4.3 DICTIONARIES

Dictionaries is an unordered collection of items. Dictionaries are a kind of hash table. The value of a dictionary can be accessed by a key. Dictionaries are enclosed by curly braces '{ }' and values can be accessed using square braces '[ ]'

Syntax:

```
dict_name= {key: value}
```

A Key can be any Immutable type like String, Number, Tuple. A value can be any datatype.

The values can be repeated and the keys should not be repeated.

Ex:

>>>dict1={}

>>>dict2={1:10,2:20,3:30}

>>>dict3={'A':'apple','B':'200'}

>>>dict4={(1,2,3):'A',(4,5):'B'}

>>>dict5={[1,2,3]:'A',[4,5]:'B'} *#Error, Only Immutable types can be assigned in Keys*

4.3.1 ACCESS, UPDATE, ADD, DELETE ELEMENTS IN DICTIONARY:

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value.

Ex:

>>>d={'name':'xyz','age':23}

>>>d['name'] → 'xyz' *# since 'name' is a String datatype, it should be represented within quotes*

>>>d[name] → *shows error* *By get()method*

>>>d.get('name') → 'xyz'

Update Values in Dictionary: You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry.

Ex:

>>> d={'name':'xyz','age':23}

>>>print(d) → {' name':'xyz','age':23}

>>>d['age'=24] *#modifying existing element*

>>>print(d) → {' name':'xyz','age':23}

By update method

>>>d1={'place':'abc'}

>>>d.update(d1)

print(d) → {'place':'abc',' name':'xyz','age':23}

Adding Values in Dictionary

>>>d['gender']='m' *#Adding new entry*

>>> print(d) → {'gender':'m', 'place':'abc',' name':'xyz','age':23}

Deleting or Removing Values in Dictionary

You can either remove individual dictionary elements or clear the entire contents of a dictionary.

>>>del d['name'] →{'gender':'m', 'place':'abc','age':23}

>>>d.clear() *# remove all entries in dictionary*

>>>del d *#delete entire dictionary*

ILLUSTRATIVE EXAMPLES

Note- *Always get an Input from the USER.*

1.Write a Python Program to find Largest, Smallest, Second Largest, Second Smallest in a List without using min() & max() function.

```
>>>def find_len(list1):
        length =
        len(list1)
        list1.sort()
        print ("Largest element is:", list1[length-
        1]) print ("Smallest element is:", list1[0])
        print ("Second Largest element is:",
        list1[length-2]) print ("Second Smallest
        element is:", list1[1])
 >>>list1=[12, 45, 2, 41, 31, 10, 8, 6, 4]
>>>Largest = find_len(list1)
```

Output:

```
Largest element is:
45 Smallest
element is: 2
Second Largest element is:
41 Second Smallest
element is: 4
```

2. Find the output of the following program

Program 2.1:

```
>>>subject= ['Physics', 'Chemistry', 'Computer']
>>>mark=[98,87,94]
>>>empty=[]
>>> print(subject, mark, empty)
```

Output:

```
['Physics', 'Chemistry', 'Computer'], [98, 87, 94], []
```

Program 2.2:

```
>>>mark=[98,87,94]
>>> mark[2]=100
>>> print(mark)
```

Output:

```
 [98, 87, 100]
```

Program 2.3:

```
>>> subject= ['Physics', 'Chemistry', 'Computer']
>>> 'Chemistry' in subject
```

Output:

```
True
```

Program 2.4:

```
>>> subject= ['Physics', 'Chemistry', 'Computer']
>>> for s in subject:
        print(s)
```

Output:

```
Physics
Chemistr
y
Compute
r
```

Program 2.5:

```
>>> for i in
        range(len(mark)):
        mark[i] = mark[i] * 2
>>> print(mark)
```

Output:

