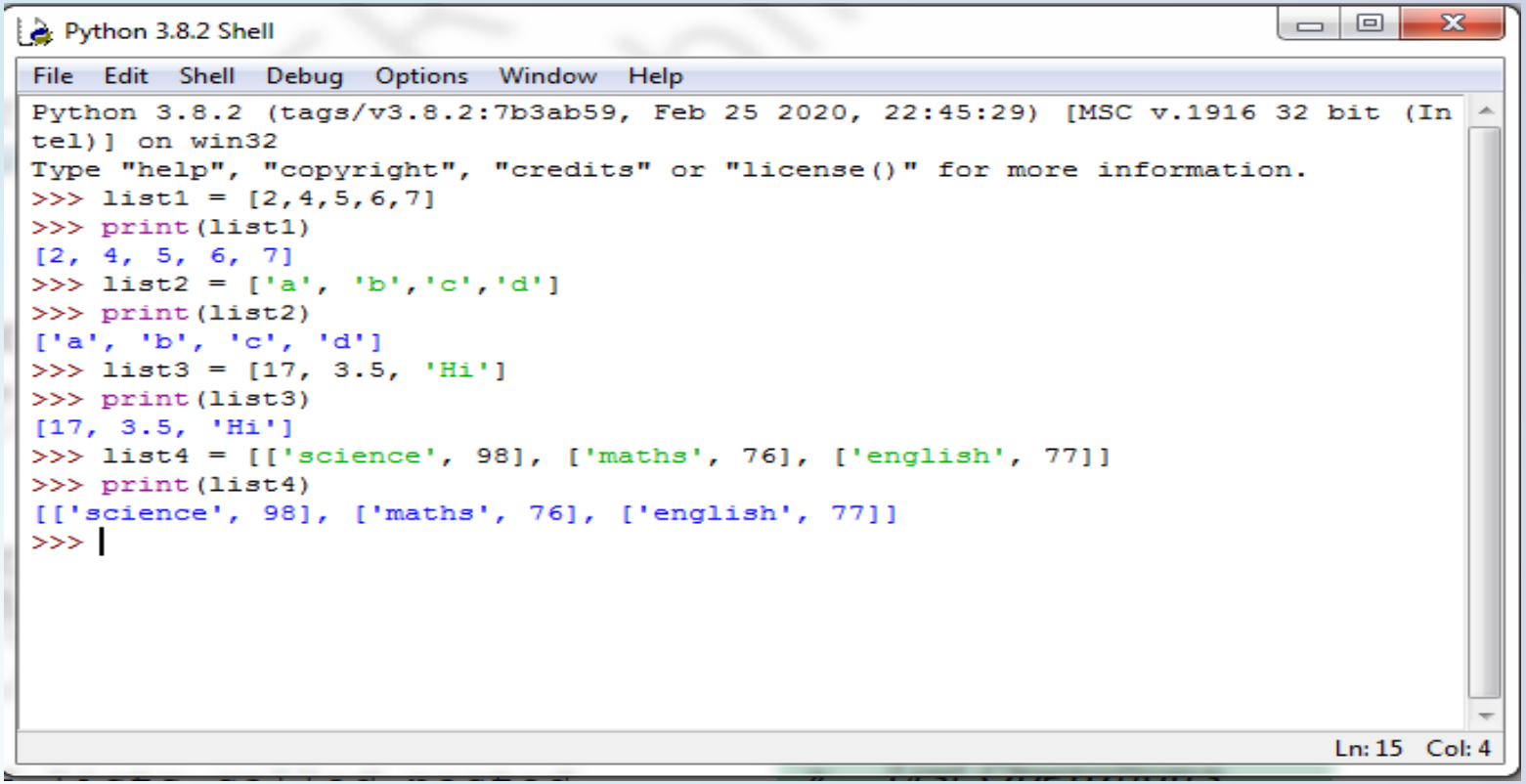
List in Python

LEARNING OUTCOMES

- Definition, Creation of a list, Traversal of a list.
- Operations on a list - concatenation, repetition, membership
- functions/methods–len(), list(),append(), extend(), insert(), count(), index(), remove(), pop(), reverse(),sort(), min(), max(), sum()
- Lists Slicing
- Nested lists
- Finding the maximum, minimum, mean of numeric values stored in a list.
- Linear search on list of numbers and counting the frequency of elements in a list.

