enumerate()	The enumerate() method adds counter to an iterable and returns it (the enumerate object).	>>>
			s1=['Orange','Mango','Banana','Pineapple']
			>>> s2=enumerate(s1)
			>>> print(list(s2))
			[(0, 'Orange'), (1, 'Mango'), (2, 'Banana'), (3, 'Pineapple')]

9.	find(str, beg, end)	Determine if str occurs in string or in a substring of string if starting index beg and ending index end are given returns index if found and -1 otherwise.	>>> s1="python programming"
			>>> s1.find("prog")
			7
			>>> s1.find("prog",5)
			7
			>>> s1.find("prog",10)
			-1

10.	index(str, beg, end)	Same as find(), but raises an exception if str not found.	>>> s1="python programming"
			>>> s1.index("prog")
			7
			>>> s1.index("prog",10)
			Traceback (most recent call last):
			File "<pyshell#64>", line 1, in <module>
			s1.index("prog",10)
			ValueError: substring not found

11.	isalnum()	Returns true if string has at least 1 character and all characters are alphanumeric and false otherwise.	>>> s1="python3"
			>>> s1.isalnum()
			True
			>>> s1="1234"
			>>> s1.isalnum()
			True

			>>> s1="python programming"
			>>> s1.isalnum()
			False

12.	isalpha()	Returns true if string has at least 1 character and all characters are alphabetic and false otherwise.	>>> s1="python"
			#no space and digit in this string
			>>> s1.isalpha()
			True
			>>> s1="python3"
			>>> s1.isalpha()
			False
			>>> s1="python programming"
			>>> s1.isalpha()
			False

13.	isdigit()	Returns true if string contains only digits and false otherwise.	>>> s1="python programming"
			>>> s1.isdigit()
			False
			>>> s1="python3"
			>>> s1.isdigit()
			False
			>>> s1="12345"
			#only digit in string
			>>> s1.isdigit()
			True

14.	islower()	Returns true if string has at least 1 cased character and all cased characters are in lowercase and false otherwise.	>>> s1="python"
			>>> s1.islower()
			True
			>>> s1="Python"
			>>> s1.islower()
			False

15.	isnumeric()	Returns true if a unicode string contains only numeric characters and false otherwise.	>>> s1="12345"
			>>> s1.isnumeric()
			True
			>>> s1="\u00B2345"
			>>> s1.isnumeric()
			True
			>>> s1="python3"
			>>> s1.isnumeric()
			False

16.	isspace()	Returns true if string contains only whitespace characters and false otherwise.	>>> s1=' '
			>>> s1.isspace()
			True
			>>> s1="python programming"
			>>> s1.isspace()
			False
			>>> s1=' \t'

			>>> s1.isspace() True

17.	istitle()	Returns true if string is properly "titlecased" and false otherwise.	>>> s1="Python Programming"
			>>> s1.istitle()
			True
			>>> s1="Python programming"
			>>> s1.istitle()
			False
			>>> s1="PYTHON"
			>>> s1.istitle()
			False
			>>> s1="123 Is A Number"
			>>> s1.istitle()
			True

18.	isupper()	Returns true if string has at least one cased character and all cased characters are in uppercase and false otherwise.	>>> s1="PYTHON PROGRAMMING"
			>>> s1.isupper()
			True
			>>> s1="Python Programming"
			>>> s1.isupper()
			False

Practice Questions

1. What is data structure? Which data structure used by Python?
2. How to define and access the elements of list?
3. What is list? How to create list?
4. What are the different operations that can be performed on a list? Explain with examples
5. Explain any two methods under lists in Python with examples.
6. Write a Python program to describe different ways of deleting an element from the given List.
7. What is tuple in Python? How to create and access it?
8. What are mutable and immutable types?
9. Is tuple mutable? Demonstrate any two methods of tuple.
10. Write in brief about Tuple in Python. Write operations with suitable examples
11. Write in brief about Set in Python. Write operations with suitable examples.
12. Explain the properties of dictionary keys.
13. Explain directory methods in Python.
14. How to create directory in Python? Give example
15. Write in brief about Dictionary in python. Write operations with suitable examples.
16. What is the significant difference between list and dictionary?
17. Compare List and Tuple.
18. Explain sort() with an example
19. How append() and extend() are different with reference to list in Python?
20. Write a program to input any two tuples and interchange the tuple variable.
21. Write a Python program to multiply two matrices
22. Write a Python code to get the following dictionary as output:
    {1:1, 3:9,5:25,7:49,9,81}
23. Write the output for the following:

(i)
```
>>>a=[1,2,3]
>>>b=[4,5,6]
>>> c=a+b
```
(ii)
```
>>>[1,2,3]*3
```
(iii)
```
>>>t=['a','b','c','d','e','f']
>>>t[1:3]=['x','y']
>>>print t
```

24. Give the output of Python code:
```
Str='Maharashtra State Board of Technical Education'
print(x[15::1])
print(x[-10:-1:2])
```
25. Give the output of following Python code:
```
t=(1,2,3,(4,),[5,6])
print(t[3])
t[4][0]=7
print(t)
```
26. Write the output for the following if the variable fruit='banana':
```
>>>fruit[:3]          o/p='ban'
>>>fruit[3:]          o/p='ana'
>>>fruit[3:3]         o/p=' '
>>>fruit[:]           o/p='banana'
```
27. What is string? How to create it? Enlist various operations on strings.

■■■