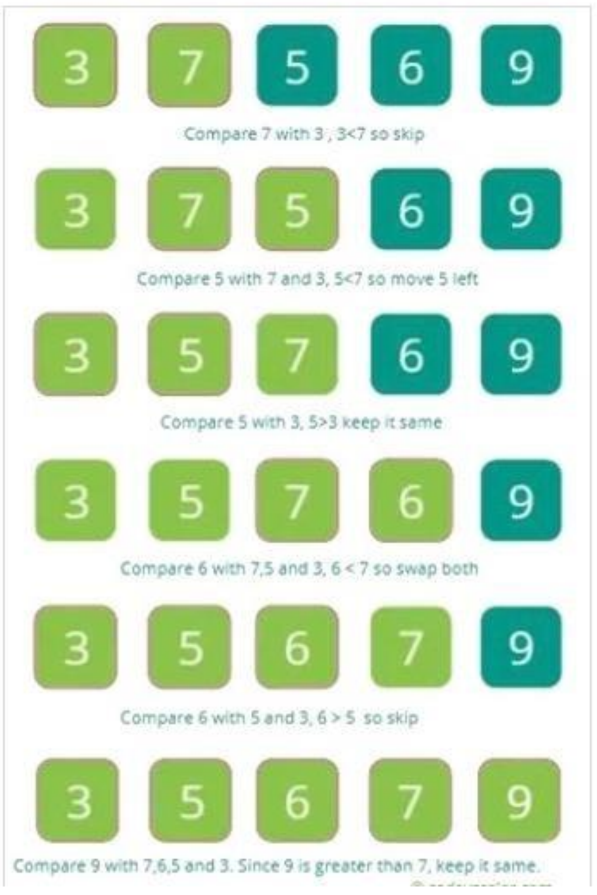
```
[196, 174, 200]
```

INSERTION SORT

This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always sorted. For example, the lower part of an array is maintained to be sorted. An element which is to be 'inserted in this sorted sub-list, has to find its appropriate place and then it has to be inserted there. Hence the name, **insertion sort**.

The array is searched sequentially and unsorted items are moved and inserted into the sorted sub-list (in the same array). This algorithm is not suitable for large data sets as its average and worst case complexity are of O(n^2), where **n** is the number of items.

Now let's see the visual representation of the algorithm –


3 7 5 6 9
Compare 7 with 3 , 3<7 so skip
3 7 5 6 9
Compare 5 with 7 and 3, 5<7 so move 5 left
3 5 7 6 9
Compare 5 with 3, 5>3 keep it same
3 5 7 6 9
Compare 6 with 7,5 and 3, 6 < 7 so swap both
3 5 6 7 9
Compare 6 with 5 and 3, 6 > 5 so skip
3 5 6 7 9
Compare 9 with 7,6,5 and 3. Since 9 is greater than 7, keep it same.

Program for Insertion sort

```
def insertionsort(a):
        for index in
              range(1,len(a):
              currentvalue=a[i]
              position=i
              while position>0 and a[position-
                  1]>currentvalue: a[position]=a[position-1]
                  position=position-1
                  a[position]=currentvalu
                  e
list=[50,60,40,30,20,70]
print( “Original list is:”,list)
insertionsort(list)
print(”List after insert:”,a)
```

Output:

```
Original list
is=[50,60,40,30,20,70] List
afterinsert:[20,30.40,50,60,70]
```