INTRODUCTION

The data type list is an ordered sequence which is mutable and made up of one or more elements. A list can have elements of different data types, such as integer, float, string, tuple or even another list. A list is very useful to group together elements of mixed data types.

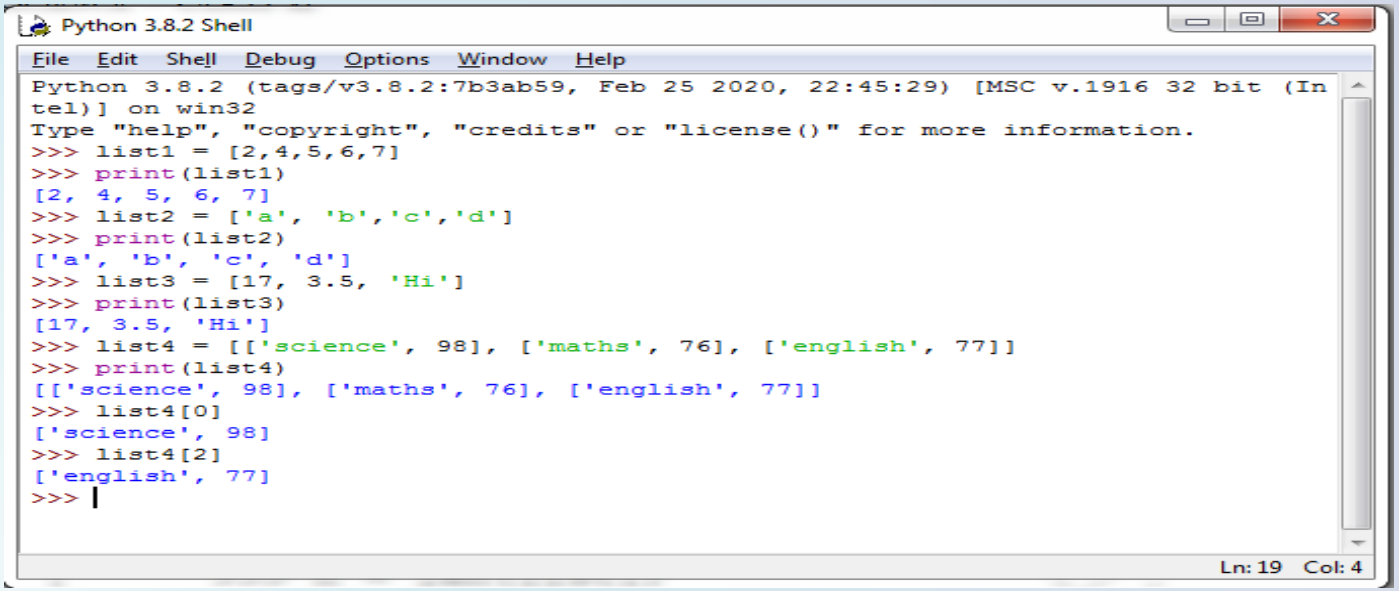


ELEMENT OF LIST

Indexing of list: Index of first item is 0 and the last item is n-1.

