DATA STRUCTURES LABORATORY
LAB MANUAL

Academic Year	:	**2017 - 2018**
Course Code	:	**ACS102**
Regulations	:	**IARE - R16**
Semester	:	**II Semester**
Branch	:	**CSE / IT / ECE / EEE**

Prepared by

Ms. B Padmaja
Associate Professor

Department of Computer Science and Engineering

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)
Dundigal, Hyderabad - 500 043

Program Outcomes (Common for all branches)	
PO1	**Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
PO2	**Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
PO3	**Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
PO4	**Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
PO5	**Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
PO6	**The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
PO7	**Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
PO8	**Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
PO9	**Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
PO10	**Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
PO11	**Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
PO12	**Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.
Program Specific Outcomes (CSE)	
PSO1	**Professional Skills:** The ability to research, understand and implement computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer-based systems of varying complexity.
PSO2	**Problem-Solving Skills:** The ability to apply standard practices and strategies in software project development using open-ended programming environments to deliver a quality product for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths, to be an entrepreneur, and a zest for higher studies.

Program Specific Outcomes (IT)	
PSO1	**Professional Skills:** The ability to understand, analyze and develop computer programs in the areas related to algorithms, system software, multimedia, web design, big data analytics, and networking for efficient analysis and design of computer - based systems of varying complexity.
PSO2	**Software Engineering Practices:** The ability to apply standard practices and strategies in software service management using open-ended programming environments with agility to deliver a quality service for business success.
PSO3	**Successful Career and Entrepreneurship:** The ability to employ modern computer languages, environments, and platforms in creating innovative career paths to be an entrepreneur, and a zest for higher studies.
Program Specific Outcomes (ECE)	
PSO1	**Professional Skills:** An ability to understand the basic concepts in Electronics & Communication Engineering and to apply them to various areas, like Electronics, Communications, Signal processing, VLSI, Embedded systems etc., in the design and implementation of complex systems.
PSO2	**Problem-Solving Skills:** An ability to solve complex Electronics and communication Engineering problems, using latest hardware and software tools, along with analytical skills to arrive cost effective and appropriate solutions.
PSO3	**Successful Career and Entrepreneurship:** An understanding of social-awareness & environmental-wisdom along with ethical responsibility to have a successful career and to sustain passion and zeal for real-world applications using optimal resources as an Entrepreneur.
Program Specific Outcomes (EEE)	
PSO1	**Professional Skills:** An ability to understand the basic concepts in Electronics & Communication Engineering and to apply them to various areas, like Electronics, Communications, Signal processing, VLSI, Embedded systems etc., in the design and implementation of complex systems.
PSO2	**Problem-Solving Skills:** An ability to solve complex Electronics and communication Engineering problems, using latest hardware and software tools, along with analytical skills to arrive cost effective and appropriate solutions.
PSO3	**Successful Career and Entrepreneurship:** An understanding of social-awareness & environmental-wisdom along with ethical responsibility to have a successful career and to sustain passion and zeal for real-world applications using optimal resources as an Entrepreneur.

ATTAINMENT OF PROGRAM OUTCOMES
& PROGRAM SPECIFIC OUTCOMES

S. No.	Experiment	Program Outcomes Attained	Program Specific Outcomes Attained			
			CSE	IT	ECE	EEE
1	**SEARCHING TECHNIQUES**	PO1, PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
2	**SORTING TECHNIQUES**	PO1, PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
3	**SORTING TECHNIQUES**	PO1, PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
4	**IMPLEMENTATION OF STACK AND QUEUE**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
5	**APPLICATIONS OF STACK**	PO3, PO4	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
6	**IMPLEMENTATION OF SINGLE LINKED LIST**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
7	**IMPLEMENTATION OF CIRCULAR SINGLE LINKED LIST**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
8	**IMPLEMENTATION OF DOUBLE LINKED LIST**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
9	**IMPLEMENTATION OF STACK USING LINKED LIST**	PO3, PO4	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
10	**IMPLEMENTATION OF QUEUE USING LINKED LIST**	PO3, PO4	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
11	**GRAPH TRAVERSAL TECHNIQUES**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2
12	**IMPLEMENTATION OF BINARY SEARCH TREE**	PO2, PO3	PSO1, PSO2	PSO1, PSO3	PSO2	PSO2

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad - 500 043

Certificate

This is to Certify that it is a bonafied record of Practical work done by Sri/Kum. ______________________________ bearing the Roll No. ____________________ of ____________ Class ______________________________ Branch in the ______________________ laboratory during the Academic year ________________ under our supervision.

Head of the Department **Lecture In-Charge**

External Examiner **Internal Examiner**

DATA STRUCTURES LABORATORY

II Semester: CSE / ECE / EEE / IT

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACS102	Foundation	-	-	3	2	30	70	100
Contact Classes: Nil	Tutorial Classes: Nil	Practical Classes: 36				Total Classes: 36		

COURSE OBJECTIVES:
The course should enable the students to:

I. **Understand** various data representation techniques in the real world.
II. **Implement** linear and non-linear data structures.
III. **Analyze** various algorithms based on their time and space complexity.
IV. **Develop** real-time applications using suitable data structure.
V. **Identify** suitable data structure to solve various computing problems.

LIST OF EXPERIMENTS

WEEK-1 SEARCHING TECHNIQUES

Write Python programs for implementing the following searching techniques.
a. Linear search
b. Binary search
c. Fibonacci search

WEEK-2 SORTING TECHNIQUES

Write Python programs for implementing the following sorting techniques to arrange a list of integers in ascending order.
a. Bubble sort
b. Insertion sort
c. Selection sort

WEEK-3 SORTING TECHNIQUES

Write Python programs for implementing the following sorting techniques to arrange a list of integers in ascending order.
a. Quick sort
b. Merge sort

WEEK-4 IMPLEMENTATION OF STACK AND QUEUE

Write Python programs to
a. Design and implement Stack and its operations using List.
b. Design and implement Queue and its operations using List.

WEEK-5 APPLICATIONS OF STACK

Write Python programs for the following:
a. Uses Stack operations to convert infix expression into postfix expression.
b. Uses Stack operations for evaluating the postfix expression.

WEEK-6 IMPLEMENTATION OF SINGLE LINKED LIST

a. Write Python programs for the following operations on Single Linked List.
 (i) Creation (ii) insertion (iii) deletion (iv) traversal
b. To store a polynomial expression in memory using single linked list.

WEEK-7 IMPLEMENTATION OF CIRCULAR SINGLE LINKED LIST

Write Python programs for the following operations on Circular Linked List.
(i) Creation (ii) insertion (iii) deletion (iv) traversal

WEEK-8 IMPLEMENTATION OF DOUBLE LINKED LIST

Write Python programs for the following:
Uses functions to perform the following operations on Double Linked List.
(i) Creation (ii) insertion (iii) deletion (iv) traversal in both ways.

WEEK-9 IMPLEMENTATION OF STACK USING LINKED LIST

Write a Python program to implement Stack using linked list.

WEEK-10 IMPLEMENTATION OF QUEUE USING LINKED LIST

Write a Python program to implement Linear Queue using linked list.

WEEK-11 GRAPH TRAVERSAL TECHNIQUES

Write Python programs to implement the following graph traversal algorithms:

a. Depth first search.
b. Breadth first search.

WEEK-12 IMPLEMENTATION OF BINARY SEARCH TREE

Write a Python program to perform the following:

a. Create a binary search tree.
b. Traverse the above binary search tree recursively in pre-order, post-order and in-order.
c. Count the number of nodes in the binary search tree.

LIST OF REFERENCE BOOKS:

1. Y Daniel Liang, “Introduction to Programming using Python”, Pearson.
2. Benjamin Baka, David Julian, “Python Data Structures and Algorithms”, Packt Publishers,2017.
3. Rance D. Necaise, “Data Structures and Algorithms using Python”, Wiley Student Edition.
4. Martin Jones, “Python for Complete Beginners”, 2015.
5. Zed A. Shaw, “Learn Python the Hard Way: a very simple introduction to the terrifyingly beautiful world of computers and code”, 3e, Addison-Wesley, 2014.
6. Hemant Jain, “Problem Solving in Data Structures and Algorithms using Python: programming interview guide”, 2016.

WEB REFERENCES:

1. https://docs.python.org/3/tutorial/datastructures.html
2. http://interactivepython.org/runestone/static/pythonds/index.html
3. http://www.tutorialspoint.com/data_structures_algorithms
4. http://www.geeksforgeeks.org/data-structures/
5. http://www.studytonight.com/data-structures/
6. http://www.coursera.org/specializations/data-structures-algorithms

WEEK-1

SEARCHING TECHNIQUES

1.1 OBJECTIVE:

a. Write a Python script to for implementing linear search technique.
b. Write a Python script to for implementing binary search technique.
c. Write a Python script to for implementing Fibonacci search technique.

1.2 RESOURCES:

Python 3.4.0

1.3 PROGRAM LOGIC:

Linear Search Algorithm

```
Algorithm linsrch (a[], x)
 { // a[1:n] is an array of n elements
       index := 0; flag := 0;
       while (index < n) do
       {
            if (x = a[index]) then
                {    flag := 1; break;
                }
               index ++;
       }
     if(flag =1)
           write(“Data found “);
    else
           write(“data not found”);
}
```

Example: Given a list of n elements and search a given element x in the list using linear search.

a. Start from the leftmost element of list a[] and one by one compare x with each element of list a[].
b. If x matches with an element, return the index.
c. If x doesn’t match with any of elements, return -1.

Consider a list with 10 elements and search for 9.

a = [56, 3, 249, 518, 7, 26, 94, 651, 23, 9]

Index →	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Iteration 1	56	3	249	518	7	26	94	651	23	9
Iteration 2	56	3	249	518	7	26	94	651	23	9
Iteration 3	56	3	249	518	7	26	94	651	23	9
Iteration 4	56	3	249	518	7	26	94	651	23	9
Iteration 5	56	3	249	518	7	26	94	651	23	9
Iteration 6	56	3	249	518	7	26	94	651	23	9
Iteration 7	56	3	249	518	7	26	94	651	23	9
Iteration 8	56	3	249	518	7	26	94	651	23	9
Iteration 9	56	3	249	518	7	26	94	651	23	9
Iteration 10	56	3	249	518	7	26	94	651	23	9

Binary Search Algorithm

```
Algorithm binsrch (a[], n, x)
 { // a[1:n] is an array of n elements
       low = 1;
       high = n;
       while (low < high) do
       {
             mid = (low + high)/2 ;
            if (x < a[mid]) then
                   high = mid – 1;
           else if (x > a[mid]) then
                   low = mid + 1;
           else
                return mid;
        }
      return 0;
}
```

Example: Given a sorted list of a[] of n elements, search a given element x in list.

a. Search a sorted list by repeatedly dividing the search interval in half. Begin with an interval covering the whole list.
b. If the search key is less than the item in the middle item, then narrow the interval to the lower half. Otherwise narrow it to the upper half.
c. Repeat the procedure until the value is found or the interval is empty.

Consider a sorted list a[] with 9 elements and the search key is 31.

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
11	23	31	33	65	68	71	89	100

Let the search key = 31.
First low = 0, high = 8, mid = (low + high) = 4
a[mid] = 65 is the centre element, but 65 > 31.
So now high = mid - 1= 4 - 1 = 3, low = 0, mid = (0 + 3) / 2 = 1

a[mid] = a[1] = 23, but 23 < 31.
Again low = mid +1 = 1 +1 =2, high = 3, mid = (2 + 3) /2 = 2
a[mid] = a[2] = 31 which is the search key, so the search is successful.

Fibonacci Search Algorithm

```
Algorithm  fib_Search (arr, x, n)
 { // arr[1:n] is an array of n elements
          M2 := 0 ;
          M1 := 1 ;
          M := M2 +M1 ;
          while (fibM < n) do
           {
                          M2 := M1;
                         M1: =M;
                          M := M2 + M1;
            }
          Offset: = -1;
          while (fibM > 1) do
           {
                          i := min(offset+M2, n-1);
                         if (arr[i] < x) then
                         {
                                     M := M1;
                                     M1 := M2;
                                     M2: = M -M1;
                                     offset = i
                         }
                          else if (arr[i] > x) then
                          {
                                     M:= M2;
                                     M1: = M1 -M2;
                                     M2 := M - M1;
                         }
                         else
                                      return i;
          }
          if(M1 and arr[offset+1] = x) then
                         return offset+1;

          return -1;
}
```

Example: Fibonacci Search is a comparison-based technique that uses Fibonacci numbers to search an element in a sorted array.
Fibonacci Numbers are recursively defined as F(n) = F(n-1) + F(n-2), F(0) = 0, F(1) = 1.

First few Fibonacci Numbers are 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, …
Let a[0..n-1] be the input list and element to be searched be x.

1. Find the smallest Fibonacci Number greater than or equal n. Let this number be M (m’th Fibonacci Number). Let the two Fibonacci numbers preceding it be M1 [(m-1)’th Fibonacci Number] and M2 [(m-2)’th Fibonacci Number].
1. While the array has elements to be inspected: Compare x with the last element of the range covered by M2
2. If x matches, return index.
3. Else If x is less than the element, move the three Fibonacci variables two Fibonacci down, indicating elimination of approximately rear two-third of the remaining array.
4. Else x is greater than the element, move the three Fibonacci variables one Fibonacci down. Reset offset to index. Together these indicate elimination of approximately front one-third of the remaining array.
2. Since there might be a single element remaining for comparison, check if M1 is 1. If Yes, compare x with that remaining element. If match, return index.

Consider a list a[] with 11 elements and the search element is 85.
n = 11

Index	0	1	2	3	4	5	6	7	8	9	10
a[i]	10	22	35	40	45	50	80	82	85	90	100

Smallest Fibonacci number greater than or equal to 11 is 13.
M2 = 5, M1 = 8, M = M1 + M2 = 13
Initialize offset = 0
Check the element at index i = min(offset + M2, n)

M2	**M1**	**M**	**Offset**	**I = min(offset + M2, n)**	**A[i]**	**Consequence**
5	8	13	0	5	45	Move one down, reset offset
3	5	8	5	8	82	Move one down, reset offset
2	3	5	8	10	90	Move two down
1	1	2	8	9	85	Return i

1.4 PROCEDURE:

1. Create: Open a new file in Python shell, write a program and save the program with .py extension.
2. Execute: Go to Run -> Run module (F5)

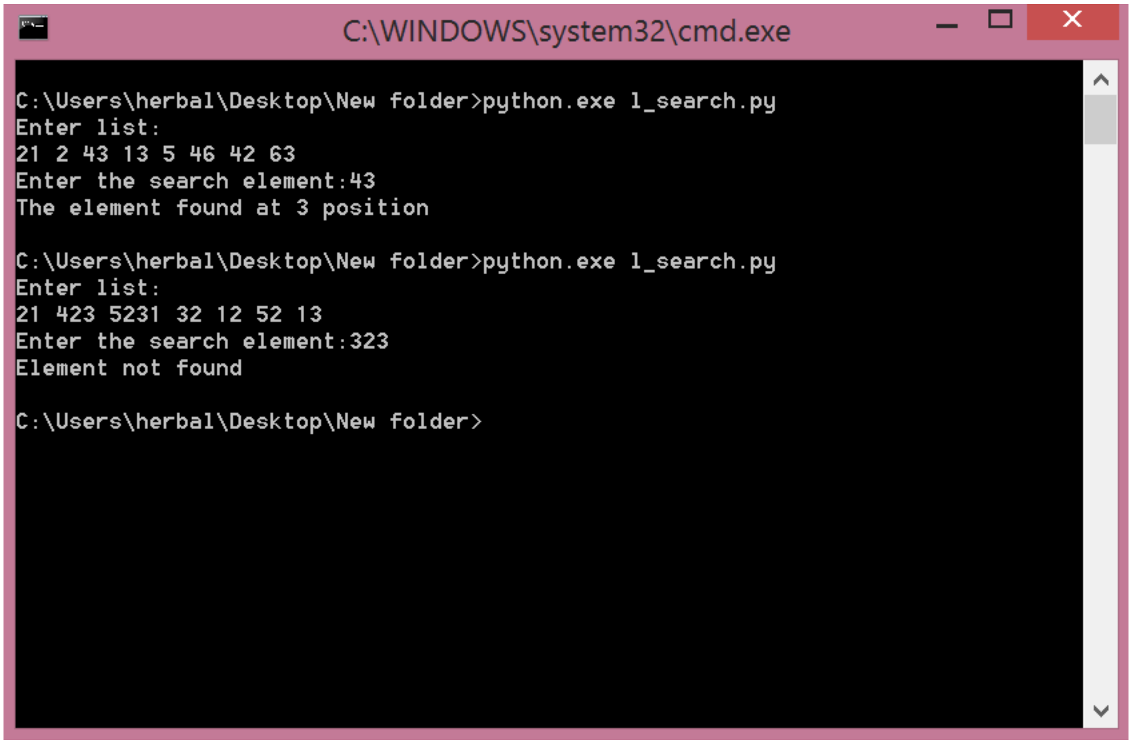
1.5 SOURCE CODE:

Implementation of Linear Search

```
def l_search(a,x,l,n):
    if l<n:
        if a[l]==x:
            print("The element found at",l+1,"position")
        else:
```

```
        l_search(a,x,l+1,n)
  else:
      print("Element not found")

print("Enter list:")
a=[int(b) for b in input().split()]
x=eval(input("Enter the search element:"))
n=len(a)
l_search(a,x,0,n)
```

Output:



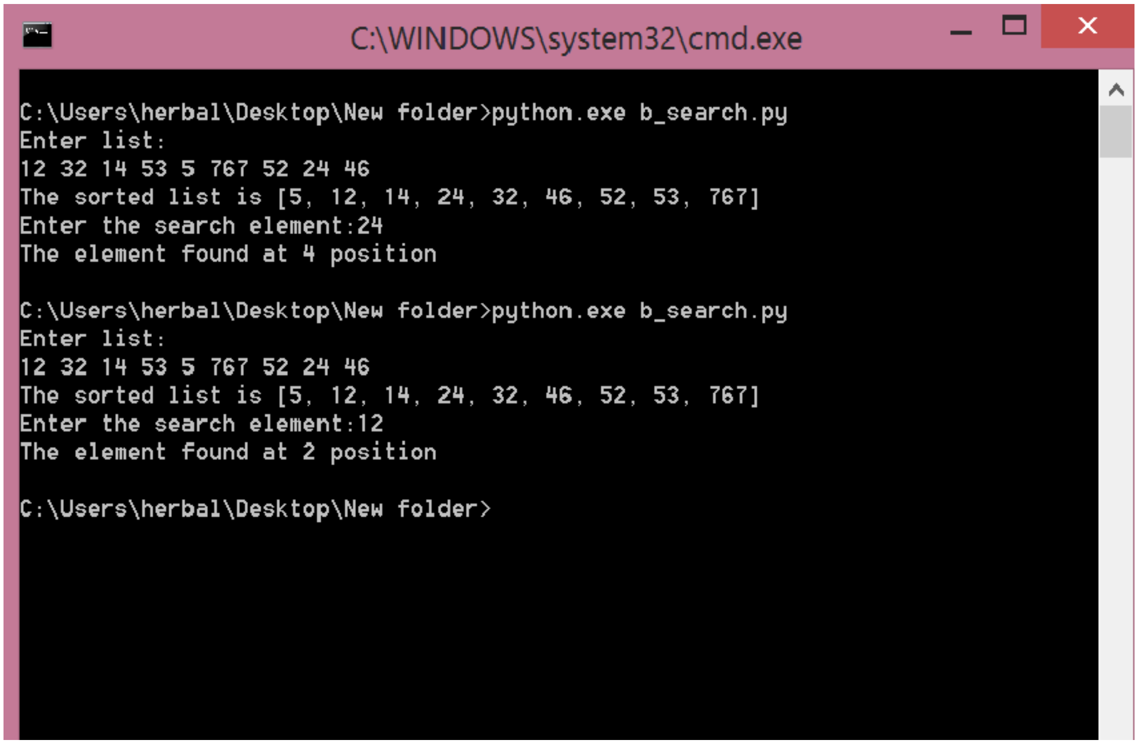
```
C:\Users\herbal\Desktop\New folder>python.exe l_search.py
Enter list:
21 2 43 13 5 46 42 63
Enter the search element:43
The element found at 3 position

C:\Users\herbal\Desktop\New folder>python.exe l_search.py
Enter list:
21 423 5231 32 12 52 13
Enter the search element:323
Element not found

C:\Users\herbal\Desktop\New folder>
```