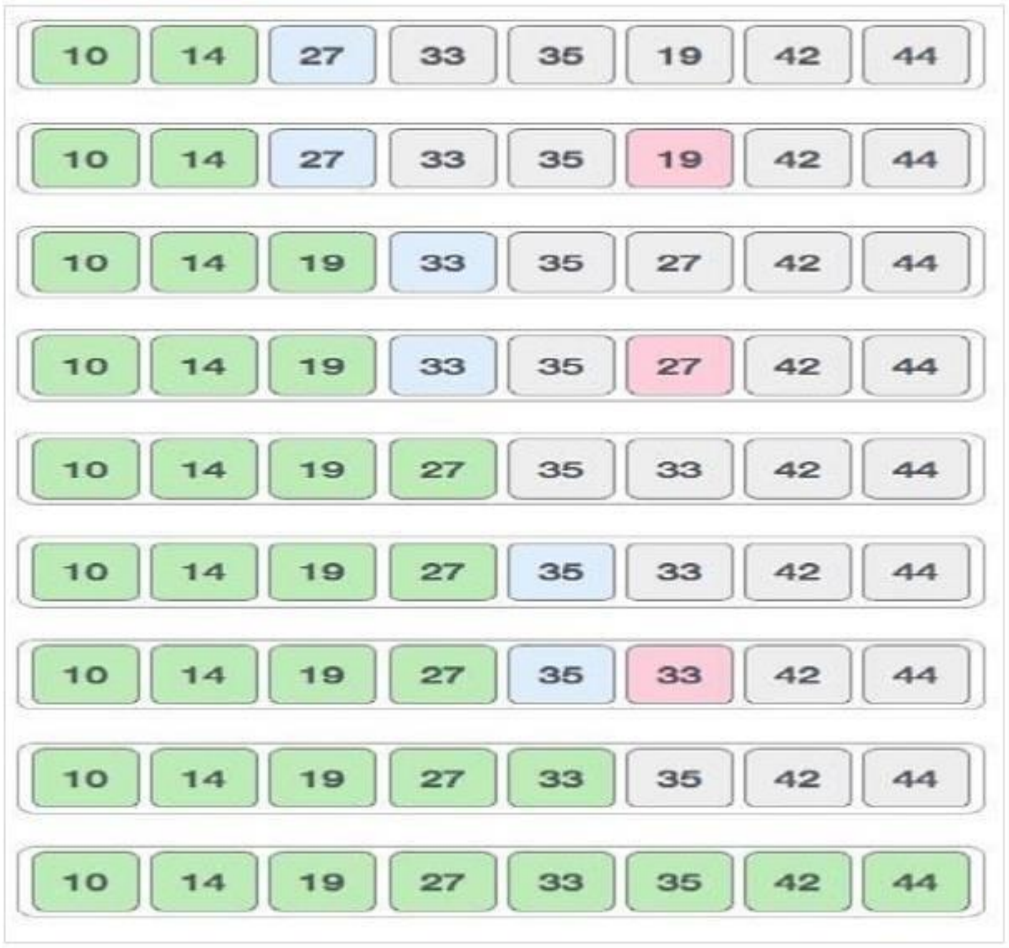
2. SELECTION SORT

In the **selection sort** algorithm, an array is sorted by recursively finding the minimum element from the unsorted part and inserting it at the beginning. Two subarrays are formed during the execution of Selection sort on a given array.

- The subarray, which is already sorted
- The subarray, which is unsorted.

During every iteration of selection sort, the minimum element from the unsorted subarray is popped and inserted into the sorted subarray.

Let's see the visual representation of the algorithm –



Now let's see the implementation of the algorithm –

Program for Selection sort:

```
def selectionSort(alist):
    for i in range(len(alist)-1,0,-
      1): pos=0
      for location in range(1,i+1):
          if alist[location]>alist[pos]:
              pos= location
              temp = alist[i]
              alist[i] =
              alist[pos]
              alist[pos] =
              temp
alist = [54,26,93,17,77,31,44,55,20]
selectionSort(alis
t) print(a list)
```

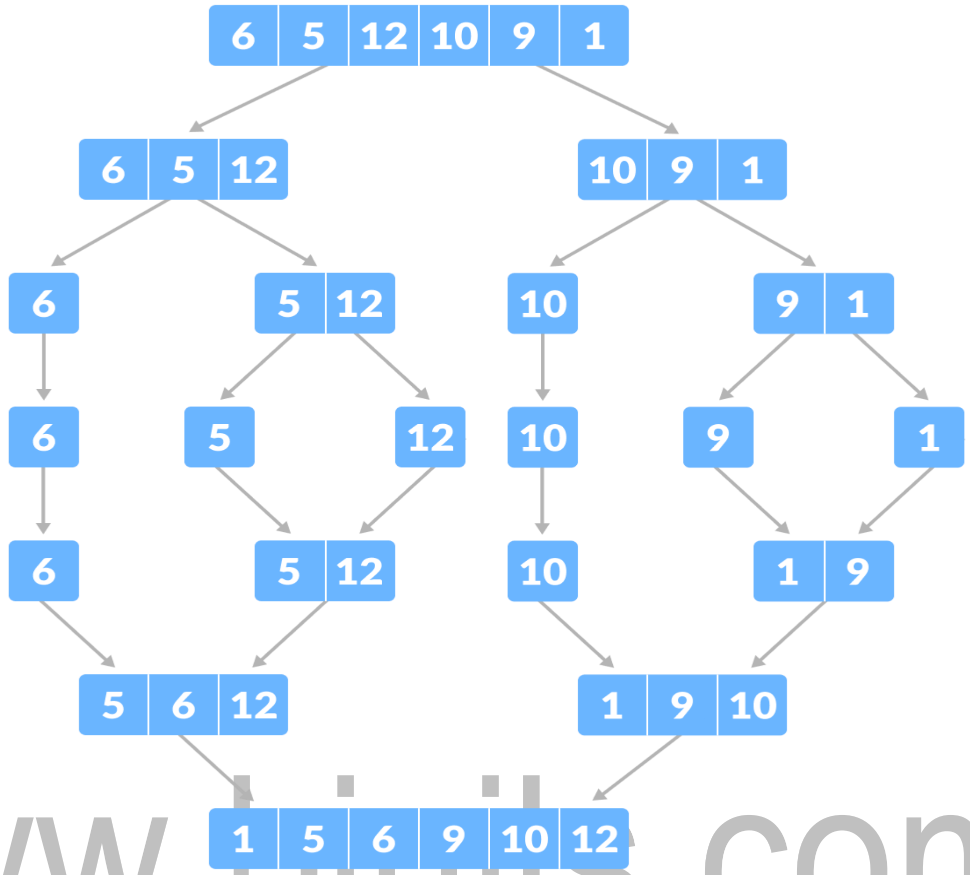
Output:

```
[17, 20, 26, 31, 44, 54, 55, 77, 93]
```