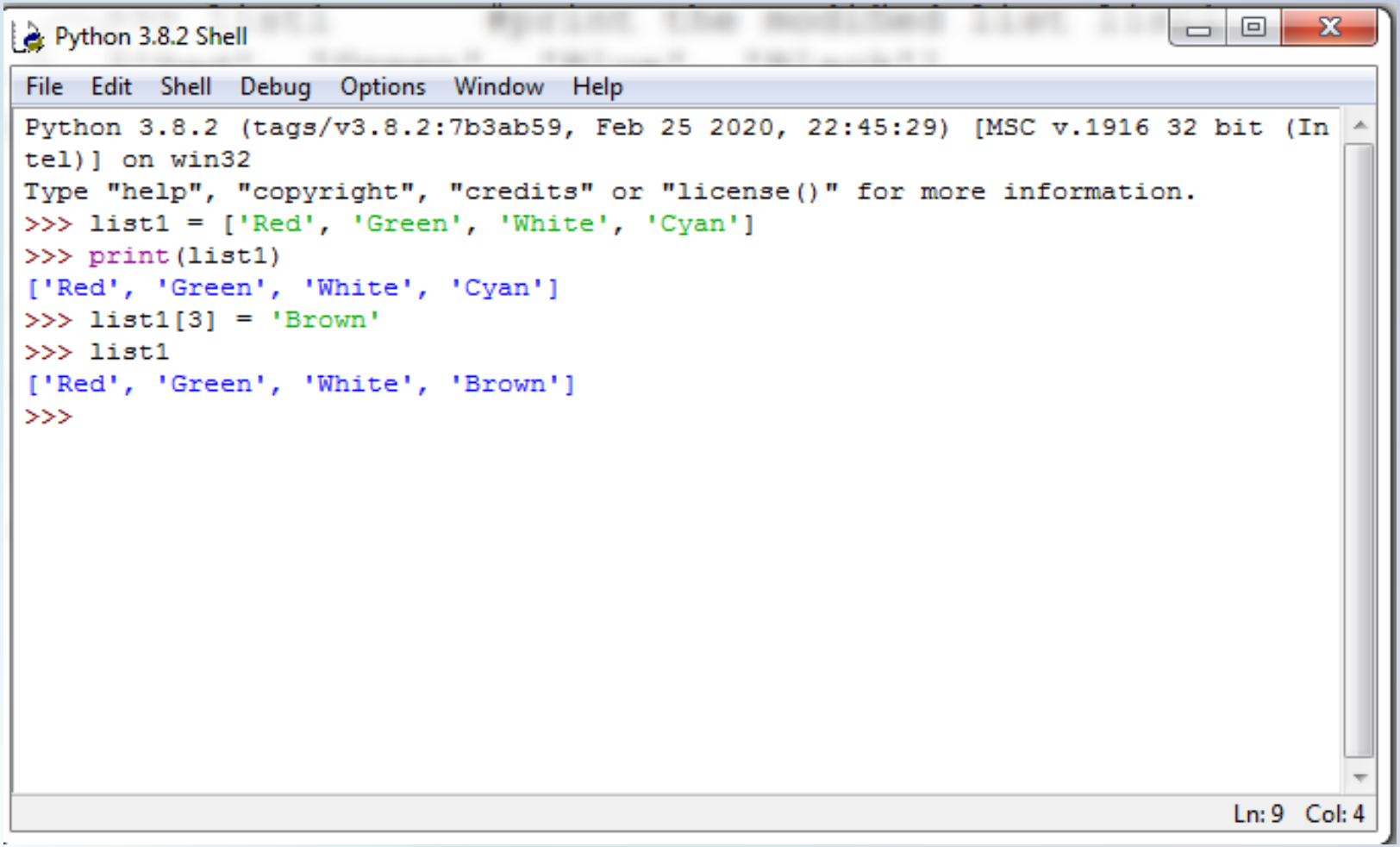
0	1	2	3	4	index
80	60	70	85	75	value
-5	-4	-3	-2	-1	Negative index

Accessing element of list: List items can be accessed using its index position.



LIST ARE MUTABLE

It means that the contents of the list can be changed after it has been created.



LIST OPERATIONS

- **Concatenation:** joining two or more list together by using symbol +.

```
Example:    >>> list1 = [1,3,5,7,9]
            >>> list2 = [2,4,6,8,10]

            >>> list1 + list2
                    [1, 3, 5, 7, 9, 2, 4, 6, 8, 10]
```

- **Repetition:** to replicate a list using repetition operator(*).

```
Example:   >>> list1 = ['Hello']

              >>> list1 * 4
        ['Hello', 'Hello', 'Hello', 'Hello']
```

- **Membership:** The membership operators in checks if the element is present in the list and returns True, else returns False.

```
Example:    >>> list1 = ['Red','Green','Blue']
            >>> 'Green' in list1
        True
            >>> 'Cyan' in list1
        False
```