Implementation of Binary Search

```
def b_search(a,x,l,n):
  if l<=n:
      mid=(l+n)//2
      if a[mid]==x:
          print("The element found at",mid+1,"position")
      else:
          if a[mid]>x:
              b_search(a,x,l,mid-1)
          else:
              b_search(a,x,mid+1,n)
  else:
      print("Element not found")

print("Enter list:")
```

```
a=[int(b) for b in input().split()]
list.sort(a)
print("the sorted list is",a)
x=eval(input("Enter the search element:"))
n=len(a)
b_search(a,x,0,n)
```

Output:

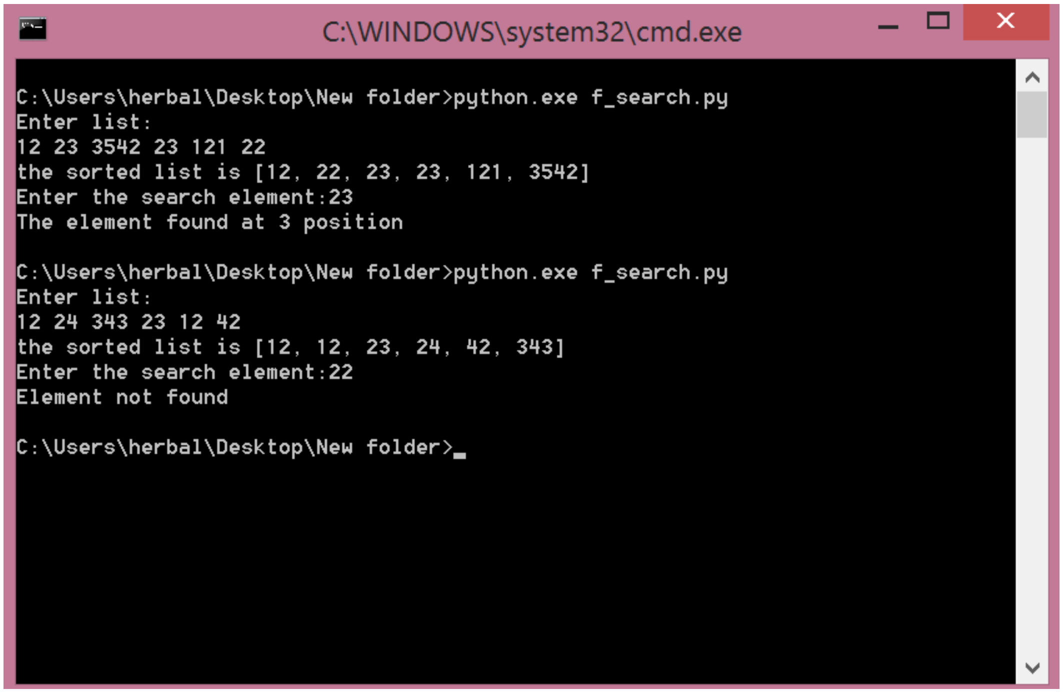


```
C:\Users\herbal\Desktop\New folder>python.exe b_search.py
Enter list:
12 32 14 53 5 767 52 24 46
The sorted list is [5, 12, 14, 24, 32, 46, 52, 53, 767]
Enter the search element:24
The element found at 4 position

C:\Users\herbal\Desktop\New folder>python.exe b_search.py
Enter list:
12 32 14 53 5 767 52 24 46
The sorted list is [5, 12, 14, 24, 32, 46, 52, 53, 767]
Enter the search element:12
The element found at 2 position

C:\Users\herbal\Desktop\New folder>
```

Implementation of Fibonacci Search

```
def f_search(a,x,n):
    f0=0
    f1=1
    f2=f0+f1
    while f2<n:
        f0=f1
        f1=f2
        f2=f0+f1
    offset=-1
    while f2>1:
        i = min(offset+f2, n-1)
        if (a[i]<x):
            f2=f1
            f1=f0
            f0=f2-f1
            offset = i
        elif (a[i]>x):
            f2=f0
            f1=f1-f2
            f0=f2-f1
        else :
            return i
    if(f1 and a[offset+1]==x):
        return offset+1
    return -1

print("Enter list:")
a=[int(b) for b in input().split()]
list.sort(a)
print("the sorted list is",a)
x=eval(input("Enter the search element:"))
n=len(a)
pos=f_search(a,x,n)
if pos>=0:
    print("The element found at",pos+1,"position")
else:
    print("Element not found")
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe f_search.py
Enter list:
12 23 3542 23 121 22
the sorted list is [12, 22, 23, 23, 121, 3542]
Enter the search element:23
The element found at 3 position

C:\Users\herbal\Desktop\New folder>python.exe f_search.py
Enter list:
12 24 343 23 12 42
the sorted list is [12, 12, 23, 24, 42, 343]
Enter the search element:22
Element not found

C:\Users\herbal\Desktop\New folder>_
```

1.6 PRE LAB VIVA QUESTIONS:

1. Define searching?
2. Define a list?
3. List out different types of searching techniques?
4. Differentiate between list and dictionary?
5.

1.7 LAB ASSIGNMENT:

1. A person has registered for voter id , he received a voter number and he need to check whether it exist in the voter or not. Use a binary searching in a recursive way to find whether the voter number exist in the list or not.
2. Use linear search technique to search for a key value in a given list of characters and print the message found or not.

1.8 POST LAB VIVA QUESTIONS:

1. Find the time complexity of linear search?
2. Find the time complexity of binary search?
3. Find the time complexity of Fibonacci search?

WEEK - 2

SORTING TECHNIQUES

2.1 OBJECTIVE:

a. Write Python script for implementing Bubble sort techniques to arrange a list of integers in ascending order.

b. Write Python script for implementing insertion sort techniques to arrange a list of integers in ascending order.

c. Write Python script for implementing selection sort techniques to arrange a list of integers in ascending order.

2.2 RESOURCES:

Python 3.4.0

2.3 PROGRAM LOGIC:

Bubble Sort Algorithm

```
Algorithm  bubblesort ( x[],  n)
 { // x[1:n] is an array of n elements
      for i := 0 to  n do
      {     for j := 0 to  n–i-1 d0
              { if (x[j] > x[j+1])
                   {
                  temp = x[j];
                 x[j] = x[j+1];
                 x[j+1] = temp;
                  }
     }       }
}
```

Example: Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are not in order.

First Pass

5	1	4	2	8	Compare the first two elements, and swaps since 5 > 1
1	5	4	2	8	Compare 5 and 4 and swap since 5 > 4
1	4	5	2	8	Compare 5 and 2 and swap since 5 > 2
1	4	2	5	8	Compare 5 and 8 and since 5 < 8, no swap.

Second Pass

1	4	2	5	8	Compare the first two elements, and as 1 < 4, no swap.
1	4	2	5	8	Compare 4 and 2, swap since 4 > 2
1	2	4	5	8	Compare 4 and 5, no swap.
1	2	4	5	8	Compare 5 and 8 and no swap.

Third Pass

1	2	4	5	8	Compare the first two elements and no swap.
1	2	4	5	8	Compare 2 and 4, no swap.
1	2	4	5	8	Compare 4 and 5, no swap.
1	2	4	5	8	Compare 5 and 8, no swap.

Fourth Pass

1	2	4	5	8	Compare the first two elements and no swap.
1	2	4	5	8	Compare 2 and 4, no swap.
1	2	4	5	8	Compare 4 and 5, no swap.
1	2	4	5	8	Compare 5 and 8, no swap.

Insertion Sort Algorithm

```
Algorithm insertionsort (a, n)
{//sort the array a[1:n] in ascending order
    for j:=2 to n do
    {
           item:=a[j];
           i=j-1;
       while((i>1) and (item< a[i])) do
      {
             a[i+1]:=a[i];
            i:=i-1;
      }
      a[i+1]:=item;
   }
}
```

Example: This is an in-place comparison-based sorting algorithm. Here, a sub-list is maintained which is always 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. Consider an unsorted list with 8 elements.

14	33	27	10	35	19	42	44	Compares the first two elements 14 and 33 and these element are already in ascending order. Now 14 is in sorted sub-list.
14	33	27	10	35	19	42	44	Compare 33 with 27 and 33 is not in correct position. So swap 33 with 27.
14	27	33	10	35	19	42	44	Now we have 14 and 27 in the sorted sub-list.
14	27	33	10	35	19	42	44	Next compare 33 with 10.
10	14	27	33	35	19	42	44	Compare 27 with 10 and 14 with 10. Insert 10 in the proper place.
10	14	27	33	35	19	42	44	Compare 33 and 35, no swap.
10	14	27	33	35	19	42	44	Compare 35 with 19
10	14	27	33	35	19	42	44	Compare 33 with 19, 27 with 19,
10	14	19	27	33	35	42	44	Compare 35 with 42, no swap.
10	14	19	27	33	35	42	44	Compare 42 with 44, no swap.

Selection Sort Algorithm

```
Algorithm selectionSort ( low, high )
 { //a[low : high] is an array  of size n
     i=0, j=0, temp=0, ;
   for i: =low to high do
    {
          minindex = i;
          for  j: =i+1 to  high do
          {
                     if( a[j] < a[minindex] ) then
```

```
                minindex := j; }
            temp := a[i];
            a[i] := a[minindex];
            a[minindex] := temp;
        }
}
```

Example: The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. In every iteration of selection sort, the minimum element from the unsorted sub-array is picked and moved to the sorted sub-array.

Consider a list a = [64, 25, 12, 22, 11]

Index	**0**	**1**	**2**	**3**	**4**	**Remarks**
List	64	25	12	22	11	Find the minimum element in a[0 .. 4] and place it at beginning.
Iteration 1	11	25	12	22	64	Find the minimum element in a[1 .. 4] and place it at beginning of a[1 .. 4]
Iteration 2	11	12	25	22	64	Find the minimum element in a[2 .. 4] and place it at beginning of a[2 .. 4]
Iteration 3	11	12	22	25	64	Find the minimum element in a[3 .. 4] and place it at beginning of a[3 .. 4]
Iteration 4	11	12	22	25	64	Finally the list is sorted.

2.4 PROCEDURE:

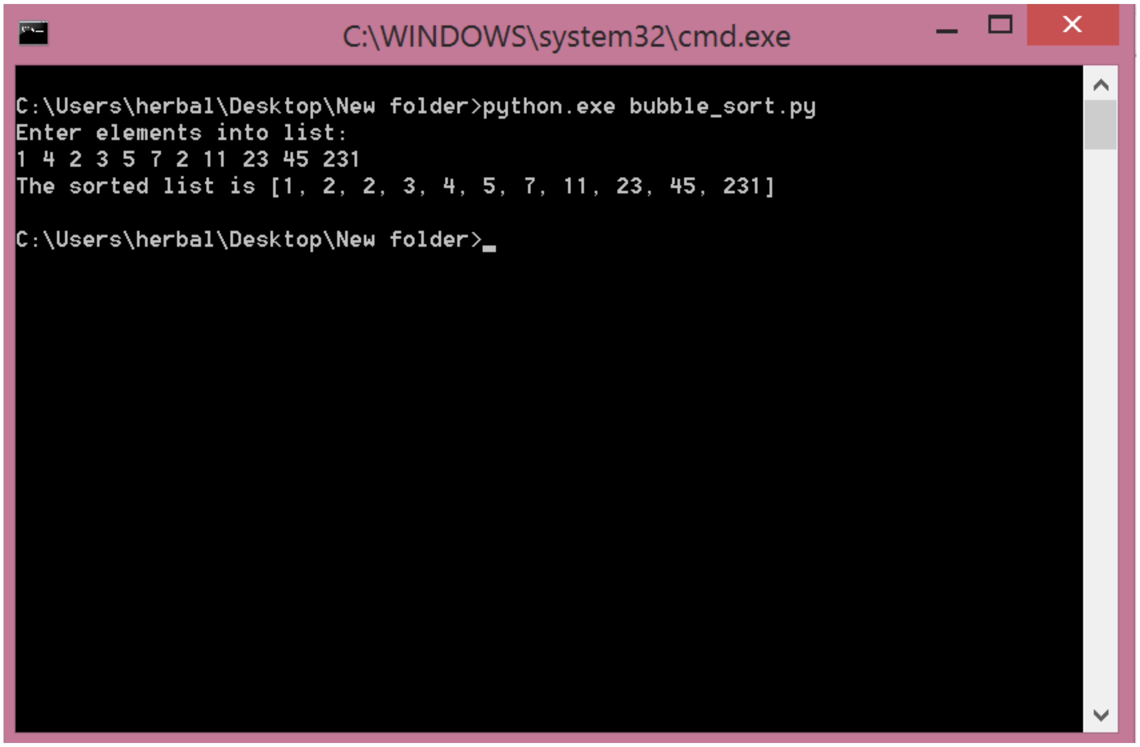
1. Create: open Python shell write a program after that save the program with .py extension.
2. Execute: F5

2.5 SOURCE CODE:

Program for implementing Bubble Sort

```
def b_sort(a):
    n = len(a)
    for i in range(n):
        for j in range(0, n-i-1):
            if a[j] > a[j+1] :
                a[j], a[j+1] = a[j+1], a[j]

print("Enter elements into list:")
a=[int(x) for x in input().split()]
b_sort(a)
print("The sorted list is ",a)
```

Output:

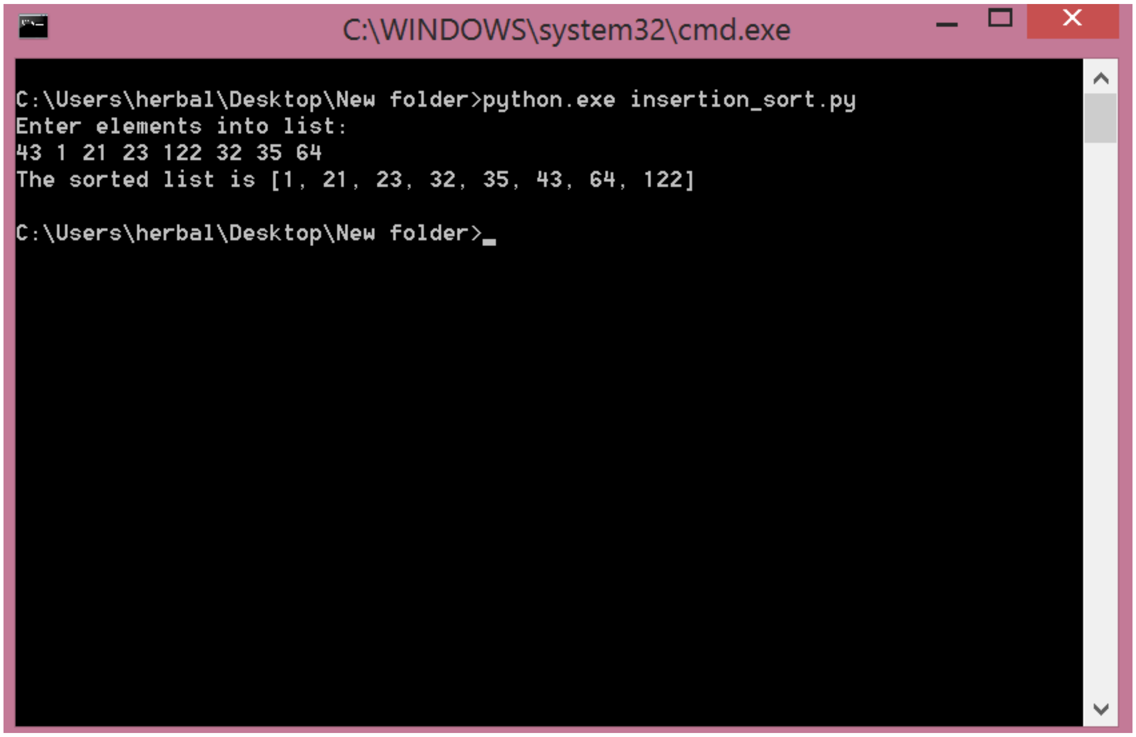


```
C:\Users\herbal\Desktop\New folder>python.exe bubble_sort.py
Enter elements into list:
1 4 2 3 5 7 2 11 23 45 231
The sorted list is [1, 2, 2, 3, 4, 5, 7, 11, 23, 45, 231]

C:\Users\herbal\Desktop\New folder>_
```

Program for implementing Insertion Sort

```
def i_sort(a):
    for i in range(1,len(a)):
        temp=a[i]
        pos=i
        while pos>0 and a[pos-1]>temp:
            a[pos]=a[pos-1]
            pos-=1
        a[pos]=temp

print("Enter elements into list:")
a=[int(x) for x in input().split()]
i_sort(a)
print("The sorted list is",a)
```

Output:

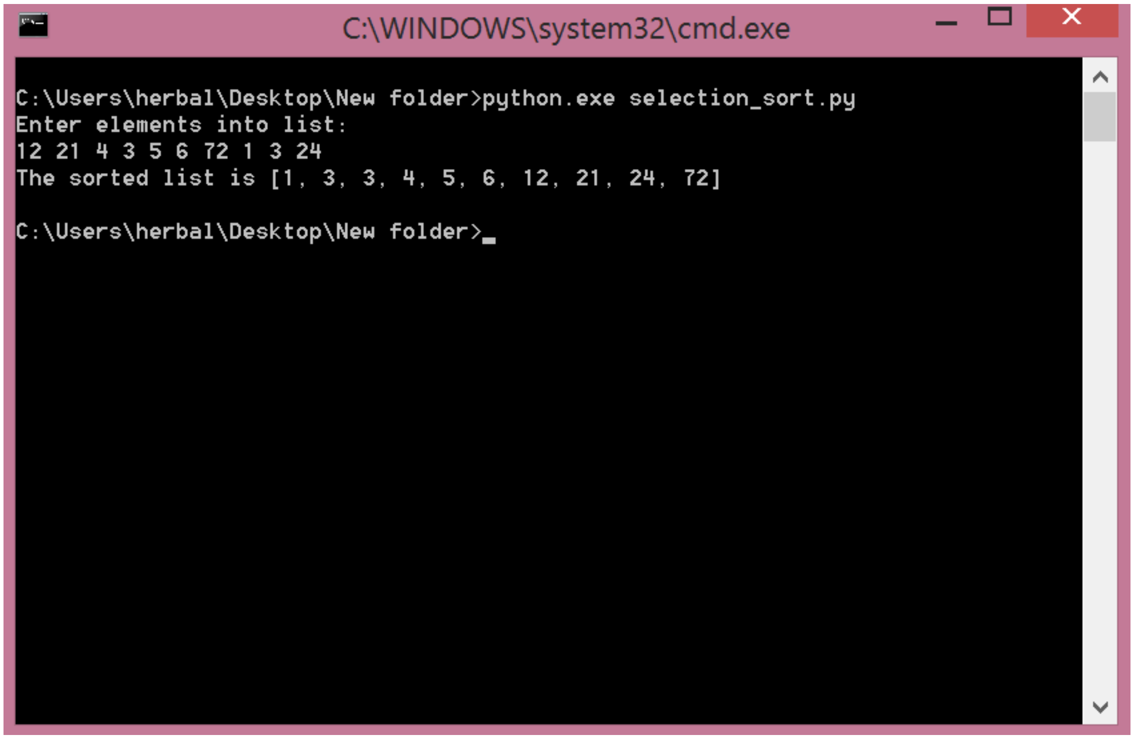


```
C:\Users\herbal\Desktop\New folder>python.exe insertion_sort.py
Enter elements into list:
43 1 21 23 122 32 35 64
The sorted list is [1, 21, 23, 32, 35, 43, 64, 122]

C:\Users\herbal\Desktop\New folder>_
```

Program for implementing Selection Sort

```
def s_sort(a):
    for i in range(len(a)):
        least=i
        for k in range(i+1,len(a)):
            if a[k]<a[least]:
                least=k
        swap(a,least,i)
def swap(a,least,i):
    temp=a[least]
    a[least]=a[i]
    a[i]=temp

print("Enter elements into list:")
a=[int(x) for x in input().split()]
s_sort(a)
print("The sorted list is",a)
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe selection_sort.py
Enter elements into list:
12 21 4 3 5 6 72 1 3 24
The sorted list is [1, 3, 3, 4, 5, 6, 12, 21, 24, 72]

C:\Users\herbal\Desktop\New folder>
```

2.6 PRE LAB VIVA QUESTIONS:

1. Define Sorting?
2. Differentiate between internal sorting and external sorting?
3. Explain the basic idea of Bubble Sort?
4. Explain the concept and application of Insertion Sort?

2.7 LAB ASSIGNMENT:

1. Formulate a program that implement Bubble sort, to sort a given list of integers in descending order.
2. Compose a program that implement Insertion sort, to sort a given list of integers in descending order.
3. Write a program that implement Selection sort, to sort a given list of integers in ascending order.
4. Formulate a program to sort N names using selection sort.
5. Write a program to sort N employee records based on their salary using insertion sort.
6. A class contains 50 students who acquired marks in 10 subjects write a program to display top 10 students roll numbers and marks in sorted order by using bubble sorting technique.

2.8 POST LAB VIVA QUESTIONS:

1. Write the time complexity of Bubble Sort?
2. Write the time complexity of Insertion Sort?
3. Write the other name of Bubble Sort?
4. Write the time complexity of Selection Sort?
5. Write the procedure used to sort the elements using Selection Sort?

WEEK- 3

SORTING TECHNIQUES

3.1 OBJECTIVE:

1. Write Python programs for implementing Quick sort technique to arrange a list of integers in ascending order.
2. Write Python programs for implementing merge sort technique to arrange a list of integers in ascending order.

3.2 RESOURCES:

Python 3.4.0

3.3 PROGRAM LOGIC:

Algorithm for Quick Sort

```
Algorithm QuickSort (p, q)
// sorts the elements a[p],…….,a[q] which resides in the global array a[1 : n] into ascending order.
// a[n+1] is considered to be defined and must be >= all the elements in a[1 : n].
{
        if (p<q) then   // if there are more than one element
        {
                //divide p into two sub-problems
                j := partition(a, p, q+1);
                // j is the position of the partitioning element.
                //solve the sub-problems.
                QuickSort (p, j-1);
                QuickSort (j+1, q);
        }
}
```

```
Algorithm Partition(a, m, p)
// within a[m], a[m+1], ………, a[p-1] the elements are rearranged in such a manner that if initially
// t = a[m], then after completion a[q] = t for some q between m and p-1, a[k] <= t for m<=k<=q,
// and a[k]>=t for q<k<p. q is returned, set a[p] = ∞
{
        v := a[m]; i:=m; j:=p;
        repeat
        {
                repeat
                        i:= i+1;
                until (a[i] >=v);
                repeat
                        j:=j-1
                until (a[j]<=v);
                if( i<j) then interchange(a, i, j);
        }until(i>=j);
        a[m]:= a[j]; a[j]=v; return j;
}
```

```
Algorithm Interchange(a, i, j)
//exchange a[i] and a[j]
{
        p:=a[i];
        a[i]:=a[j];
        a[j]:=p;
}
```

Example: Quick sort is a divide and conquer algorithm. Quick sort first divides a large list into two smaller sub-lists: the low elements and the high elements. Quick sort can then recursively sort the sub-lists.

The steps are:

1. Pick an element, called a pivot, from the list.
2. Reorder the list so that all elements with values less than the pivot come before the pivot, while all elements with values greater than the pivot come after it. After this partitioning, the pivot is in its final position. This is called the partition operation.
3. Recursively apply the above steps to the sub-list of elements with smaller values and separately the sub list of elements with greater values.

Step-by-step example:

0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Remarks**
38	08	16	06	79	57	24	56	02	58	04	70	45	
Pivot				Up						Down			Swap up and down
38	08	16	06	04	57	24	56	02	58	79	70	45	
Pivot					Up			Down					Swap up and down
38	08	16	06	04	02	24	56	57	58	79	70	45	
Pivot						Down	Up						Swap Pivot and down
24	08	16	06	04	02	38	56	57	58	79	70	45	
Pivot					Down	Up							Swap Pivot and down
02	08	16	06	04	24	38							
Pivot Down	Up												Swap Pivot and down
02	08	16	06	04	24	38							
	Pivot	Up		Down									Swap up and down
02	08	04	06	16	24	38							
	Pivot		Down	Up									Swap Pivot and down
02	06	04	08	16	24	38							
	Pivot	Down	Up										Swap Pivot and down
02	04	06	08	16	24	38							Left sub-list is sorted

Repeat the similar procedure for right sub-list also

Algorithm for Merge sort

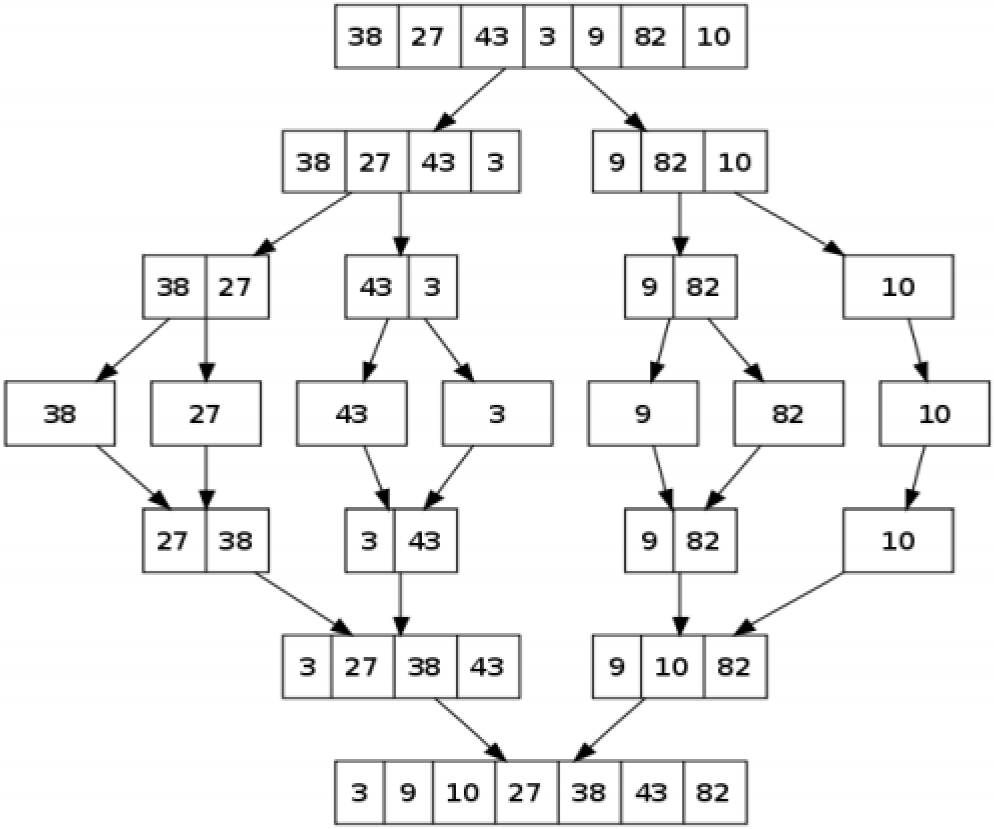
```
Algorithm MergeSort (low, high)
//a[low : high] is a global array to be sorted.
//Small(P) id true if there is only one element to sort.
{
    If(low < high) then  // if there are more than one element
    {
        //Divide P into sub-problems
        //Find where to split the set.
        Mid := (low + high) / 2;
        //Solve the sub-problems.
        MergeSort (low, mid);
        MergeSort (mid+1, high);
        //Combine the solutions
        Merge(low, mid, high);
    }
}
```

```
Algorithm Merge(low, mid, high)
// a[low: high] is a global array containing two sorted subsets in a[low: mid] and in a[mid+1: high].
//The goal is to merge these two sets into a single set residing in a[low: high]. b[] is an auxiliary
//global array.
{
    h:=low; i:=low; j:=mid+1;
    while ((h<=mid) and (j<=high)) do
    {
        if(a[h] <= a[j]) then
        {
            b[i]:=a[h];
            h:=h+1;
        }
        else
        {
            b[i]:=a[j];
            j:=j+1;
        }
        i:=i+1;
    }
    if (h>mid) then
        for k:=j to high do
        {
            b[i]:= a[k]; i:= i+1;
        }
    else
        for k:=h to mid do
        {
            b[i]:= a[k]; i:= i+1;
        }
    for k:= low to high do a[k] = b[k]

}
```

Example: This is a divide and conquer algorithm. Merge sort works as follows :

1. Divide the input which we have to sort into two parts in the middle. Call it the left part and right part.
2. Sort each of them separately by using the same function recursively.
3. Then merge the two sorted parts.

Step-by-step example:



3.4 PROCEDURE:

1. Create: open Python shell write a program after that save the program with .py extension.
2. Execute: F5

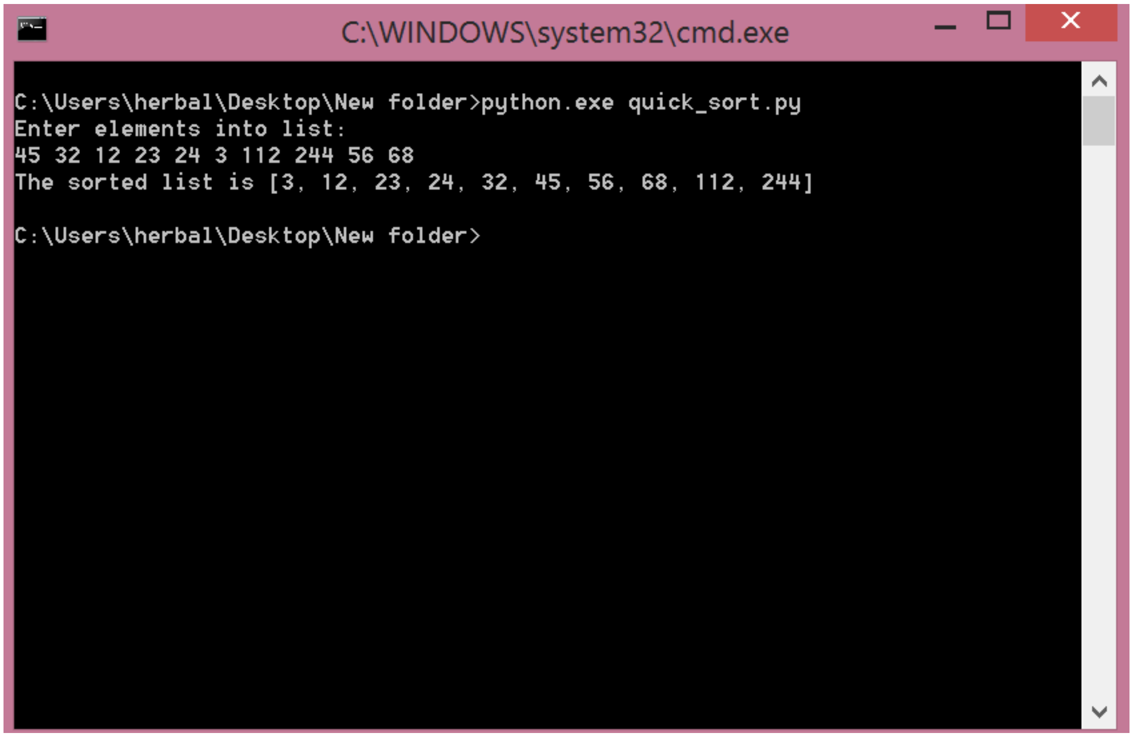
3.5 SOURCE CODE:

Program for implementation of Quick Sort

```
def q_sort(a,low,high):
    if low<high:
        pivotpos=partition(a,low,high)
        q_sort(a,low,pivotpos-1)
        q_sort(a,pivotpos+1,high)

def partition(a,low,high):
    pivotvalue=a[low]
    up=low+1
```

```
        down=high
        done=False
        while not done:
            while up<=down and a[up]<=pivotvalue:
                up+=1
            while down>=up and a[down]>=pivotvalue:
                down-=1
            if down<up:
                done=True
            else:
                temp=a[up]
                a[up]=a[down]
                a[down]=temp
        temp=a[low]
        a[low]=a[down]
        a[down]=temp
        return down

    print("Enter elements into list:")
    a=[int(x) for x in input().split()]
    high=len(a)
    q_sort(a,0,high-1)
    print("The sorted list is",a)
```

Output:

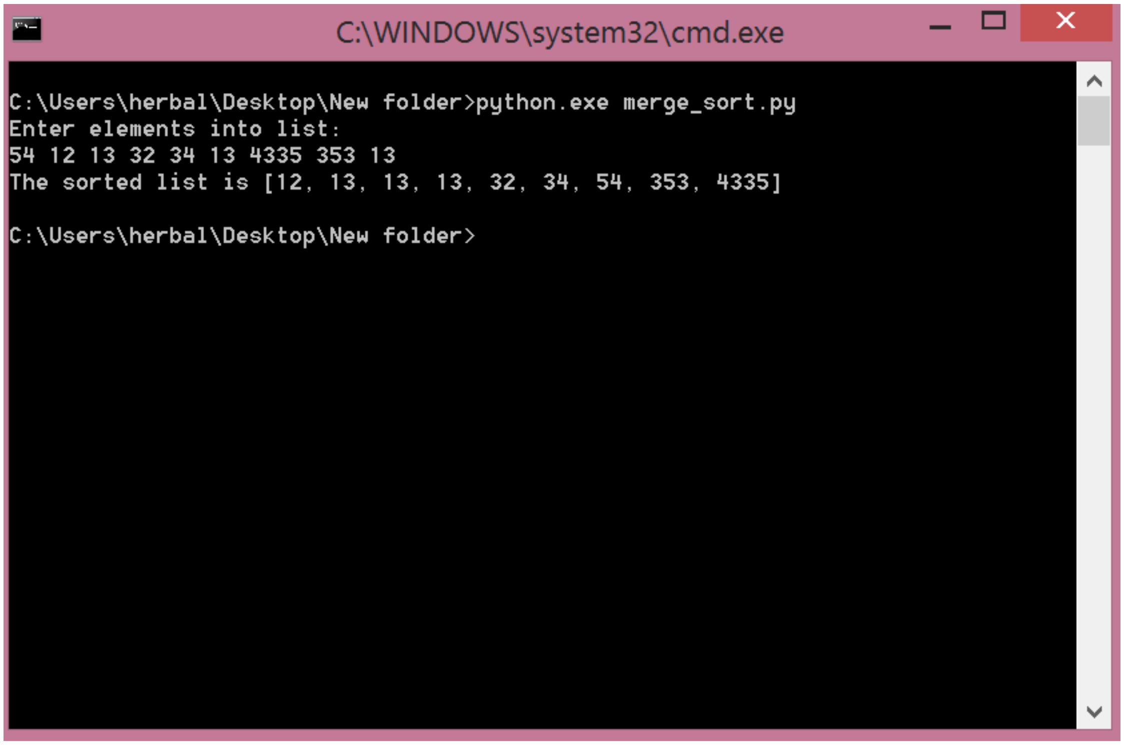


```
C:\Users\herbal\Desktop\New folder>python.exe quick_sort.py
Enter elements into list:
45 32 12 23 24 3 112 244 56 68
The sorted list is [3, 12, 23, 24, 32, 45, 56, 68, 112, 244]

C:\Users\herbal\Desktop\New folder>
```

Program for implementing Merge Sort

```
def m_sort(a):
    for i in range(len(a)):
        if i>1:
            mid=len(a)//2
            l_half=a[:mid]
            r_half=a[mid:]
            m_sort(l_half)
            m_sort(r_half)
            i=j=k=0
            while i<len(l_half) and j<len(r_half):
                if l_half[i]<r_half[j]:
                    a[k]=l_half[i]
                    i+=1
                else:
                    a[k]=r_half[j]
                    j+=1
                k+=1
            while i<len(l_half):
                a[k]=l_half[i]
                i+=1
                k+=1
            while j<len(r_half):
                a[k]=r_half[j]
                j+=1
                k+=1

print("Enter elements into list:")
a=[int(x) for x in input().split()]
m_sort(a)
print("The sorted list is",a)
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe merge_sort.py
Enter elements into list:
54 12 13 32 34 13 4335 353 13
The sorted list is [12, 13, 13, 13, 32, 34, 54, 353, 4335]

C:\Users\herbal\Desktop\New folder>
```

3.7 PRE-LAB VIVA QUESTIONS:

1. Write the advantage of merge sort ?
2. Write the advantage of quick sort?
3. Differentiate between merging and sorting?
4. How merge sort works?