3. MERGE SORT

Merge sort is a sorting technique based on divide and conquer technique. With worst- case time complexity being O(n log n), it is one of the most respected algorithms.

Merge sort first divides the array into equal halves and then combines them in a sorted manner.



Program for Merge sort

```
def mergeSort(alist):
  print("Splitting
  ",alist) if
  len(alist)>1:
    mid = len(alist)//2
    lefthalf = alist[:mid]
    righthalf =
    alist[mid:]
    mergeSort(lefthalf)
    mergeSort(righthalf
    ) i=0
    j=0
    k=
    0
```

```
        while i < len(lefthalf) and j <
          len(righthalf): if lefthalf[i] < righthalf[j]:
            alist[k]=lefthalf[i
            ] i=i+1
          else:
            alist[k]=righthalf[j
            ] j=j+1
          k=k+1
        while i <
          len(lefthalf):
          alist[k]=lefthalf[i]
          i=i+1
          k=k+1
        while j <
          len(righthalf):
          alist[k]=righthalf[j]
          j=j+1
          k=k+1
      print("Merging
      ",alist)
    alist = [50, 60, 40, 20, 70, 100]
    mergeSort(alis
    t) print(alist)
```

Output:

```
Original list is: [50, 60, 40, 20, 70, 100]
Sorted list is: [20, 40, 50, 60, 70, 100]
```

HISTOGRAM

To create a **histogram,** the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and the count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. **Program:**

```
def  histogram(items):
        for n in items:
                output=’’
                times=n
                while (times>0):
                        output+=’*’
                times=times-1
                print(output)
                histogram([2,3,4,3,
                2])
```

Output :