- **Slicing:** Access the list elements in subparts.

```
Example:     >>>list =['I','N','D','I','A']
             >>>print(list[0:3])
              >>>print(list[3:])
              >>>print(list[:])

          ['I', 'N', 'D']
          ['I', 'A']
          ['I', 'N', 'D', 'I', 'A']
```

TRAVERSING A LIST

- **List traversing using a for loop:**

```
>>> list1 = ['Red','Green','Blue','Yellow', 'Black']
>>> for item in list1:
print(item)
```

or

```
  >>> for i in range(len(list1)):
print(list1[i])
```

- **List traversing using while loop:**

- ```
  >>> list1 = ['Red','Green','Blue','Yellow', 'Black']
  >>> i = 0
  >>> while i < len(list1):
  print(list1[i])
  i += 1
  ```
  ```

BUILT-IN LIST FUNCTIONS

- **Len():** returns the length of the list passed.

  Example: len(list1)

- **List():** creates an empty list if no arguments are passed and creates a list if a sequence is passed as an argument.

  Example: list1= list() or list1=list(str)

- **Append():** appends a single element passed as an argument at the end of the list.

  Example: list1.append(20)

- **Extend():** appends each element of the list passed as argument at the end of the given list.

  Example: list1= [10,20,30] list2= [40,50]

  list1.extend(list2)

- **Insert():** inserts an element at a particular index in the list.

  Example: list1.insert(2, 50)

BUILT-IN FUNCTIONS (CONTD.)

- **Count():** returns the number of times a given element appears in the list.

  Example: list1.count(20)

- **Index():** returns index of the first occurrence of the element in the list. Gives ValueError, if element is not present.

  Example: List1.index(20)

- **Remove():** removes the given element from the list. If multiple occurrence is found, only the first occurrence is removed.

  Example: list1.remove(30)

- **Pop():** returns the element whose index is passed as parameter to this function and also removes it from the list.

  Example: list1.pop(3) list1.pop()

- **Reverse():** reverses the order of elements in the given list.

  Example: list1.reverse()

- **Sort():** sorts the elements of the given list in-place.

  Example: list1.sort()

BUILT-IN FUNCTIONS (CONTD.)

- **Sorted():** takes the list as parameter and creates a new list consisting of the same elements arranged in ascending order.

  Example: list2 = sorted(list1)

- **Min():** returns smallest element of the list.

  Example: min(list1)

- **Max():** returns the largest element of the list.

  Example: max(list1)

- **Sum():** returns sum of the element of the list.

  Example: sum(list1)

NESTED LISTS

When a list appears as an element of another list, it is called a nested list.

Example:
```
>>> list1 = [1,2,'a','c',[6,7,8],4,9]
#fifth element of list is also a list
>>> list1[4]
[6, 7, 8]
```

To access the element of the nested list of list1, we have to specify two indices list1[i][j].

```
>>> list1[4][1]
7
```

Note: index i gives the fifth element of list1which is a list and index j gives the second element in the nested list

COPYING LISTS

- There are three methods to copy a list into another list.
- **Method1:** We can slice our original list and store it into a new variable.

```
Example: >>> list1 = [1,2,3,4,5]
         >>> list2 = list1[:]
         >>> list2
             [1, 2, 3, 4, 5]
```

- **Method2:** We can use the built-in function list().

```
Example:  >>> list1 = [10,20,30,40]
          >>> list2 = list(list1)
          >>> list2
            [10, 20, 30, 40]
```

- **Method3:** We can use the copy () function.

```
Example: >>> import copy
         >>> list1 = [1,2,3,4,5]
         >>> list2 = copy.copy(list1)
         >>> list2
             [1, 2, 3, 4, 5]
```

FINDING MIN, MAX AND MEAN VALUE

- Program to find the max element in the list.