3.8 LAB ASSIGNMENT:

1. Apply the quick sort on the following elements 21, 11, 5, 78, 49, 54, 72, 88.
2. Apply the merge sort on the following elements 21, 11, 5, 78, 49, 54,72, 88, 56, 28,10.

3.9 POST-LAB VIVA QUESTIONS:

1. Write the time complexity of Merge Sort?
2. Which sorting technique is an in-place sort that requires only O(n log n) operations regardless of the order of the input?
3. List the application of merge sort?

WEEK- 4

IMPLEMENTATION OF STACK AND QUEUE

4.1 OBJECTIVE:

a. Write a Python program to implement Stack and its operations using list.
b. Write a Python program to implement Queue and its operations using list.

4.2 RESOURCES:

Python 3.4.0

1.3 PROGRAM LOGIC:

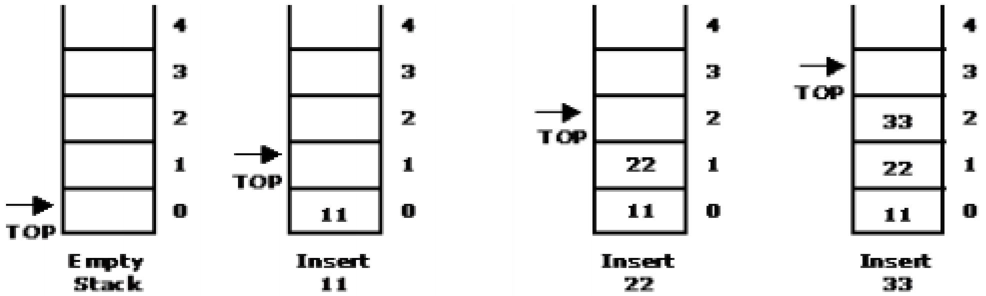
Procedure for Stack using List

1. STACK: Stack is a linear data structure which works under the principle of last in first out. Basic operations: push, pop, display.
2. PUSH: if (top==MAX), display Stack overflow. Otherwise reading the data and making stack [top] =data and incrementing the top value by doing top++.
3. Pop: if (top==0), display Stack underflow. Otherwise printing the element at the top of the stack and decrementing the top value by doing the top.
4. DISPLAY: If (top==0), display Stack is empty. Otherwise printing the elements in the stack from stack [0] to stack [top].

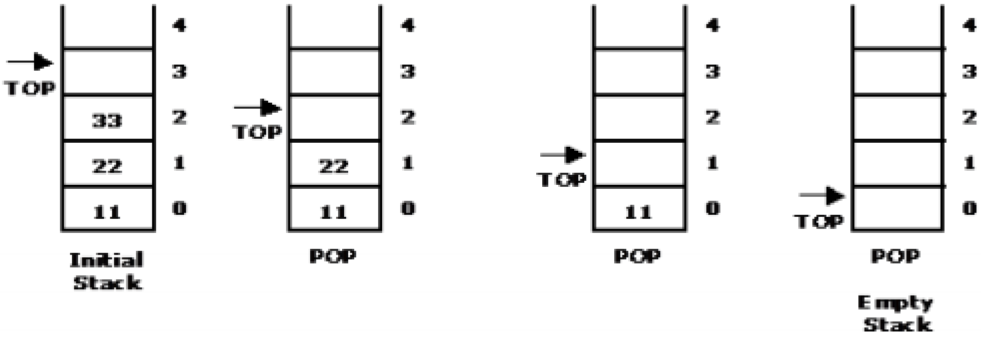
Procedure for Queue using List

1. QUEUE: Queue is a linear data structure which works under the principle of first in first out. Basic operations: Insertion, deletion, display.
2. Inserion: if (rear==MAX), display Queue is full. Else reading data and inserting at queue [rear], and doing rear++.
3. Deletion: if (front==rear), display Queue is empty .Else printing element at queue [front] and doing front++.
4. Display: if (front==rear) ,display No elements in the queue .Else printing the elements from queue[front] to queue[rear].

Example: Consider a stack with 5 elements capacity. When an element is added to a stack, the operation is performed by Push().



When an element is taken off from the stack, the operation is performed by Pop().



1.4 PROCEDURE:

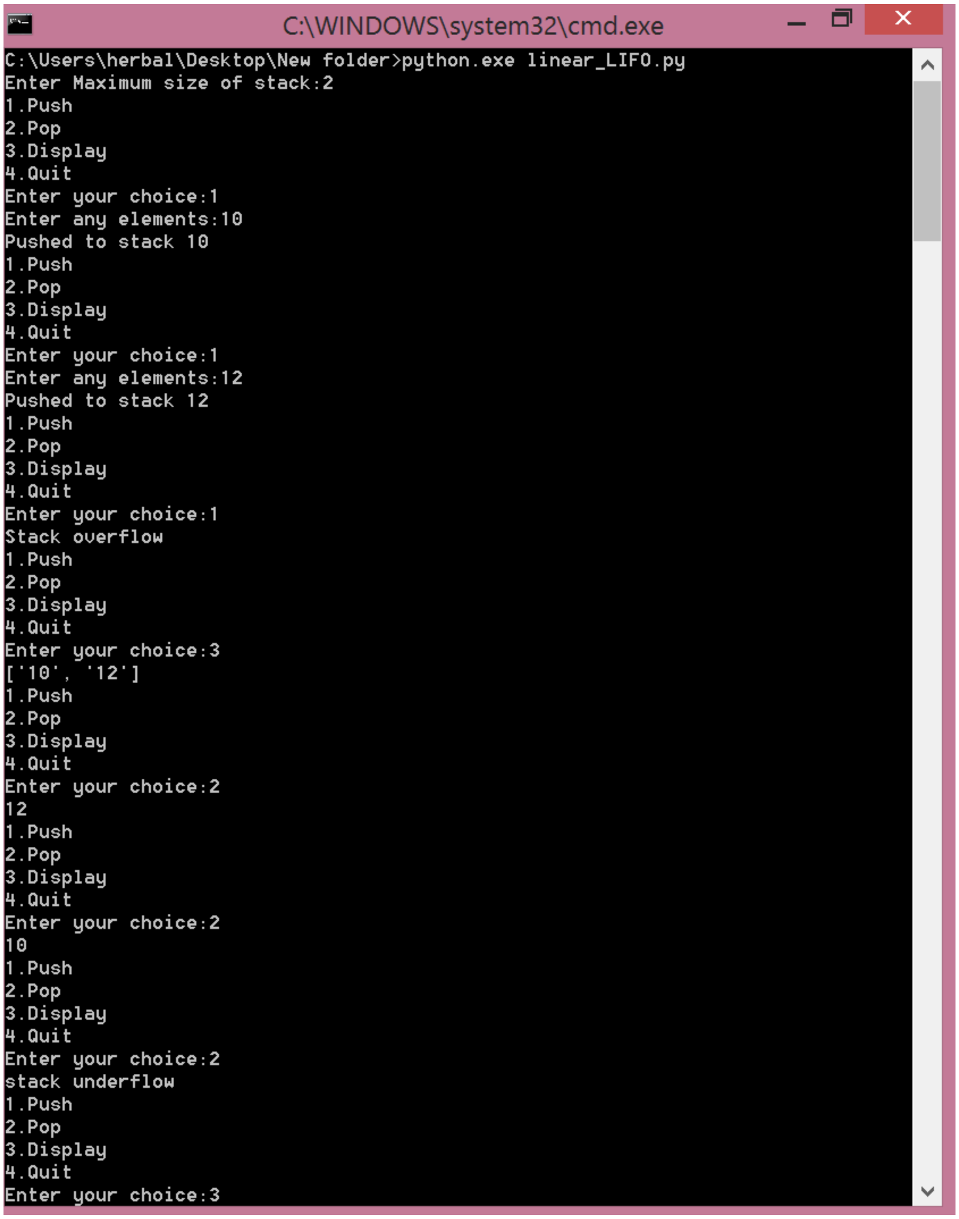
1. Create: open Python shell write a program after that save the program with .py extension.
2. Execute: F5

4.5 SOURCE CODE:

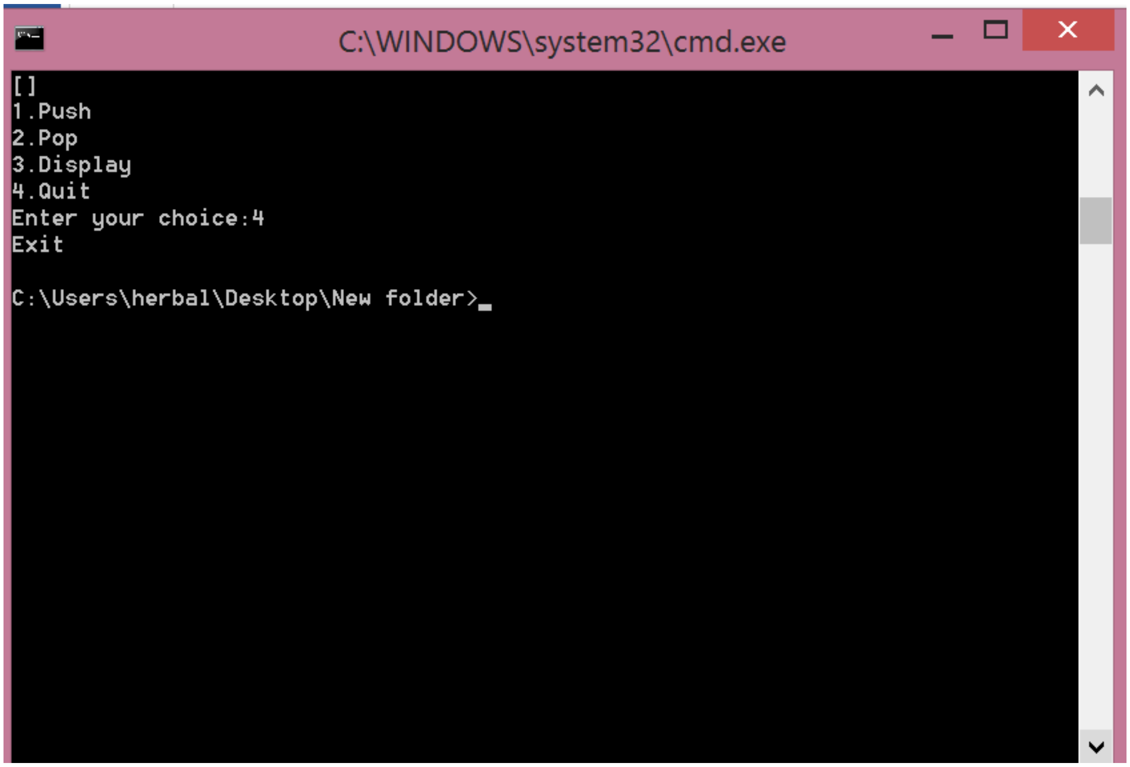
Program for implementing Stack using list

```
top=0
mymax=eval(input("Enter Maximum size of stack:"))
def createStack():
    stack=[]
    return stack
def isEmpty(stack):
    return len(stack)==0
def Push(stack,item):
    stack.append(item)
    print("Pushed to stack",item)
def Pop(stack):
    if isEmpty(stack):
        return "stack underflow"
    return stack.pop()
stack=createStack()
while True:
    print("1.Push")
    print("2.Pop")
    print("3.Display")
    print("4.Quit")
    ch=int(input("Enter your choice:"))
    if ch==1:
        if top<mymax:
            item=input("Enter any elements:")
            Push(stack,item)
            top+=1
        else:
            print("Stack overflow")
    elif ch==2:
        print(Pop(stack))
    elif ch==3:
        print(stack)
    else:
        print("Exit")
        break
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe linear_LIFO.py
Enter Maximum size of stack:2
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:1
Enter any elements:10
Pushed to stack 10
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:1
Enter any elements:12
Pushed to stack 12
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:1
Stack overflow
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:3
['10', '12']
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:2
12
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:2
10
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:2
stack underflow
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:3
```



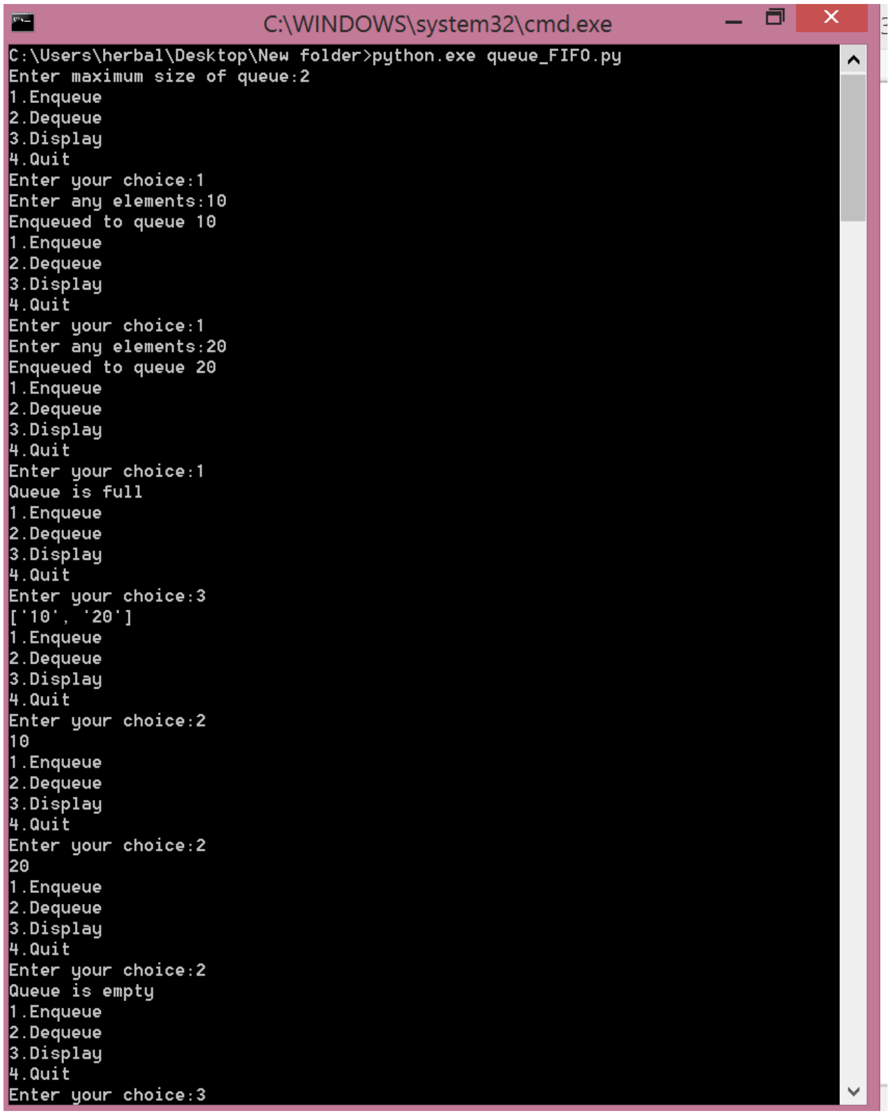
```
[]
1.Push
2.Pop
3.Display
4.Quit
Enter your choice:4
Exit

C:\Users\herbal\Desktop\New folder>_
```

Program for implementing Linear Queue using list

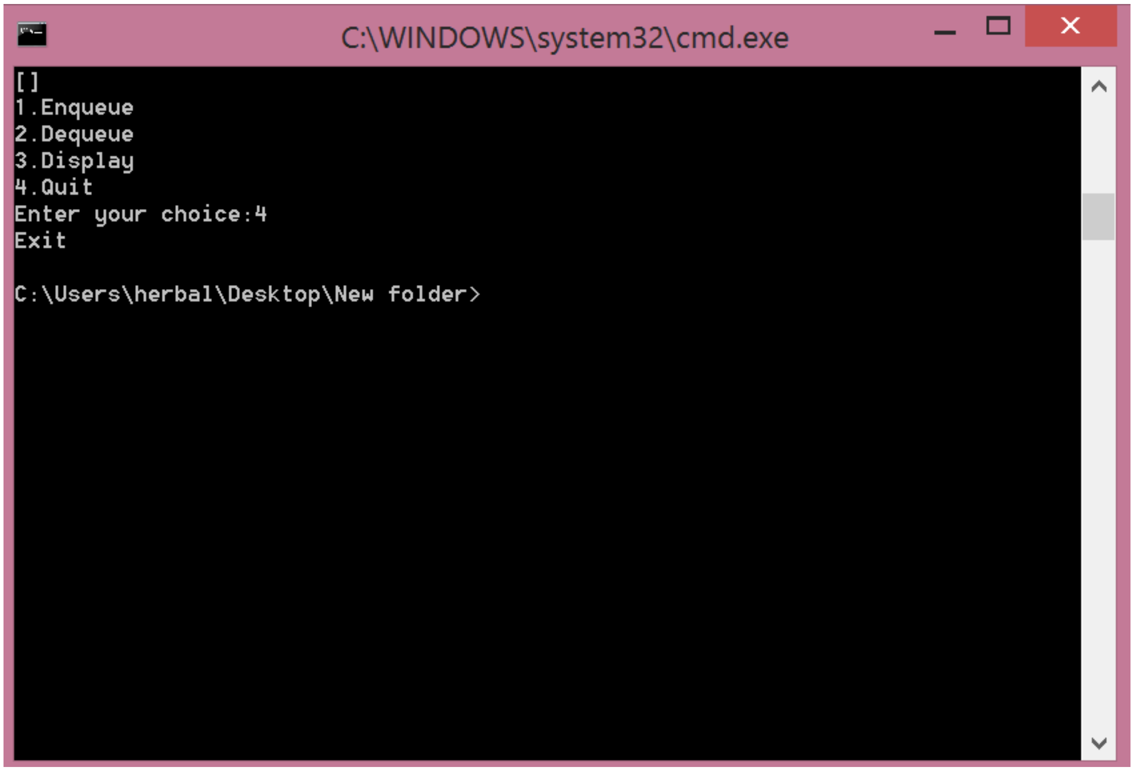
```
front=0
rear=0
mymax=eval(input("Enter maximum size of queue:"))
def createQueue():
    queue=[]
    return queue
def isEmpty(queue):
    return len(queue)==0
def enqueue(queue,item):
    queue.append(item)
    print("Enqueued to queue",item)
def dequeue(queue):
    if isEmpty(queue):
        return "Queue is empty"
    item=queue[0]
    del queue[0]
    return item
queue=createQueue()
while True:
    print("1.Enqueue")
    print("2.Dequeue")
    print("3.Display")
    print("4.Quit")
```

```
ch=int(input("Enter your choice:"))
if ch==1:
    if rear<mymax:
        item=input("Enter any elements:")
        enqueue(queue,item)
        rear+=1
    else:
        print("Queue is full")
elif ch==2:
    print(dequeue(queue))
elif ch==3:
    print(queue)
else:
    print("Exit")
    break
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe queue_FIFO.py
Enter maximum size of queue:2
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:1
Enter any elements:10
Enqueued to queue 10
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:1
Enter any elements:20
Enqueued to queue 20
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:1
Queue is full
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:3
['10', '20']
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:2
10
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:2
20
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:2
Queue is empty
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:3
```



```
[]
1.Enqueue
2.Dequeue
3.Display
4.Quit
Enter your choice:4
Exit

C:\Users\herbal\Desktop\New folder>
```

4.7 PRE LAB VIVA QUESTIONS:

1. Define a stack?
2. Stack data structure uses which principle?
3. Stack belongs to which type of data structure?
4. What do you mean by stack underflow?
5. What do you mean by stack overflow?
6. List out the basic operations of a stack?
7. How to implement stack?
8. Define a queue?
9. List out the basic operations of a queue?
10. Define a circular queue?
11. Which principle is followed in queue?
12. List out the applications of queue?