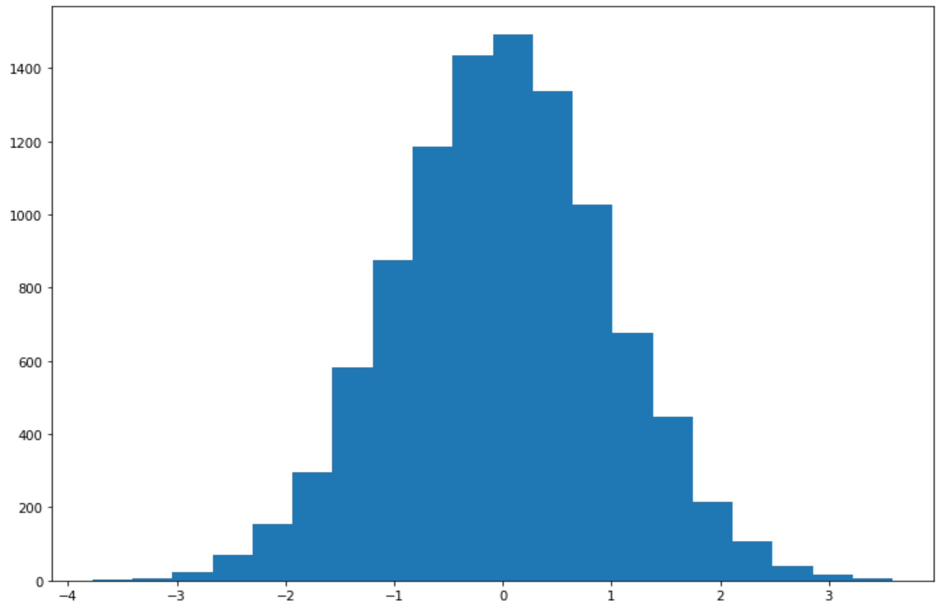
```
**
***
****
***
**
```

Example :2

```
import matplotlib.pyplot as
plt import numpy as np
from matplotlib import
colors from matplotlib.ticker
import PercentFormatter
```

```
# Creating dataset
np.random.seed(236857
52)
N_points = 10000
n_bins = 20
# Creating distribution
x = np.random.randn(N_points)
y = .8 ** x + np.random.randn(10000) +
25 # Creating histogram
fig, axs = plt.subplots(1,
1, figsize =(10, 7),
tight_layout = True)
axs.hist(x, bins =
n_bins) # Show plot
plt.show()
```

Output :



4.1.6 LIST ALIASING:

Since variables refer to objects, if we assign one variable to another, both variables refer to same objects.(One or more variable can refer the same object)

```
>>>a=[1,2,3]
>>>b=a
>>>a is b              # Displays True
```

4.1.7 LIST CLONING:

If we want to modify a list and also keep copy of the original we can use cloning to copy the list and make that as a reference.

Ex:

```
a=[1,2,3]
b=a[:]
print(b) → [1,2,3]
```

4.1.8 LIST PARAMETER:

Passing a list as an argument actually passes a reference to the list, not the copy of the list. We can also pass a list as an argument to the function.

Ex:

```
>>> def mul(a_list): #a_list is a list passing as a parameter
        for index,value in
                enumerate(a_list):
                a_list[index]=2*value
        print(a_list)
>>> a_list=[1,2,3,4,5]
>>> mul(a_list)
```

Output:

```
[2, 4, 6, 8, 10]
```

4.1.9 DELETING LIST ELEMENTS

To remove a list element, del operator can be used if an element to be deleted is known. In the following code, the element 'Chennai' is deleted by mentioning its index in the del operator.

Ex :

```
stulist = ['Rama', 'Chennai',
2018, 'CSE', 92.7] print 'Initial
list is : ', stulist

del stulist[1]

print 'Now the list is : ', stulist
```

Output:

```
Initial list is : ['Rama', 'Chennai',
2018, 'CSE', 92.7] Now the list is :
['Rama',
2018, 'CSE', 92.7]
```

pop() and remove() methods can also be used to delete list elements

4.1.4 LIST LOOP:

A loop is to access all the elements in a list.

```
>>> a=['apple','mango','lime','orange']
>>> print(a) → ['apple','mango','lime','orange'] # displays all at a time
>>> print(a[0]) → 'apple'
```

i) Method 1:

```
for var in a:
      print(a) #displays all at a time
```

Output:

The loop goes on 4 times and print all values 'apple','mango','lime','orange'

```
'apple','mango','lime','orange'
'apple','mango','lime','orange'
'apple','mango','lime','orange'
```

ii) Method 2:

```
for var in a:
     print(var) #displays a item at a time
```

Output:

The loop goes on 4 times and print items one by one.

```
'apple'
'mango
' 'lime'
'orange'
```

iii) Method 3

```
    >>>i=0
    >>> for var in a:
```

```
                print(“I like”,a[i])
          i=i+
          1
```

Output:

The value of i makes the loop to goes on 4 times and print items one by one.

```
‘apple’
’mango
’ ‘lime’
’orange’
```

1. Write a python program to print the items in the list using while loop.

```
>>>a=[‘apple’,’mango’,’lime’,’orange’]
>>>i=0
>>>while len(a)>i:
          print(“I like”,a[i])
          i=i+1
```

Output:

The value of i makes the loop to goes on 4 times and print items one by one.

```
‘apple’
’mango
’ ‘lime’
’orange’
```

4.1.5 LIST ARE MUTABLE:

Unlike String, List is mutable (changeable) which means we can change the elements at any point.