```
def max_num_in_list( list ):
max = list[ 0 ]
for a in list:
if a > max:
max = a
return max
print(max_num_in_list([1, 2, -8, 0]))
```

```
list1, list2 = [123, 'xyz', 'zara', 'abc'], [456, 700, 200]
print "Max value element : ", max(list1)
print "Max value element : ", max(list2)
Output
Max value element :  zara
Max value element :  700
```

- Program to find the mean of a list

```
def Average(lst):
return sum(lst) / len(lst)

lst = [15, 9, 55, 41, 35, 20, 62, 49]
average = Average(lst)
# Printing average of the list
print("Average of the list =", round(average, 2))
```

Output:
Average of the list = 35.75

LINEAR SEARCH

- A **linear search** or **sequential search** is a method for finding an element within a list. It sequentially checks each element of the list until a match is found or the whole list has been searched.
- **Program to implement Linear Search**

```
list_of_elements = [4, 2, 8, 9, 3, 7]
x = int(input("Enter number to search: "))
found = False
for i in range(len(list_of_elements)):
if(list_of_elements[i] == x):
found = True
print("%d found at %dth position"%(x,i))
break
if(found == False):
print("%d is not in list"%x)
```

COUNTING THE FREQUENCY OF ELEMENTS IN A LIST

- **Frequency of an element in list**

```
import collections
my_list = [101,101,101,101,201,201,201,201]
print("Original List : ",my_list)
ctr = collections.Counter(my_list)
print("Frequency of the elements in the List : ",ctr)
```

OUTPUT:
Original List : [101, 101,101, 101, 201, 201, 201, 201]
Frequency of the elements in the List : Counter({101: 4, 201:4})

- **NOTE :**SAME CAN BE ACHIEVED USING COUNT FUNCTION.

E.G. lst.count(x)

PROGRAMS:

- WAP to find the smallest and second smallest element in the list?
- WAP a menu driven program to insert and delete the element in the list having sub-menus like insert at the end or specific position and delete the element using value or using index.
- WAP to search the element in the list?

2-D LIST

The **2-D list** is the list having rows and columns.

For example: list1=[[1,2],[3,4],[5,6],[7,8]]

So this list is having 4 rows and 2 columns.

- **How to insert the elements in 2-D list:**

```
list1=[]
row=[]
r=int(input("enter the number of rows"))
c=int(input("enter the number of columns"))
for i in range(r):
  for j in range(c):
    element= int(input("element "+ str(i)+ ", "+ str(j)+":"))
    row.append(element)
  list1.append(row)
```

2-D LIST(CONTD…)

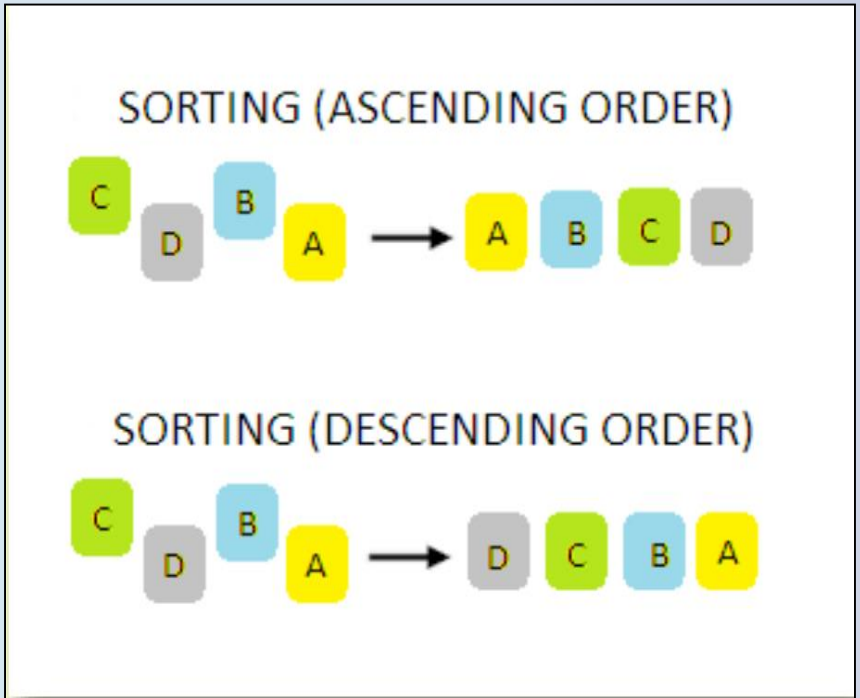
Printing the 2-D list:

```
for i in range(r):
   print("[", end="")
   for j in range(c):
      print(list3[i][j], end="")
   print("]")
```

```
========== RESTART: C:/Program Files/Pytho
Enter the no. of rows.3
Enter the no. of cols.2
Element0,0:1
Element0,1:1
Element1,0:2
Element1,1:2
Element2,0:3
Element2,1:3
[11]
[22]
[33]
>>>
```

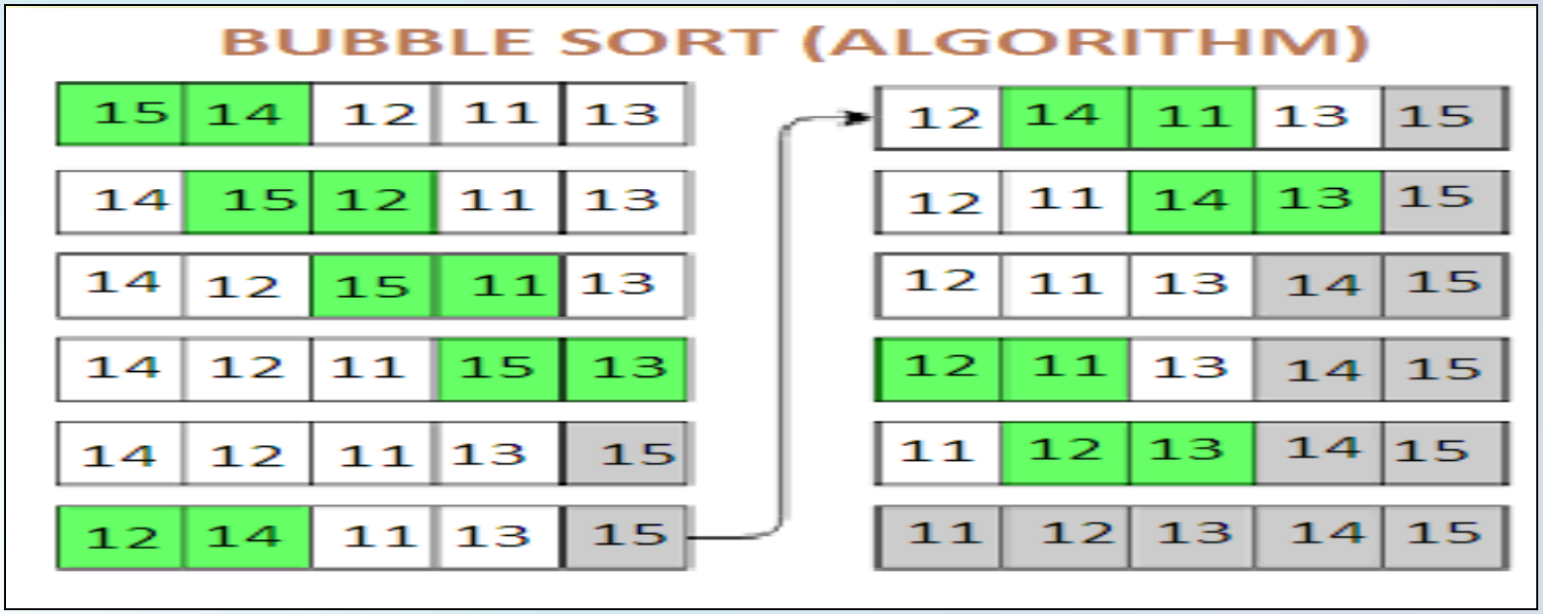
SORTING

- Sorting is process of arranging items systematically ,according to a comparison operator applied on the elements.
- There are various sorting algorithms .Two of them are-
  1. Bubble Sort
  2. Insertion Sort



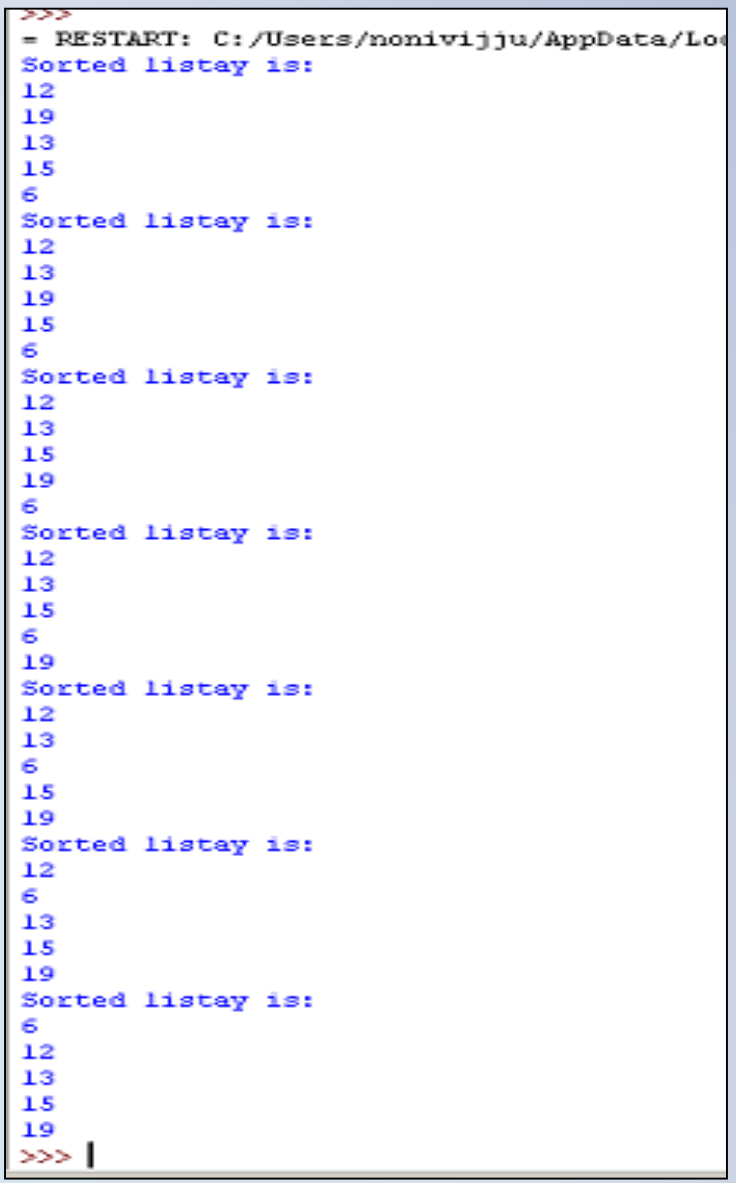
BUBBLE SORT