4.8 LAB ASSIGNMENT

1. Write a program to implement stack and its operations using arrays.
2. Formulate a program to reverse a list of numbers using stack.
3. Write a program to find the factorial of a number using stack.
4. Develop a program to check a given expression is balanced or not using stack
5. Compose a program to implement Queue operations using arrays.
6. Formulate a program to implement circular queue operations using arrays.
7. Write a program to implement a priority queue?

4.9 POST LAB VIVA QUESTIONS:

1. Write the time complexity of PUSH operation?
2. Write the time complexity of POP operation?
3. List out the applications of stack?

4. How to remove an element from stack?
5. How to insert an element into a stack?
6. Write the time complexity to insert an element into a queue?
7. Write the time complexity to delete an element from a queue?
8. List out the advantage of circular queue over linear queue?
9. Define a priority queue?
10. Define DEQUE?

WEEK-5

APPLICATIONS OF STACK

5.1 OBJECTIVE:

a. Write a Python program to convert infix expression into postfix expression using stack.
b. Write a Python program to evaluate the postfix expression using stack.

5.2 RESOURCES:

Python 3.4.0

5.3 PROGRAM LOGIC:

Procedure to convert Infix Expression into Postfix Expression

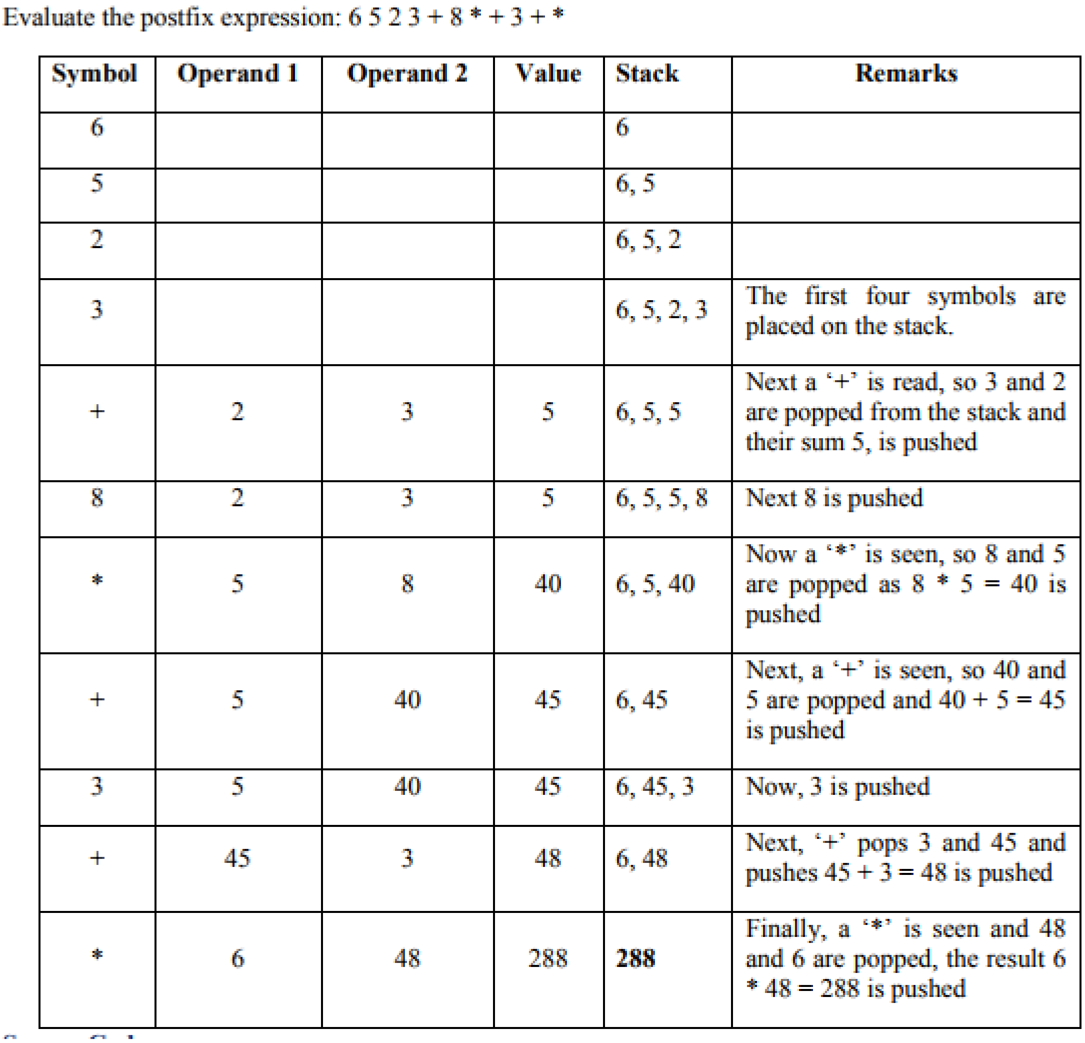
1. Read an infix expression and scan the symbols from left to right.
2. If the symbol is an operand, then write down in the postfix string.
3. If the symbol is a left parenthesis, then push it onto stack.
4. If the symbol is a right parenthesis, then pop the operators from until it find a left parenthesis or the stack is empty.
5. If the symbol is an operator, then check it's priority with the top most operator in the stack.
6. If the incoming operator is having high priority then the top most operator in the stack, then push the new operator onto stack, otherwise pop the existing operator and push the new operator.
7. Display the content of the postfix string.

Example: Convert the following expression A + B * C - D / E * H into its equivalent postfix expression.

Symbol	Postfix String	Stack	Remarks
A	A		Place A in the postfix string
+	A	+	Push + onto stack
B	A B	+	Place B in the postfix string
*	A B	+ *	Push * onto stack
C	A B C	+ *	Place C in the postfix string
-	A B C * +	-	Pop * and + from stack and push -.
D	A B C * + D	-	Place D in the postfix string
/	A B C * + D	- /	Push / onto stack
E	A B C * + D E	- /	Place E in the postfix string
*	A B C * + D E /	- *	Push * onto stack
H	A B C * + D E / H	- *	Place H in the postfix string
End of string	A B C * + D E / H * -	The input is now empty, pop the output symbols from the stack until it is empty.	

Procedure to evaluate a Postfix Expression

1. Read a postfix expression and scan the symbols from left to right.
2. If the symbol is an operand, then push it onto the stack.
3. If the symbol is an operator, the pop the top most two symbols and apply the operator.
4. Then push the result again in to stack.
5. Display the final result which is in stack.



Evaluate the postfix expression: 6 5 2 3 + 8 * + 3 + *

Symbol	Operand 1	Operand 2	Value	Stack	Remarks
6				6	
5				6, 5	
2				6, 5, 2	
3				6, 5, 2, 3	The first four symbols are placed on the stack.
+	2	3	5	6, 5, 5	Next a '+' is read, so 3 and 2 are popped from the stack and their sum 5, is pushed
8	2	3	5	6, 5, 5, 8	Next 8 is pushed
*	5	8	40	6, 5, 40	Now a '*' is seen, so 8 and 5 are popped as 8 * 5 = 40 is pushed
+	5	40	45	6, 45	Next, a '+' is seen, so 40 and 5 are popped and 40 + 5 = 45 is pushed
3	5	40	45	6, 45, 3	Now, 3 is pushed
+	45	3	48	6, 48	Next, '+' pops 3 and 45 and pushes 45 + 3 = 48 is pushed
*	6	48	288	**288**	Finally, a '*' is seen and 48 and 6 are popped, the result 6 * 48 = 288 is pushed

3.4 PROCEDURE:

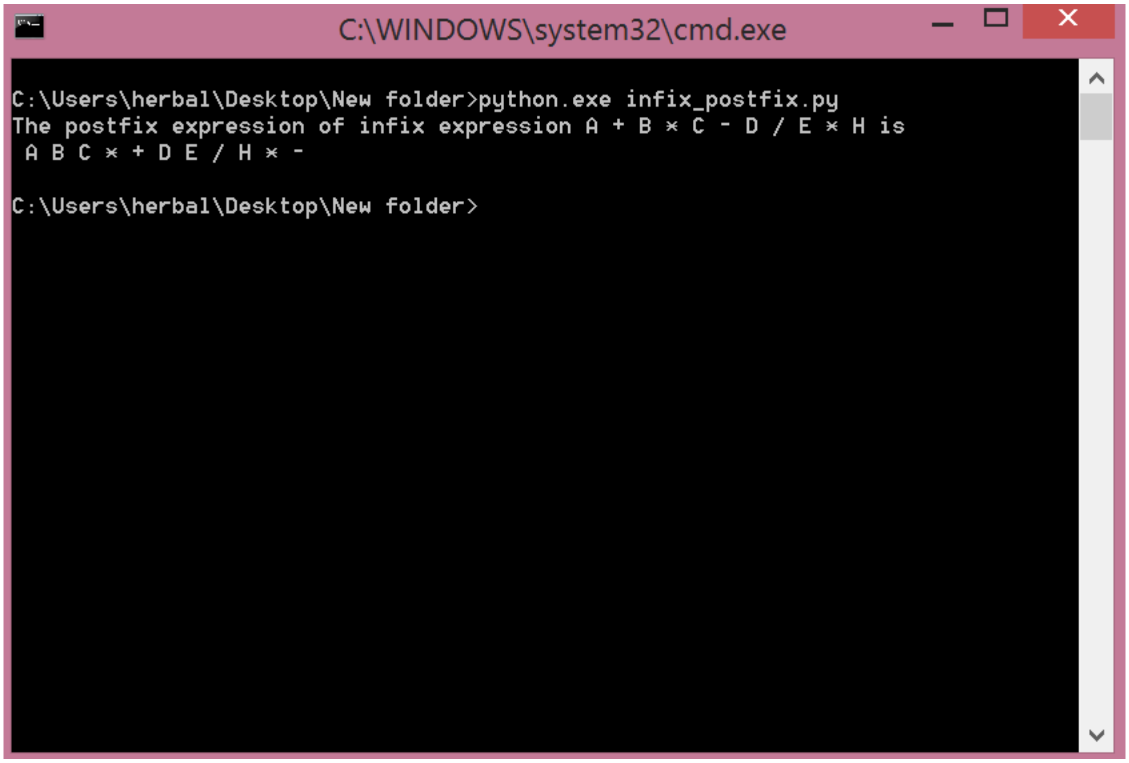
1. Create: open Python shell write a program after that save the program with .py extension.
2. Execute: F5

5.5 SOURCE CODE:

Program to convert Infix Expression Into Postfix Expression

```
class Stack:
    def __init__(self):
        self.items = []

    def isEmpty(self):
        return self.items == []

    def push(self, item):
        self.items.append(item)

    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items)-1]

    def size(self):
        return len(self.items)

def infix_postfix(infixexp):
    prec={}
    prec["^"]=4
    prec["*"]=3
    prec["/"]=3
    prec["+"]=2
    prec["-"]=2
    prec["("]=1
    opStack=Stack()
    postfixList=[]
    tokenList=infixexp.split()
    for token in tokenList:
        if token in "ABCDEFGHIJKLMNOPQRSTUVWXYZ" or \
           token in "abcdefghijklmnopqrstuvwxyz" or token in "0123456789":
            postfixList.append(token)
        elif token == '(':
            opStack.push(token)
        elif token == ')':
            topToken = opStack.pop()
            while topToken != '(':
                postfixList.append(topToken)
                topToken=opStack.pop()
        else:
            while (not opStack.isEmpty()) and \
                  (prec[opStack.peek()]>=prec[token]):
                postfixList.append(opStack.pop())
            opStack.push(token)
    while not opStack.isEmpty():
        postfixList.append(opStack.pop())
    return " ".join(postfixList)

a=infix_postfix('A + B * C - D / E * H')
print("The postfix expression of infix expression A + B * C - D / E * H is \n",a)
```

Output:

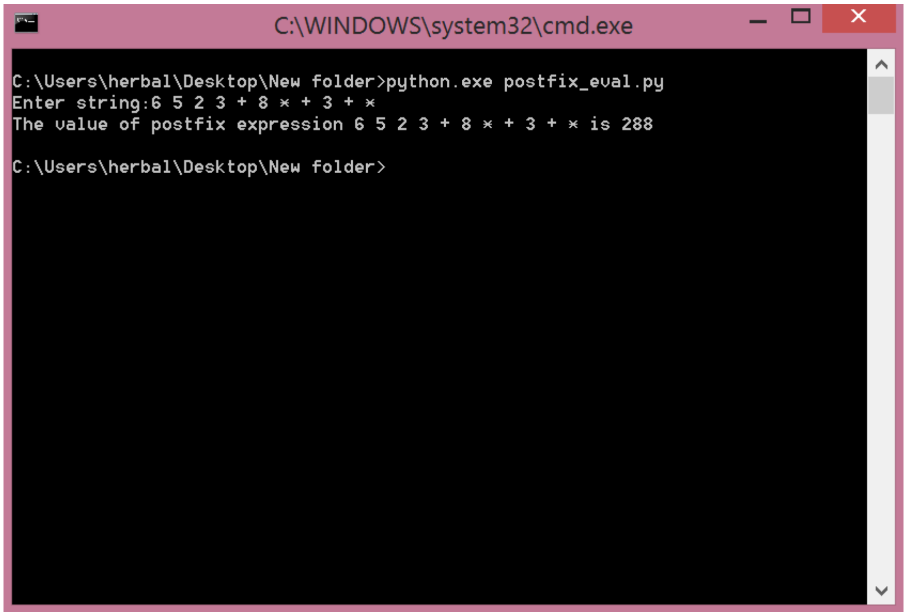


```
C:\Users\herbal\Desktop\New folder>python.exe infix_postfix.py
The postfix expression of infix expression A + B * C - D / E * H is
 A B C * + D E / H * -

C:\Users\herbal\Desktop\New folder>
```

Program for evaluating the postfix expression

```
class Stack:
    def __init__(self):
        self.items = []
    def isEmpty(self):
        return self.items == []
    def push(self, item):
        self.items.append(item)
    def pop(self):
        return self.items.pop()
    def peek(self):
        return self.items[len(self.items)-1]
    def size(self):
        return len(self.items)

def postfix_eval(s):
   s=s.split()
   n=len(s)
   stack =[]
   for i in range(n):
      if s[i].isdigit():
        #append function is equivalent to push
         stack.append(int(s[i]))
      elif s[i]=="+":
         a=stack.pop()
         b=stack.pop()
         stack.append(int(a)+int(b))
        elif s[i]=="*":
            a=stack.pop()
            b=stack.pop()
            stack.append(int(a)*int(b))
        elif s[i]=="/":
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)/int(a))
        elif s[i]=="-":
            a=stack.pop()
            b=stack.pop()
            stack.append(int(b)-int(a))
    return stack.pop()

def doMath(op,op1,op2):
    if op == "^":
        return op1 ^ op2
    elif op == "*":
        return op1 * op2
    elif op == "/":
        return op1 / op2
    elif op == "//":
        return op1 // op2
    elif op == "+":
        return op1 + op2
    else:
        return op1 - op2

s=input("enter string:")
val=postfix_eval(s)
print("The value of postfix expression",s,"is",val)
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe postfix_eval.py
Enter string:6 5 2 3 + 8 * + 3 + *
The value of postfix expression 6 5 2 3 + 8 * + 3 + * is 288

C:\Users\herbal\Desktop\New folder>
```

5.7 PRE-LAB VIVA QUESTIONS:

1. What is an expression?
2. Which operator is having highest priority?
3. Give an example for prefix expression?
4. Give an example for postfix expression?

5.8 LAB ASSIGNMENT:

1. Formulate a program to convert infix expression into postfix expression.
2. Write a program to evaluate any postfix expression.
3. Compose a program to convert infix expression into prefix expression.
4. Write a program to convert prefix expression into postfix expression.
5. Write a program to evaluate any prefix expression.

5.9 POST-LAB VIVA QUESTIONS:

1. What is the output of the following expression: 2 3 4 5 + * -
2. What is the advantage of postfix expression?
3. What is the maximum difference between number of operators and operands?
4. Which expression doesn't require parenthesis?
5. What is the output of the following expression: + * - 2 3 4 5

WEEK-6

IMPLEMENTATION OF SINGLE LINKED LIST

6.1 OBJECTIVE:

a. Write Python program to perform the following operations on single linked list.
(i) Creation (ii) insertion (iii) deletion (iv) traversal

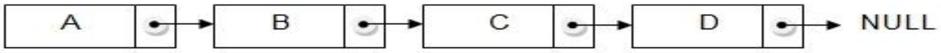
6.2 RESOURCES:

Python 3.4.0

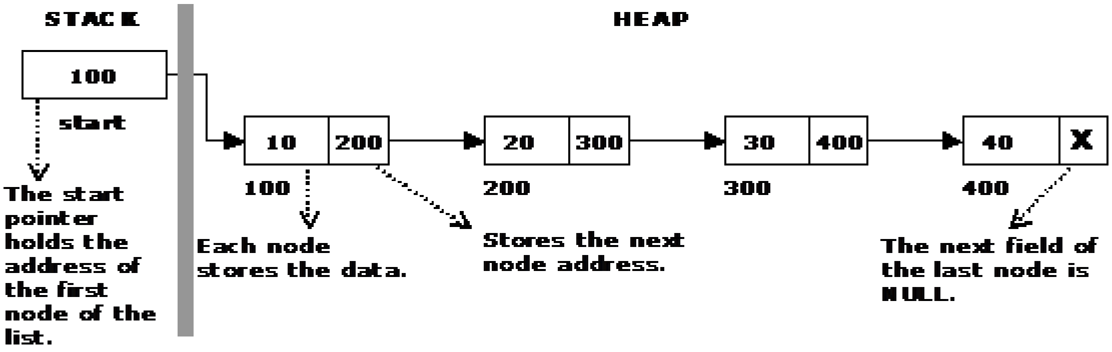
6.3 PROGRAM LOGIC:

Procedure for Single linked list

1. A singly linked list's node is divided into two parts. The first part holds or points to information about the node, and second part holds the address of next node. A singly linked list travels one way.
2. The beginning of the linked list is stored in a "**start**" pointer which points to the first node. The first node contains a pointer to the second node. The second node contains a pointer to the third node, ... and so on.
3. The last node in the list has its next field set to NULL to mark the end of the list.
4. The basic operations in a single linked list are: Creation, Insertion, Deletion, Traversing.