Ex:

```
>>>a=[‘apple’,’mango’,’lime’,’orange’]
>>>a[0]=’grape’
>>>print(a) → ’grape’, ’mango’,’lime’,’orange’
```

4.1.1 LIST OPERATIONS:

1. + Operator which concatenates two lists.

```
>>>list2=[1,2,3,4,5,6,7,8]
>>>list3=['Hello',3.5,'abc',4]
>>>print(list2+list3)
```

Output

```
1,2,3,4,5,6,7,8, 'Hello',3.5,'abc',4
```

2. * Operator multiples the list to the specific numbers

```
>>>list2=[1,2,3,4,5,6,7,8]
>>>list2*2
```

Output

```
1,2,3,4,5,6,7,8,1,2,3,4,5,6,7,8
```

4.1.2 LIST SLICE

A subsequence of a sequence is called a slice and the operation that extracts a subsequence is called slicing. For slicing we use square brackets [ ]. Two integer values splitted by **(:).**

Syntax:

List_Name[Starting_Value : Ending_Value]

Ex:

```
>>>a=['a','b','c','d','e']
```

List	a=	'a'	'b'	'c'	'd'	'e'
Index from Left		0	1	2	3	4
Index from Right		-5	-4	-3	-2	-1

```
>>> print(a[:]) → ['a', 'b', 'c', 'd', 'e']        #Prints ALL
>>> print(a[1:]) → ['b', 'c', 'd', 'e']            #Print from 1st Position to Last Position
>>> print(a[1:3]) → ['b', 'c']                     #Print from 1st Position to Last – 1 Position
```

>>> print(a[:-1]) → ['a', 'b', 'c', 'd'] #Print from Backwards except -1th Position

>>> print(a[1:-1]) → ['b', 'c', 'd'] #Print from 1st Position till -1th Position

4.1.3 LIST METHODS (or) TYPES OF FUNCTIONS IN LIST

*Consider the values of **list a and list b** be*

>>>a=[‘apple’,’mango’,’lime’]

>>>b=[‘grape’]

S.No	Name	Synta x	Description	Example
1.	append()	listname.append()	The method append() will add the item to the end of a list	a.append(‘orange’)
2.	insert()	listname.insert(index, it em)	This method inserts an item at a particular place and two arguments (index,item)	a.insert(1,’banana ’)
3.	extend()	listname.extend(item 1 ,item2)	This method is used to combine two list with the items in the argument.	a.extend(‘grape ’) (or) a.extend(b)
4.	remove()	listname.remove(item)	This method will remove an item in the list.	a.remove(’apple’)
5.	pop()	listname.pop(index)	This method returns the item by the index position and removes it.	a.pop(1) >>>mango
6.	index()	listname.index(item)	This method will return index value of list and takes index value as argument.	a.index(‘lime’) >>>2
7.	copy()	dest_list=listname.c op y()	This method is used to copy a list to another list.	c=a.copy()

8.	reverse()	listname.reverse()	This method is used to reverse the items in a list.	a.reverse()
9.	count()	listname.count(item)	This method is used to count the duplicate items in the list which takes the item as arguments.	a.count('lime') >>>1
10.	sort()	listname.sort()	This method is used to arrange the list from ascending to descending alphabetically.	a.sort() >>>a=['apple', 'lime', 'mango']
11.	clear()	listname.clear()	This method is used to clear all the values in the list.	a.clear() -->[]

4.3.2 METHODS ON DICTIONARIES:

Consider the value of **Dictionary d and d1** as follows: d={'a':1,'b':2,'c':3,'d':4}

d1={'e':5,'f':6}

S.No	Name	Syntax	Description	Example
1.	len()	len(dictonary)	Gives the total length of the dictionary.	len(d) → 4
2.	keys()	dictionary.keys()	Return the dictionary's keys.	>>> d.keys() ['a', 'c', 'b', 'd']
3.	values()	dictionary.values()	Return the dictionary's Values	>>> d.values() [1, 3, 2, 4]
4.	items()	dictionary.items()	Return the dictionary's Keys and Values	>>> d.items() [('a', 1), ('c', 3), ('b', 2), ('d', 4)]
5.	key in dict	key in dict	Returns True if the Key is in Dictionary, else returns false.	>>> 'a' in d True
6.	key not in dict	key not in dict	Returns True if the Key is not in Dictionary, else returns false.	>>> 'e' not in d True
7.	has_key (key)	dictionary.has_key(key)	Checks whether the dictionary has the specified key.	>>> d.has_key('a') True
8.	get()	dictionary.get(key)	Get the value and Return the value of key.	>>> d.get('b') 2