- It is one of the simplest sorting algorithms. The two adjacent elements of a list are checked and swapped if they are in wrong order and this process is repeated until the whole list elements are sorted. The steps of performing a bubble sort are:
  1. Compare the first and the second element of the list and swap them if they are in wrong order.
  2. Compare the second and the third element of the list and swap them if they are in wrong order.
  3. Proceed till the last element of the list in a similar fashion.
  4. Repeat all of the above steps until the list is sorted.



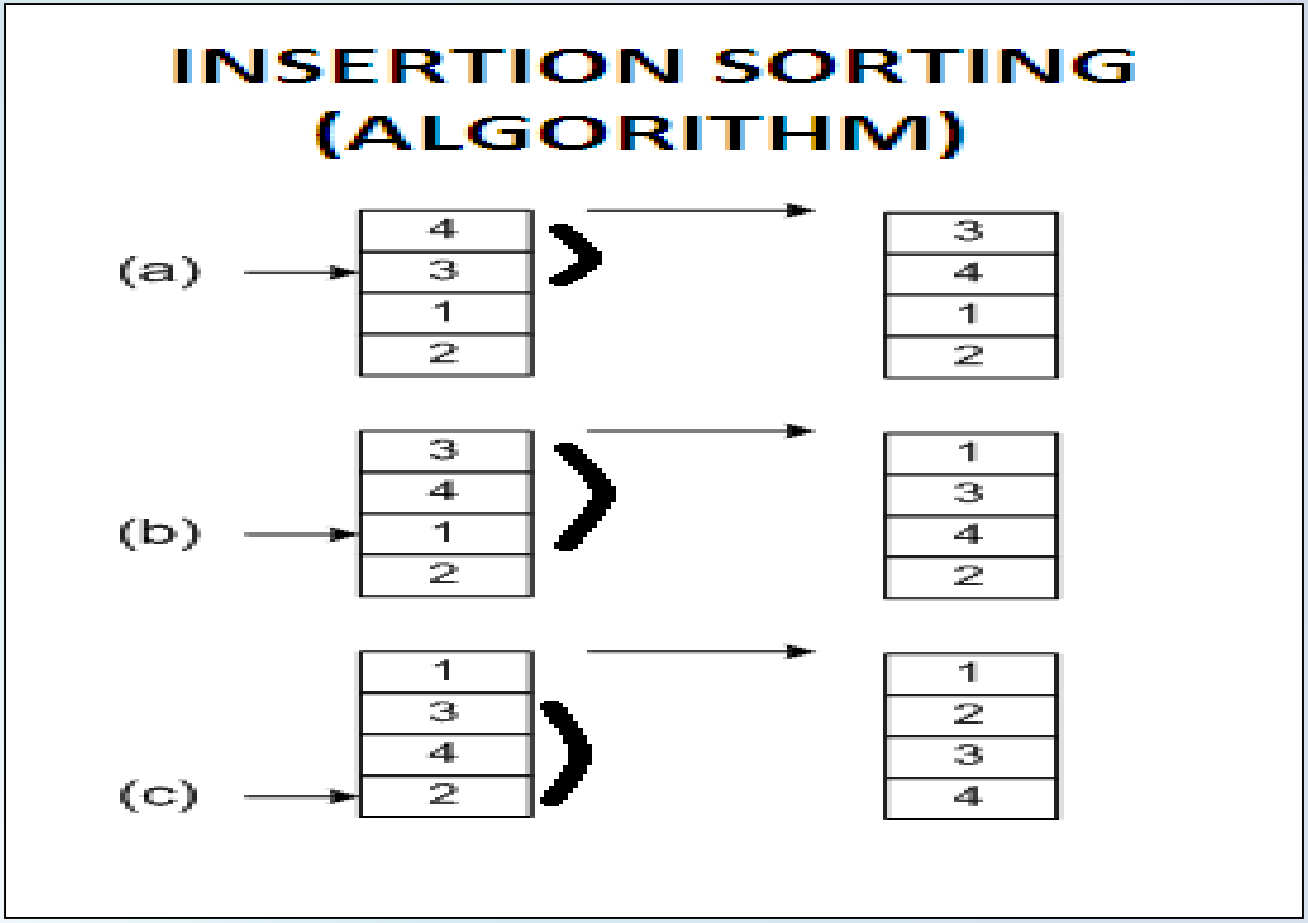
BUBBLE SORT(PROGRAM)

```
a = [6, 19, 1, 15, 11, 12, 14]
#repeating loop len(a) number of times
for j in range(len(a)):
#initially swapped is false
swapped = False
i = 0
while i<len(a)-1:
#comparing the adjacent elements
if a[i]>a[i+1]:
#swapping
a[i],a[i+1] = a[i+1],a[i]
#Changing the value of swapped
swapped = True
i = i+1
#if swapped is false then the list is sorted
#we can stop the loop
if swapped == False:
break
print (a)
```



INSERTION SORT

Insertion sort is a simple sorting algorithm .It is just similar the way we sort playing cards in our hands.



INSERTION SORT (PROGRAM)

```
list = [19, 12, 13, 15, 6]
for i in range(1, len(list)):
key = list[i]
# Move elements of list[0..i-1], that are
# greater than key, to one position next
# of their current position
j = i-1
while j >=0 and key < list[j] :
list[j+1] = list[j]
j -= 1
list[j+1] = key
print ("Sorted listay is:")
for i in range(len(list)):
print ("%d" %list[i])
```