Single Linked List


STACK
HEAP
100
start
The start pointer holds the address of the first node of the list.
10
200
20
300
30
400
40
X
100
200
300
400
Each node stores the data.
Stores the next node address.
The next field of the last node is NULL.

6.4 PROCEDURE:

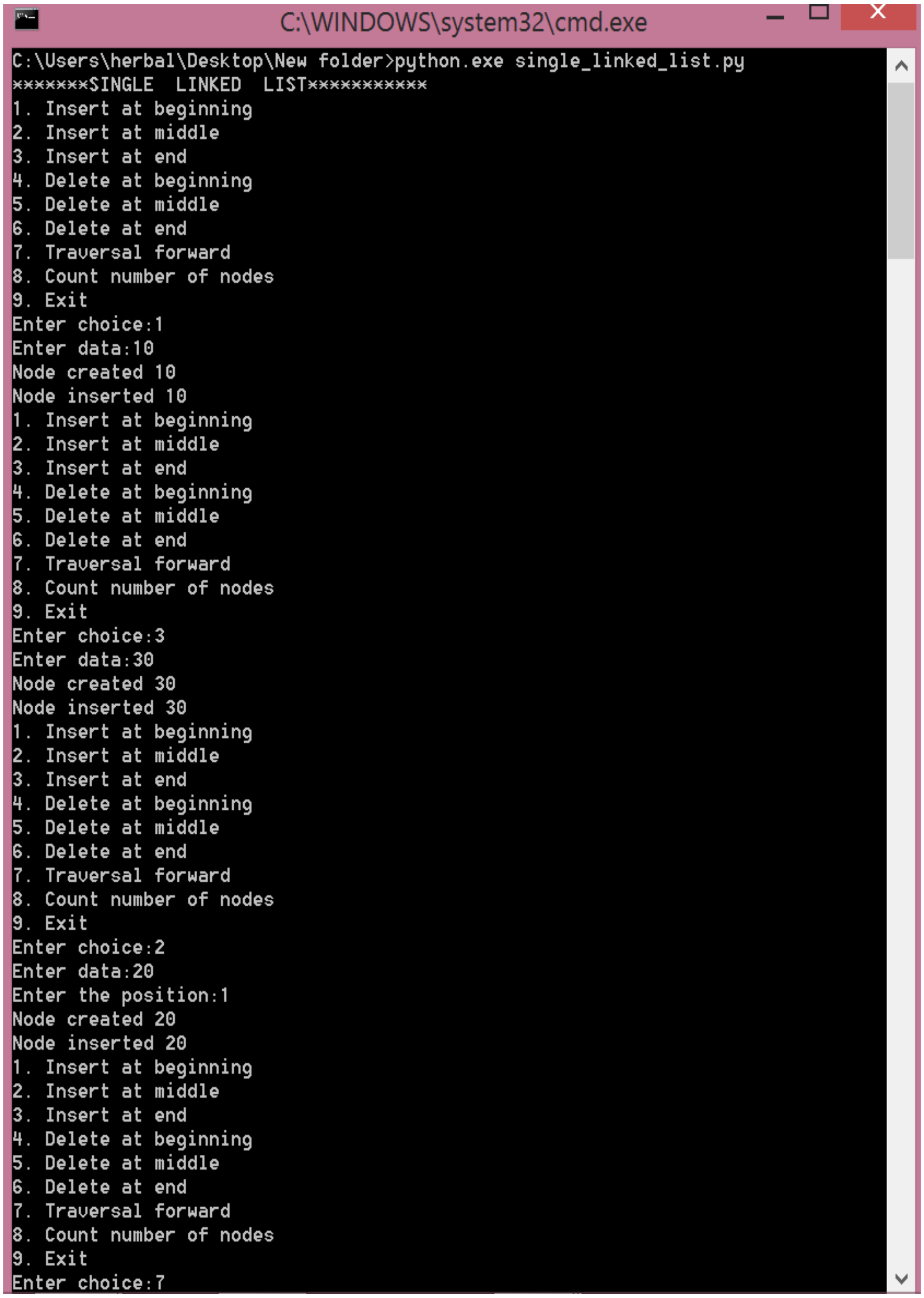
1. Create: open Python GUI, write a program after that save the program with .py extension.
2. Execute: F5

6.5 SOURCE CODE:

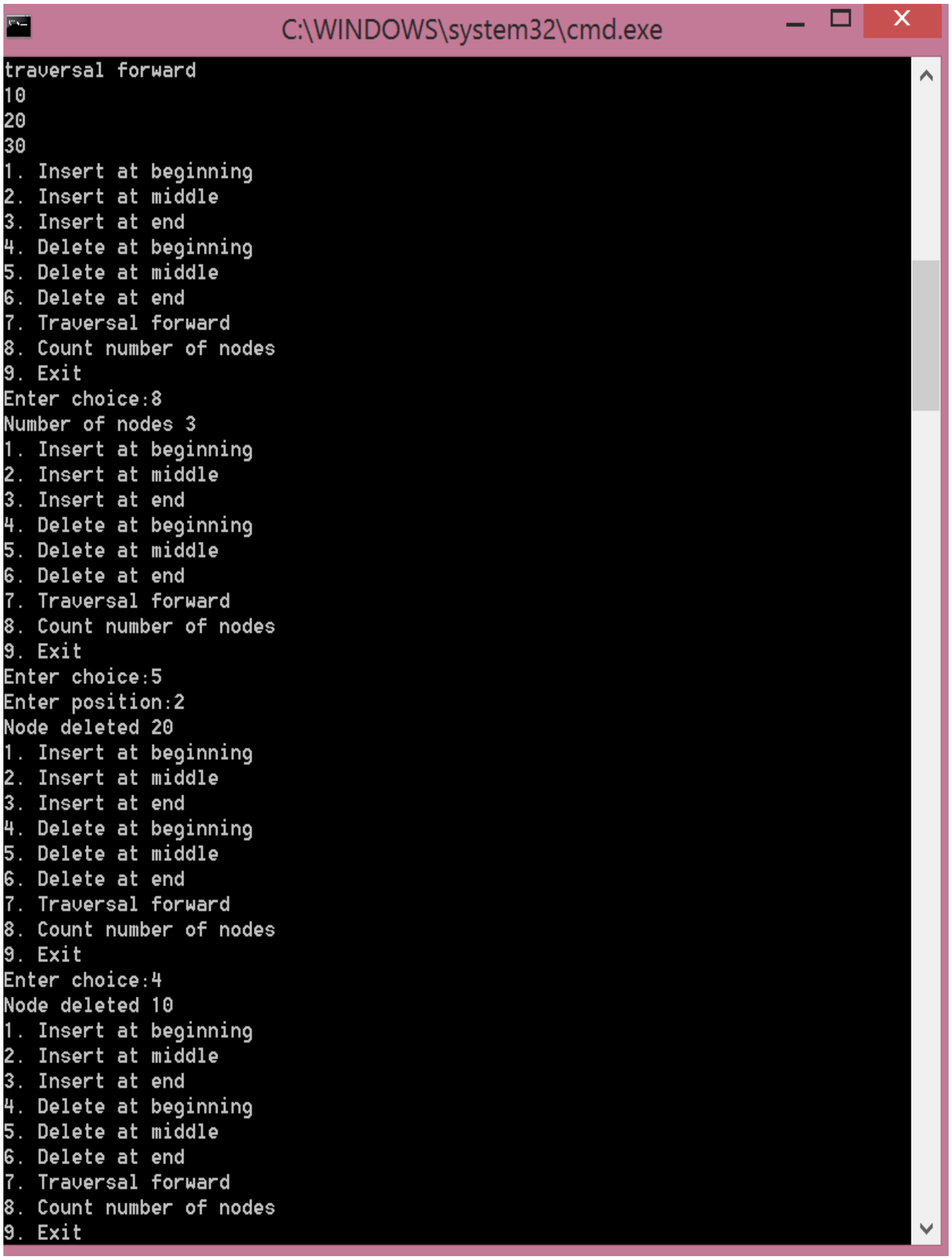
```
class Node:
    def __init__(self,data):
        print("Node created",data)
        self.data=data
        self.next=None
```

```
class S_L_List:
    def __init__(self):
        self.head=None
        self.ctr=0
    def insert_beginning(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
        else:
            node.next=self.head
            self.head=node
        self.ctr+=1
        print("Node inserted",data)
        return
    def insert_middle(self,pos,data):
        if pos==0:
            self.insert_beginning(data)
        elif pos==self.ctr+1:
            self.insert_end(data)
        else:
            node=Node(data)
            temp=self.head
            i=0
            while (i<pos-1):
                temp=temp.next
                i+=1
            node.next=temp.next
            temp.next=node
            self.ctr+=1
            print("Node inserted",data)
        return
    def insert_end(self,data):
        node=Node(data)
        node.next=None
        if self.head==None:
            self.head=node
            return
        temp=self.head
        while (temp.next is not None):
            temp=temp.next
        temp.next=node
        self.ctr+=1
        print("Node inserted",data)
        return
    def delete_beginning(self):
        if self.head==None:
            print("No nodes exist")
        elif self.ctr==1:
            print("Node deleted",self.head.data)
            self.head=None
            self.ctr-=1
        else:
            print("Node deleted",self.head.data)
            self.head=self.head.next
            self.ctr-=1
        return
    def delete_middle(self,pos):
        if self.head==None:
            print("No nodes exist")
        elif pos==0:
            self.delete_beginning()
        elif pos==self.ctr:
            self.delete_end()
        else:
            temp=self.head
            prev=temp
            i=0
            while (i<pos):
                prev=temp
                temp=temp.next
                i+=1
            prev.next=temp.next
            print("Node deleted",temp.data)
            temp.next=None
            self.ctr-=1
        return
    def delete_end(self):
        if self.ctr==0:
            print("No Nodes present")
        elif self.ctr==1:
            self.ctr=0
            print("Node deleted",self.head.data)
            self.head=None
        else:
            temp=self.head
            prev=self.head
            while (temp.next is not None):
                prev=temp
                temp=temp.next
            print("Node deleted",temp.data)
            prev.data=None
            self.ctr-=1
        return
    def traverse_forward(self):
        if self.head==None:
            print("No nodes exist")
        print("traversal forward")
        temp=self.head
        while (temp is not None):
            print(temp.data)
            temp=temp.next

def menu():
    print("1. Insert at beginning")
    print("2. Insert at middle")
    print("3. Insert at end")
    print("4. Delete at beginning")
    print("5. Delete at middle")
    print("6. Delete at end")
    print("7. Traversal forward")
    print("8. Count number of nodes")
    print("9. Exit")
    ch=eval(input("Enter choice:"))
    return ch

print("*******SINGLE  LINKED  LIST***********")
l=S_L_List()
while True:
    ch=menu()
    if ch==1:
        data=eval(input("Enter data:"))
        l.insert_beginning(data)
    elif ch==2:
        data=eval(input("Enter data:"))
        pos=eval(input("Enter the position:"))
        l.insert_middle(pos,data)
    elif ch==3:
        data=eval(input("Enter data:"))
        l.insert_end(data)
    elif ch==4:
        l.delete_beginning()
    elif ch==5:
        pos=eval(input("Enter position:"))
        l.delete_middle(pos)
    elif ch==6:
        l.delete_end()
    elif ch==7:
        l.traverse_forward()
    elif ch==8:
        print("Number of nodes",l.ctr)
    else:
        print("Exit")
        break
```

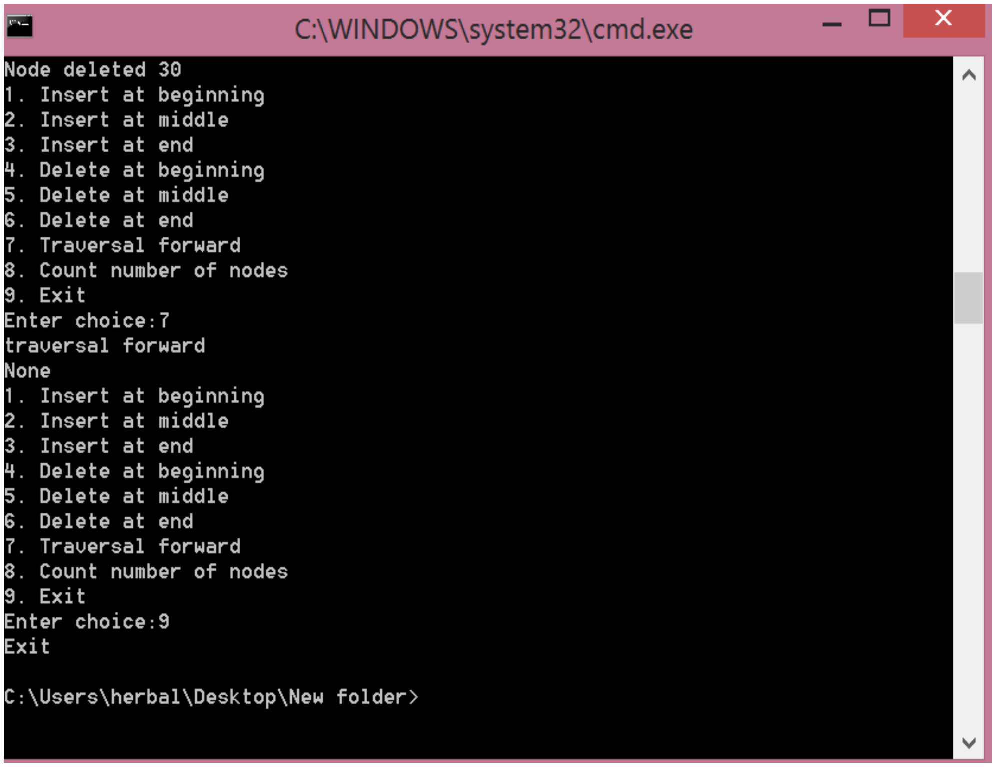
6.6 INPUT / OUTPUT:



```
C:\Users\herbal\Desktop\New folder>python.exe single_linked_list.py
*******SINGLE  LINKED  LIST***********
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:1
Enter data:10
Node created 10
Node inserted 10
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:3
Enter data:30
Node created 30
Node inserted 30
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:2
Enter data:20
Enter the position:1
Node created 20
Node inserted 20
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:7
```



```
traversal forward
10
20
30
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:8
Number of nodes 3
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:5
Enter position:2
Node deleted 20
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:4
Node deleted 10
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
```



```
Node deleted 30
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:7
traversal forward
None
1. Insert at beginning
2. Insert at middle
3. Insert at end
4. Delete at beginning
5. Delete at middle
6. Delete at end
7. Traversal forward
8. Count number of nodes
9. Exit
Enter choice:9
Exit

C:\Users\herbal\Desktop\New folder>
```

6.7 PRE-LAB VIVA QUESTIONS:

1. What is linked list?
2. What type of memory allocation is used in linked list?
3. How many self referential pointers are used in single linked list?
4. What is double linked list?
5. Which node contains NULL pointer in a single linked list?
6. How many nodes you can have in a single linked list?
7. What are the components of a polynomial expression?

6.7 LAB ASSIGNMENT:

1. Formulate a program to create a singly linked list and perform insertion, deletion and traversing operations on a singly linked list.
2. Write a program to merge two linked list?
3. Compose a program to print odd nodes of a linked list?
4. Write a program to divide the linked list into two parts into odd and even list?
5. Formulate a program to convert a single linked to circular linked list?
6. Compose a program to store and add two polynomial expressions in memory using linked list.

6.8 POST-LAB VIVAQUESTIONS:

1. What is the time complexity to insert a node at the beginning of linked list?
2. What is the time complexity to traverse a linked list?
3. How many modifications are required to delete a node at the beginning?
4. How many modifications are required to insert a node in the middle of the linked list?
5. What are the types of linked list?
6. What are the applications of a linked list?

WEEK-7

IMPLEMENTATION OF CIRCULAR SINGLE LINKED LIST

7.1 OBJECTIVE:

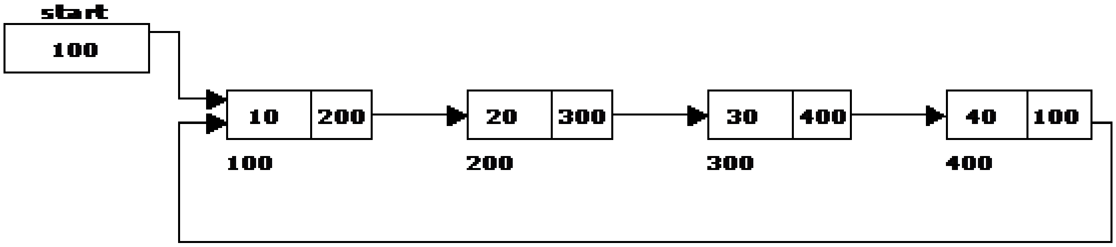
a. Write a python program to implement circular linked list
i)Creation (ii)insertion (iii)deletion (iv)traversal

7.2 RESOURCES:

Python 3.4.0

7.3 PROGRAM LOGIC:

1. In Circular single linked list the link field of the last node points back to the address of the first node.
2. A circular linked list has no beginning and no end. It is necessary to establish a special pointer called start pointer always pointing to the first node of the list.
3. The basic operations in a circular single linked list is: creation, insertion, deletion and traversing.