9.	update()	Dest_dictionary.update (Source_dict)	Update the dictionary with the key from existing keys.	>>>d.update(d1) >>> print(d) {'a': 1, 'c': 3, 'b': 2, 'e': 5, 'd': 4, 'f': 6}
10.	cmp()	cmp(dictionary1, dictionary2)	Compares elements of both dictionary. Returns 0 if the elements are same, Else returns 1.	>>> cmp(d,d1) 1
11	copy()	new_dict = original_dict.copy( )	Return a copy of the dictionary.	>>> d2=d1.copy() >>> print(d2) {'e': 5, 'f': 6}
12.	pop()	dictionary.pop('key')	Remove the item with key and return its value	>>> d.pop('f') 6
13.	popitems( )	dictionary.popitem()	Remove and return an item with its key and value.	>>> d.popitem() ('a', 1)
14.	clear()	dictionary.clear()	Remove all items form the dictionary.	d.clear()

4.4 LIST COMPREHENSION:

List comprehension is an elegant way to define and create list in Python. These lists have often the qualities of sets. It consists of brackets containing an expression followed by a for clause, then zero or more for or if clauses.

Ex-1:

```
>>>x = [i for i in range(10)]
>>>print x
```

Output:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Ex-2:

```
>>>squares = []
>>>for x in range(10):
          squares.append(x**
     2) print (squares)
>>>squares = [x**2 for x in range(10)] # List comprehensions to get the same result:
>>>print (squares)
```

Output:

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

4.2 TUPLE

Tuples are sequence of values much like the list. The values stored in the tuple can be of any type and they are indexed by integers.

The main difference between list and tuple is **Tuple is immutable**. Tuple is represented using '( )'

Syntax:

```
tuple_name=(items)
```

Ex:

```
>>> tuple1=('1','2','3','5')
>>>tuple2=('a','b','c')
>>>tuple3='3','apple','100'
```

TUPLES ARE IMMUTABLE:

The values of tuple cannot be changed.

```
>>> tuple1=('1','2','3','5')
tuple1[1]='4' #It Shows Error
```

Tuples can be immutable but if you want to add an item we can add it by

```
t1=('a','b')
t1=('A',)+t1[1:] → t1=('A','b)
```

The disadvantage in this method is we can only add the items from the beginning.

4.2.1 TUPLE ASSIGNMENT

Tuple Assignment means assigning a tuple value into another tuple.

Ex:

```
 t=('Hello','hi')
>>>m,n=t
>>>print(m) → Hello
>>>print(n) → hi
>>>print(t) → Hello,hi
```

In order to interchange the values of the two tuples The following method is used.

```
>>>a=('1','4')
>>>b=('10','15')
>>>a,b=b,a
>>>print(a,b)
(('10','15'), ('1','4'))
```

COMPARING TUPLES

The comparison operator works with tuple and other sequence. It will check the elements of one tuple to another tuple. If they are equal it return true. If they are not equal it returns false.

```
>>>t1=('1','2','3','4','5')
>>>t2=('a','b','c')
>>>t1<t2 #It returns false
```

4.2.2 TUPLE AS RETURN VALUE

In a function a tuple can return multiple values where a normal function can return single value at a time.

Example-1:

Using a built-in function **divmod** which return quotient and remainder at the same time.

```
>>>t=divmod(7,3)
>>>print(t) → (2,1)
>>>quot,rem=divmod(7,3)
>>>print(quot) → 2
>>>print(rem) →1
```

Example-2:

```
def swap(a,b,c):
        return(c,b,a)
a=100
```

```
b=20
0
c=30
0
>>>print("Before Swapping",a,b,c)
>>>print("After Swapping",swap(a,b,c))
```

Output:

```
Before Swapping 100,200,300
After Swapping 300,200,100
```