7.4 PROCEDURE:

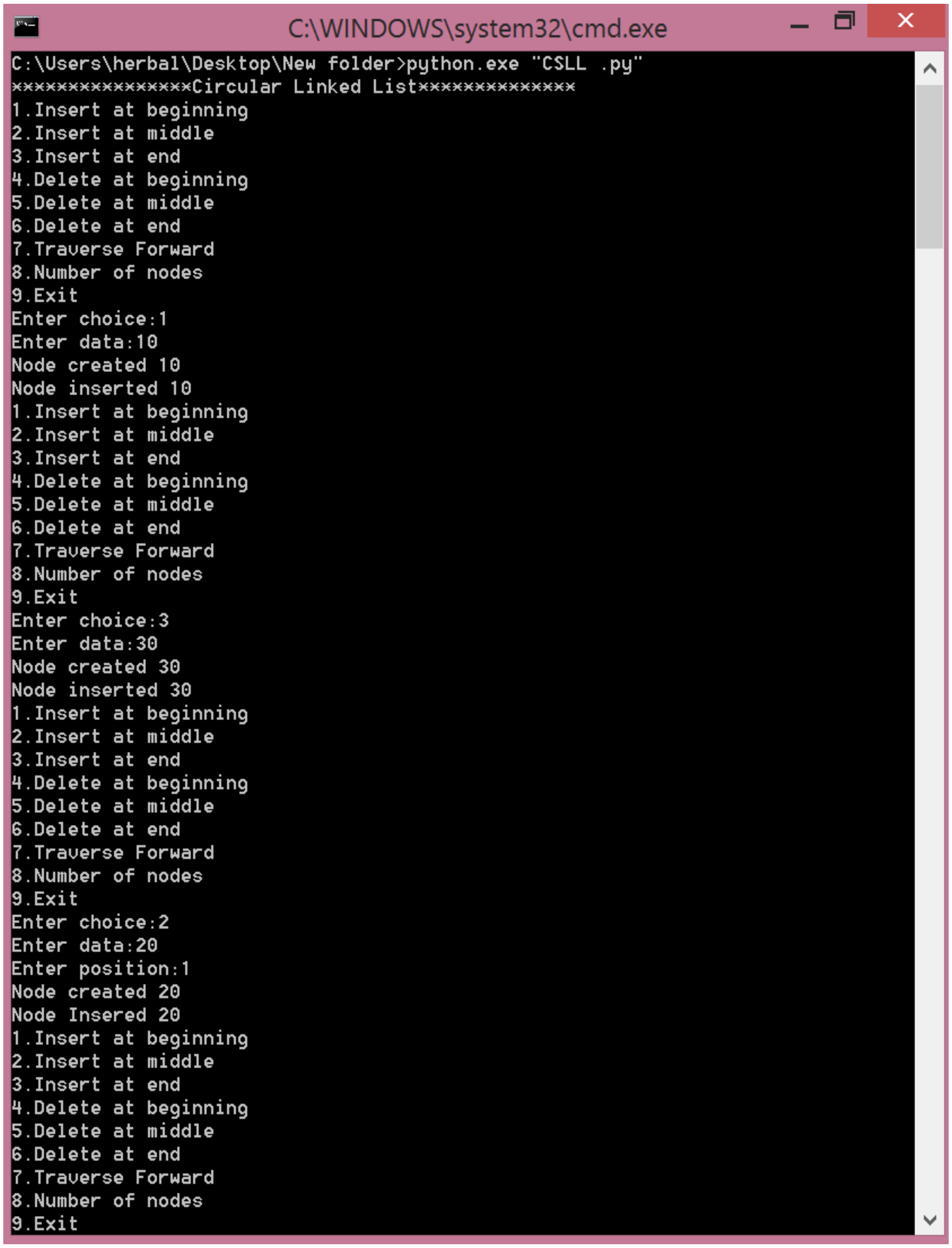
1. Create: open Python GUI, write a program after that save the program with .py extension.
2. Execute: F5

7.5 SOURCE CODE:

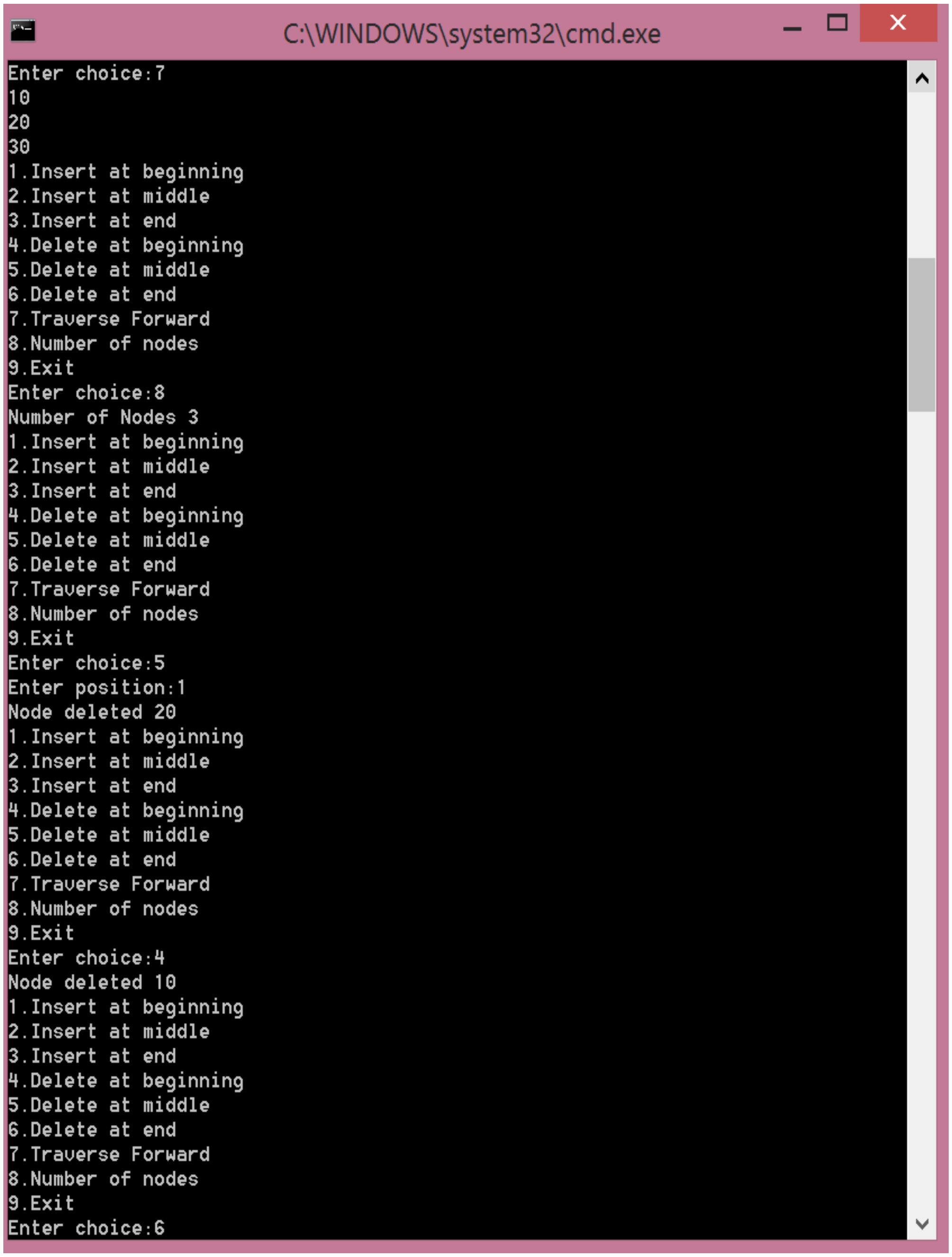
```
class Node:
   def __init__(self,data):
      self.next=None
      self.data=data
      print("Node created",data)

class CLList:
   def __init__(self):
      self.head=None
      self.ctr=0
   def insert_beg(self,data):
      node=Node(data)
      if self.head==None:
         self.head=node
         node.next=self.head
```

```
        else:
            temp=self.head
            while temp.next is not self.head:
                temp=temp.next
            temp.next=node
            node.next=self.head
            self.head=node
        print("Node inserted",data)
        self.ctr+=1
        return
    def insert_end(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
            node.next=self.head
        else:
            temp=self.head
            while temp.next is not self.head:
                temp=temp.next
            temp.next=node
            node.next=self.head
        self.ctr+=1
        print("Node inserted",data)
        return
    def insert_inter(self,pos,data):
        node=Node(data)
        if pos<1 or pos>self.ctr:
            print("invalid position")
        else:
            temp=self.head
            i=1
            while i<pos:
                temp=temp.next
                i+=1
            node.next=temp.next
            temp.next=node
            self.ctr+=1
            print("Node Insered",data)
        return
    def delete_beg(self):
        if self.head==None:
            print("No Nodes exist")
        elif self.ctr==1:
            print("Node deleted",self.head.data)
            self.head=None
            self.ctr-=1
        else:
            print("Node deleted",self.head.data)
            temp=self.head
            while temp.next is not self.head:
                temp=temp.next
            self.head=self.head.next
            temp.next=self.head
            self.ctr-=1
        return
    def delete_end(self):
        if self.head==None:
            print("No Nodes exist")
        elif self.ctr==1:
            print("Node deleted",self.head.data)
            self.head=None
            self.ctr-=1
        else:
            temp=self.head
            prev=temp
            while temp.next is not self.head:
                prev=temp
                temp=temp.next
            print("Node deleted",temp.data)
            prev.next=temp.next
            self.ctr-=1
        return
    def delete_inter(self,pos):
        if self.head==None:
            print("No nodes exist")
        elif pos<1 or pos>self.ctr:
            print("Invalid position")
        elif self.ctr==1:
            print("Node deleted",self.head.data)
            self.head=None
            self.ctr-=1
        else:
            temp=self.head
            prev=temp
            i=0
            while i<pos:
                prev=temp
                temp=temp.next
                i+=1
            prev.next=temp.next
            print("Node deleted",temp.data)
            self.ctr-=1
        return
    def traverse(self):
        temp=self.head
        i=0
        while i<self.ctr:
            print(temp.data)
            temp=temp.next
            i+=1
        return

def Menu():
    print("1.Insert at beginning")
    print("2.Insert at middle")
    print("3.Insert at end")
    print("4.Delete at beginning")
    print("5.Delete at middle")
    print("6.Delete at end")
    print("7.Traverse Forward")
    print("8.Number of nodes")
    print("9.Exit")
    ch=int(input("Enter choice:"))
    return ch

c=CLList()
print("****************Circular Linked List**************")
while True:
    ch=Menu()
    if ch==1:
        data=input("Enter data:")
        c.insert_beg(data)
    elif ch==2:
        data=input("Enter data:")
        pos=int(input("Enter position:"))
        c.insert_inter(pos,data)
    elif ch==3:
        data=input("Enter data:")
        c.insert_end(data)
    elif ch==4:
        c.delete_beg()
    elif ch==5:
        pos=int(input("Enter position:"))
        c.delete_inter(pos)
    elif ch==6:
        c.delete_end()
    elif ch==7:
        c.traverse()
    elif ch==8:
        print("Number of Nodes",c.ctr)
    else:
        print("Exit")
        break
```

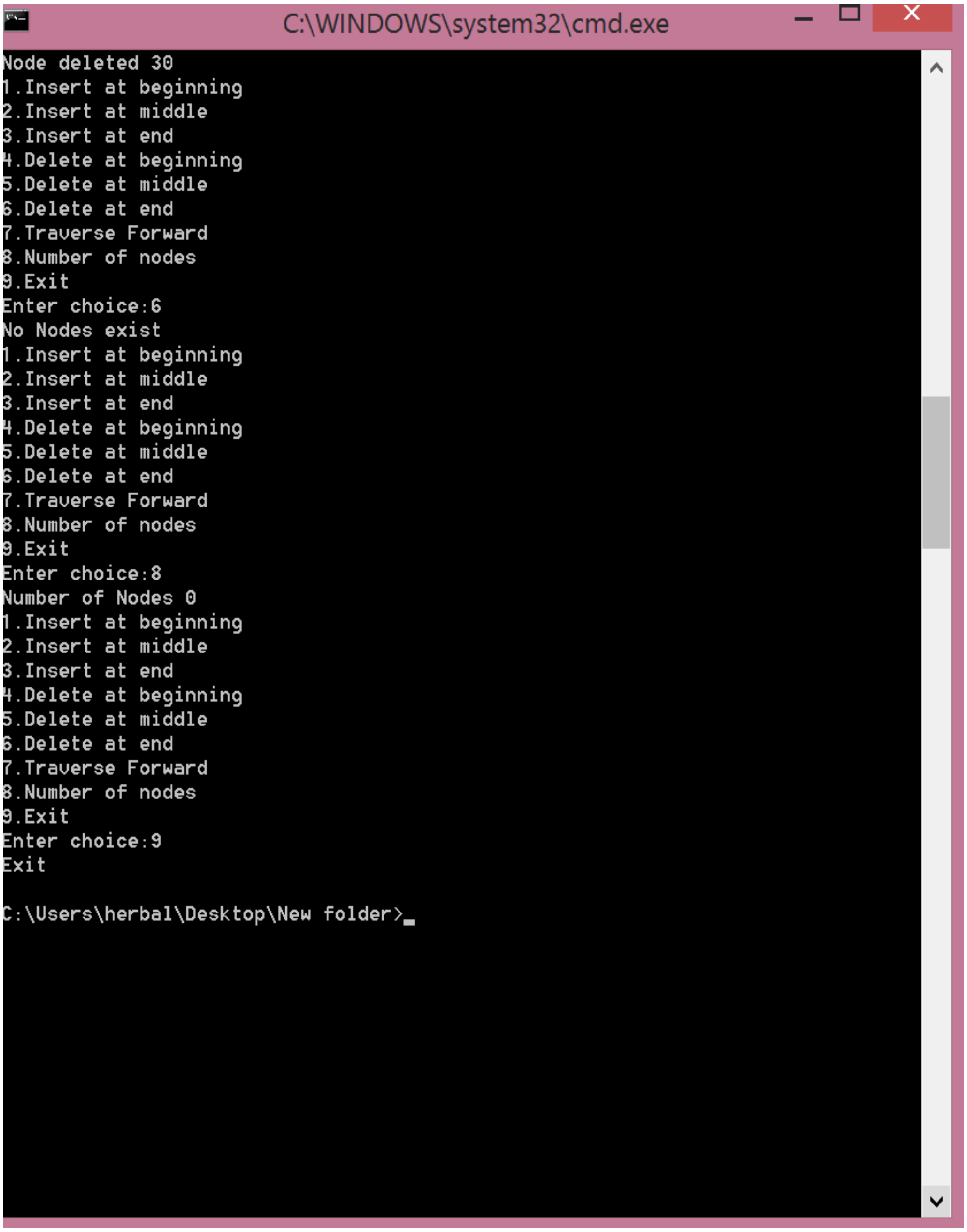
Output:



```
C:\Users\herbal\Desktop\New folder>python.exe "CSLL .py"
****************Circular Linked List**************
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:1
Enter data:10
Node created 10
Node inserted 10
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:3
Enter data:30
Node created 30
Node inserted 30
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:2
Enter data:20
Enter position:1
Node created 20
Node Insered 20
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
```



```
Enter choice:7
10
20
30
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:8
Number of Nodes 3
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:5
Enter position:1
Node deleted 20
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:4
Node deleted 10
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:6
```



```
Node deleted 30
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:6
No Nodes exist
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:8
Number of Nodes 0
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:9
Exit

C:\Users\herbal\Desktop\New folder>_
```

7.6 PRE-LAB VIVA QUESTIONS:

1 What is circular linked list?
2 What type of memory allocation is used in linked circular list?
3 How many self referential pointers are used in circular single linked list?
4 What is double linked list?
5 Which node contains NULL pointer in a circular single linked list?
6 How many nodes you can have in a circular single linked list?

7.7 LAB ASSIGNMENT:

1. Formulate a program to create a circular singly linked list and perform insertion, deletion and traversing operations on a singly linked list.
2. Write a program to merge two linked list?
3. Compose a program to print odd nodes of a circular linked list?
4. Write a program to divide the circular linked list into two parts into odd and even list?
5. Formulate a program to convert a single linked to circular linked list?

7.8 POST-LAB VIVA QUESTIONS:

1. What is the time complexity to insert a node at the beginning of circular linked list?
2. What is the time complexity to traverse a circular linked list?
3. How many modifications are required to delete a node at the beginning?
4. How many modifications are required to insert a node in the middle of the circular linked list?
5. What are the types of linked list?
6. What are the applications of a circular linked list?

WEEK-8

IMPLEMENTATION OF DOUBLE LINKED LIST

8.1 OBJECTIVE:

a. Create a doubly linked list of integers.
b. Delete a given integer from the above doubly linked list.
c. Display the contents of the above list after deletion

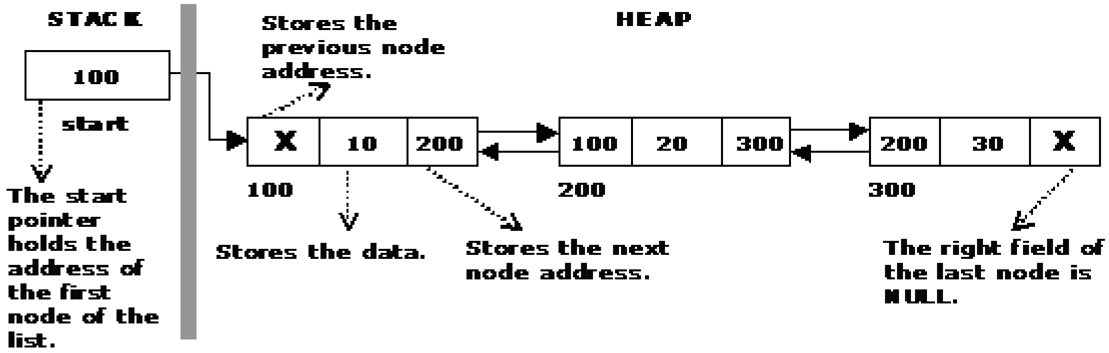
8.2 RESOURCES:

Python 3.4.0

8.3 PROGRAM LOGIC:

1. In a doubly-linked list each node of the list contain two references (or links) – one to the previous node and other to the next node. The previous link of the first node and the next link of the last node points to NULL.
2. A double linked list is a two-way list in which all nodes will have two links. This helps in accessing both successor node and predecessor node from the given node position. It provides bi-directional traversing.
3. Each node contains three fields: Left link, Data and Right link.
4. The left link points to the predecessor node and the right link points to the successor node. The data field stores the required data.

5. The basic operations in a double linked list are: creation, insertion, deletion and traversing.
6. The beginning of the double linked list is stored in a "start" pointer which points to the first node. The first node's left link and last node's right link is set to NULL.


STACK
100
start
The start pointer holds the address of the first node of the list.
Stores the previous node address.
HEAP
X
10
200
100
20
300
200
30
X
100
200
300
Stores the data.
Stores the next node address.
The right field of the last node is NULL.

8.4 PROCEDURE:

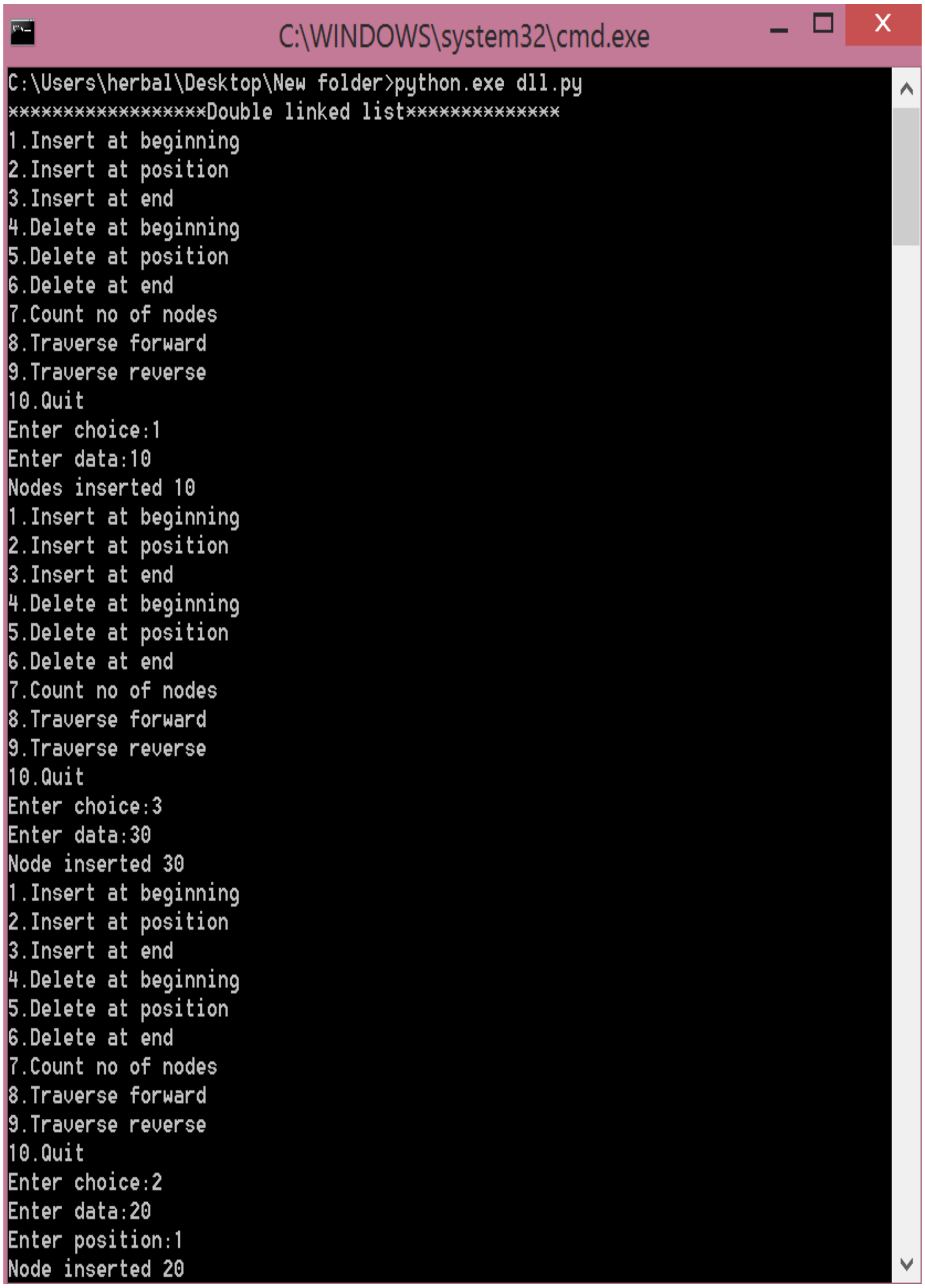
1. Create: open Python GUI,write a program after that save the program with .py extension.
2. Execute: F5

8.5 SOURCE CODE:

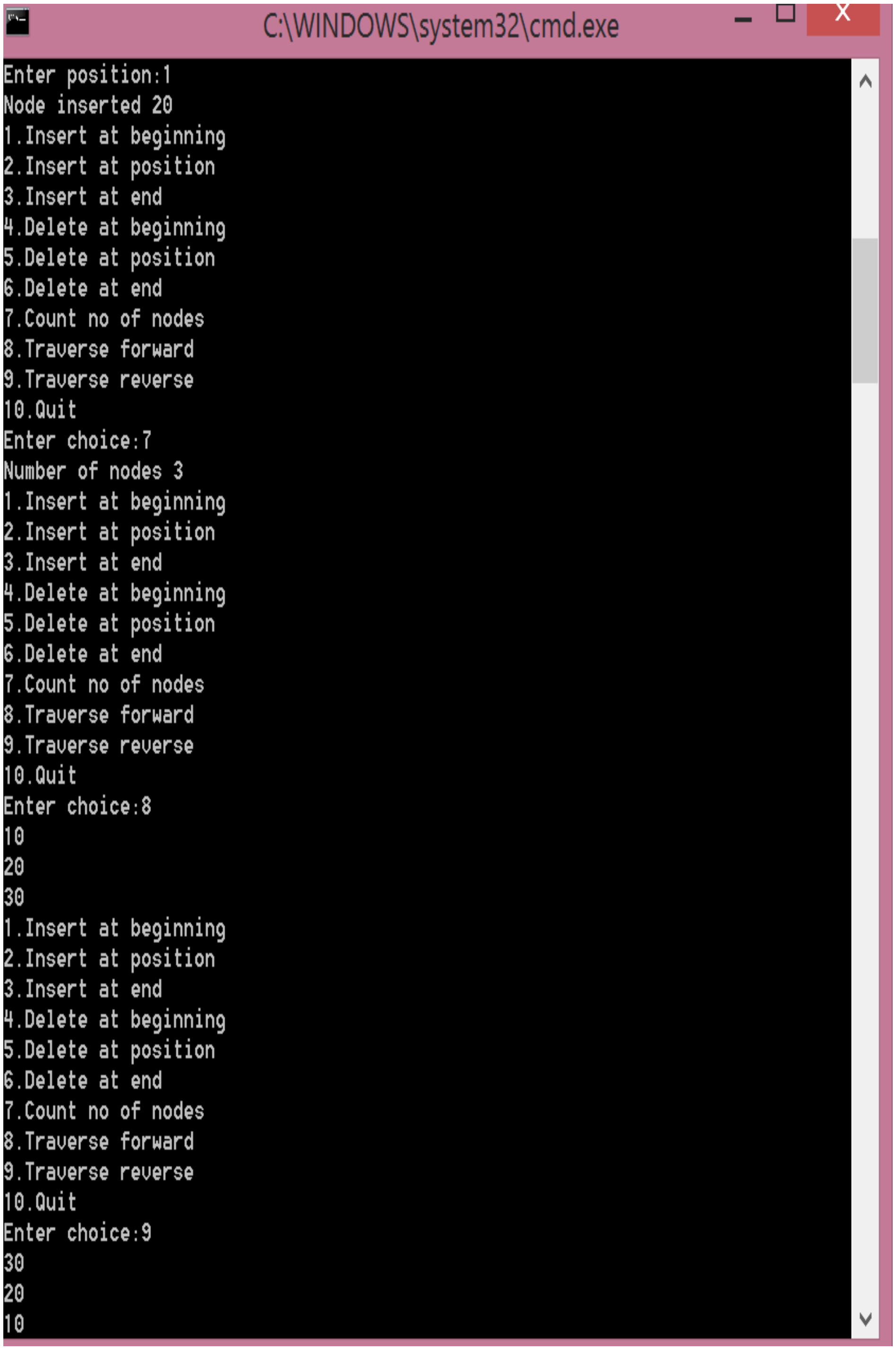
```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=self.prev=None
```

```
class DLinkedList:
    def __init__(self):
        self.head=None
        self.ctr=0
    def insert_beg(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
        else:
            node.next=self.head
            self.head.prev=node
            self.head=node
        self.ctr +=1
        print("Nodes inserted",data)
        return
    def insert_end(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
        else:
            temp=self.head
            while(temp.next is not None):
                temp=temp.next
            temp.next=node
            node.prev=temp
        self.ctr +=1
        print("Node inserted",data)
        return
    def delete_beg(self):
        if self.head==None:
            print("No node exist")
        else:
            print("Node deleted",self.head.data)
            self.head=self.head.next
            self.head.prev=None
            self.ctr -=1
        return
    def delete_end(self):
        if self.head==None:
            print("No nodes exist")
        elif self.ctr==1:
            self.ctr=0
            print ("Node deleted",self.head.data)
            self.head=None
        else:
            temp=self.head
            while temp.next is not None:
                temp=temp.next
            print("Node deleted",temp.data)
            temp=temp.prev
            temp.next=None
            self.ctr -=1
        return
def insert_pos(self,pos,data):
    if pos==0:
        self.insert_beg(data)
    elif pos==self.ctr:
        self.insert_end(data)
    else:
        node=Node(data)
        temp=self.head
        i=1
        while i<pos-1:
            temp=temp.next
            i +=1
        node.next=temp.next
        temp.next.prev=node
        temp.next=node
        node.prev=temp
        self.ctr +=1
        print("Node inserted",data)
    return
def delete_pos(self,pos):
    if self.head==None:
        print("Node is empty")
    else:
        if pos==0:
            self.delete_beg()
        elif pos==self.ctr:
            self.delete_end()
        else:
            temp=self.head
            i=0
            while i<pos:
                temp=temp.next
                i+=1
            print("node deleted",temp.data)
            temp.prev.next=temp.next
            temp.next.prev=temp.prev
            temp.next=None
            temp.preve=None
            self.ctr -=1
        return
def traverse_f(self):
    if self.head==None:
        print("No nodes exist")
    temp=self.head
    i=0
    while i<self.ctr:
        print(temp.data)
        temp=temp.next
        i+=1
    return
def traverse_r(self):
        if self.head==None:
            print("No nodes exist")
        temp=self.head
        while temp.next is not None:
            temp=temp.next
        while temp is not None:
            print(temp.data)
            temp=temp.prev
def menu():
    print("1.Insert at beginning")
    print("2.Insert at position")
    print("3.Insert at end")
    print("4.Delete at beginning")
    print("5.Delete at position")
    print("6.Delete at end")
    print("7.Count no of nodes")
    print("8.Traverse forward")
    print("9.Traverse reverse")
    print("10.Quit")
    ch=eval(input("Enter choice:"))
    return ch

print("******************Double linked list**************")
d=DLinkedList()
while True :
    ch=menu()
    if ch==1:
        data=eval(input("Enter data:"))
        d.insert_beg(data)
    elif ch==2:
        data=eval(input("Enter data:"))
        pos=int(input("Enter position:"))
        d.insert_pos(pos,data)
    elif ch==3:
        data=eval(input("Enter data:"))
        d.insert_end(data)
    elif ch==4:
        d.delete_beg()
    elif ch==5:
        pos=int(input("Enter position:"))
        d.delete_pos(pos)
    elif ch==6:
        d.delete_end()
    elif ch==7:
        print("Number of nodes",d.ctr)
    elif ch==8:
        d.traverse_f()
    elif ch==9:
        d.traverse_r()
    else:
        print("Exit")
        break
```

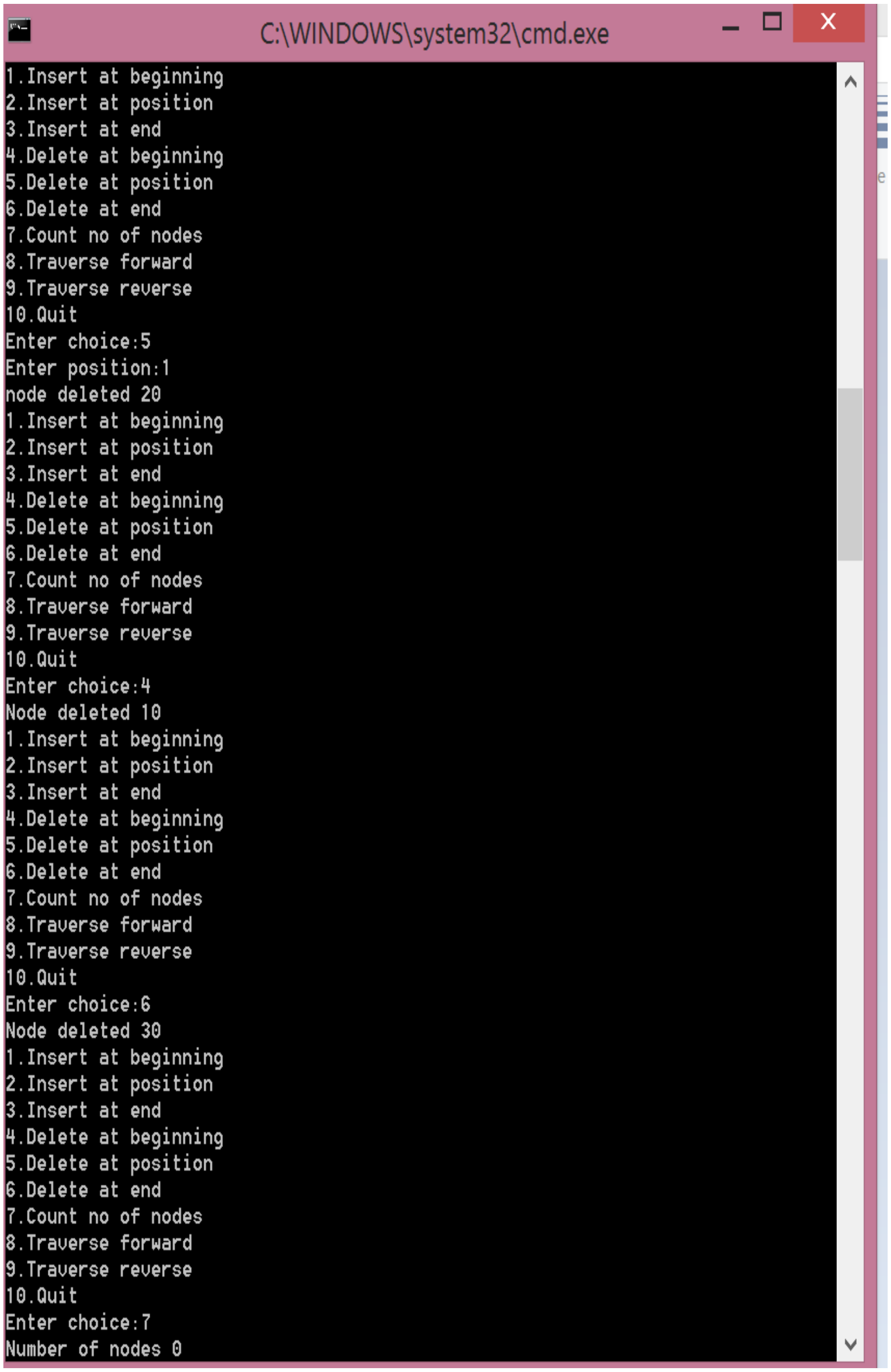
Output:



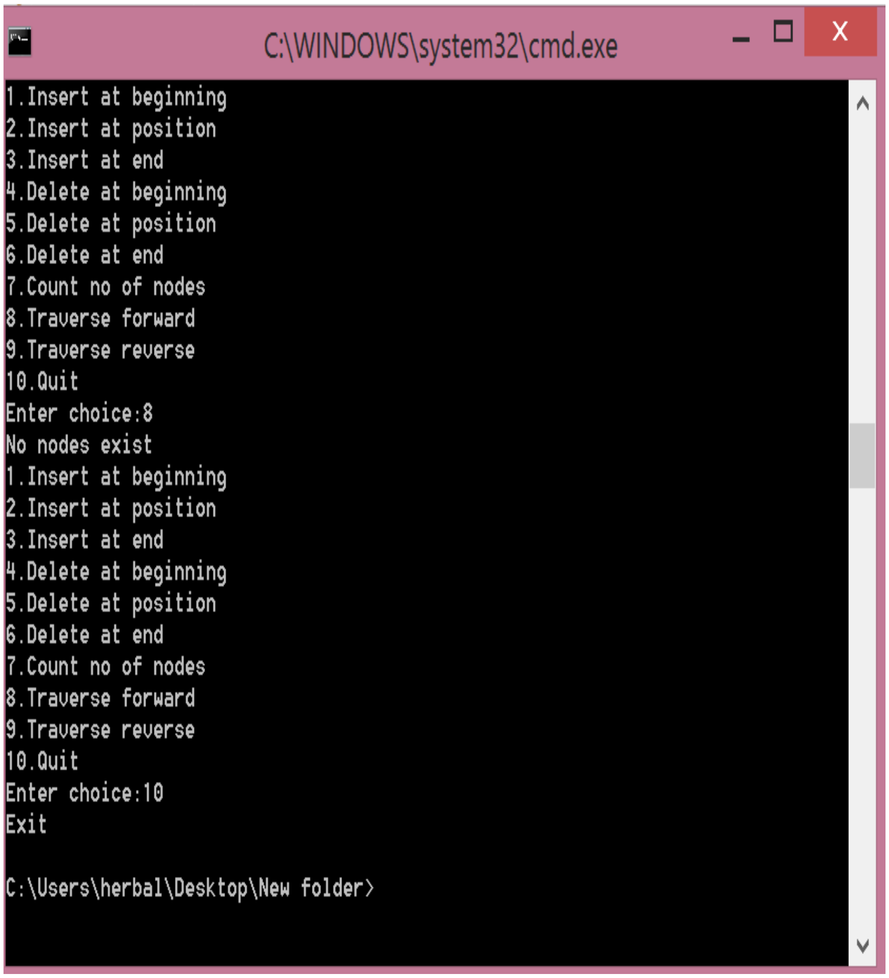
```
C:\Users\herbal\Desktop\New folder>python.exe dll.py
******************Double linked list**************
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:1
Enter data:10
Nodes inserted 10
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:3
Enter data:30
Node inserted 30
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:2
Enter data:20
Enter position:1
Node inserted 20
```



```
Enter position:1
Node inserted 20
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:7
Number of nodes 3
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:8
10
20
30
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:9
30
20
10
```



```
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:5
Enter position:1
node deleted 20
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:4
Node deleted 10
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:6
Node deleted 30
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:7
Number of nodes 0
```



```
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:8
No nodes exist
1.Insert at beginning
2.Insert at position
3.Insert at end
4.Delete at beginning
5.Delete at position
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:10
Exit

C:\Users\herbal\Desktop\New folder>
```

8.6 PRE-LAB VIVA QUESTIONS:

1. What is double linked list
2. How to represent a node in double linked list
3. Differentiate between single and double linked list

8.7 LAB ASSIGNMENT:

1. Write a program to insert a node at first , last and at specified position of double linked list?
2. Write a program to eliminate duplicates from double linked list?
3. Write a program to delete a node from first, last and at specified position of double linked list?

8.8 POST-LAB VIVA QUESTIONS:

1. How to represent double linked list?
2. How will you traverse double linked list?
3. List the advantages of double linked list over single list?

WEEK-9

IMPLEMENTATION OF STACK USING LINKED LIST

9.1 OBJECTIVE:

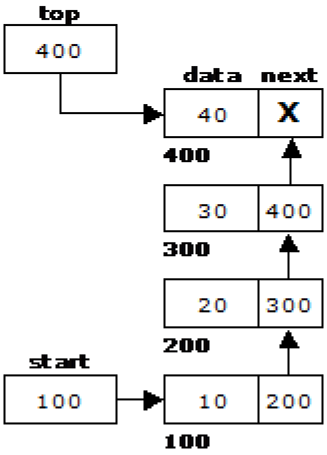
Write a Python script to implement stack using linked list.

9.2 RESOURCES:

Python 3.4.0

9.3 PROGRAM LOGIC:

1. STACK: Stack is a linear data structure which works under the principle of last in first out. Basic operations: push, pop, display.
2. PUSH: if (newnode==NULL), display Stack overflow. if(start == NULL) then start = newnode. Otherwise use loop and copy address of new node in to old node by creating link.
3. Pop: if (top == NULL), display Stack underflow. Otherwise printing the element at the top of the stack and decrementing the top value by doing the top.
4. DISPLAY: if (top == NULL), display Stack is empty. Otherwise printing the elements in the stack from top.



9.4 PROCEDURE:

1. Create: open Python GUI write a program after that save the program with .py extension.
2. Execute: F5

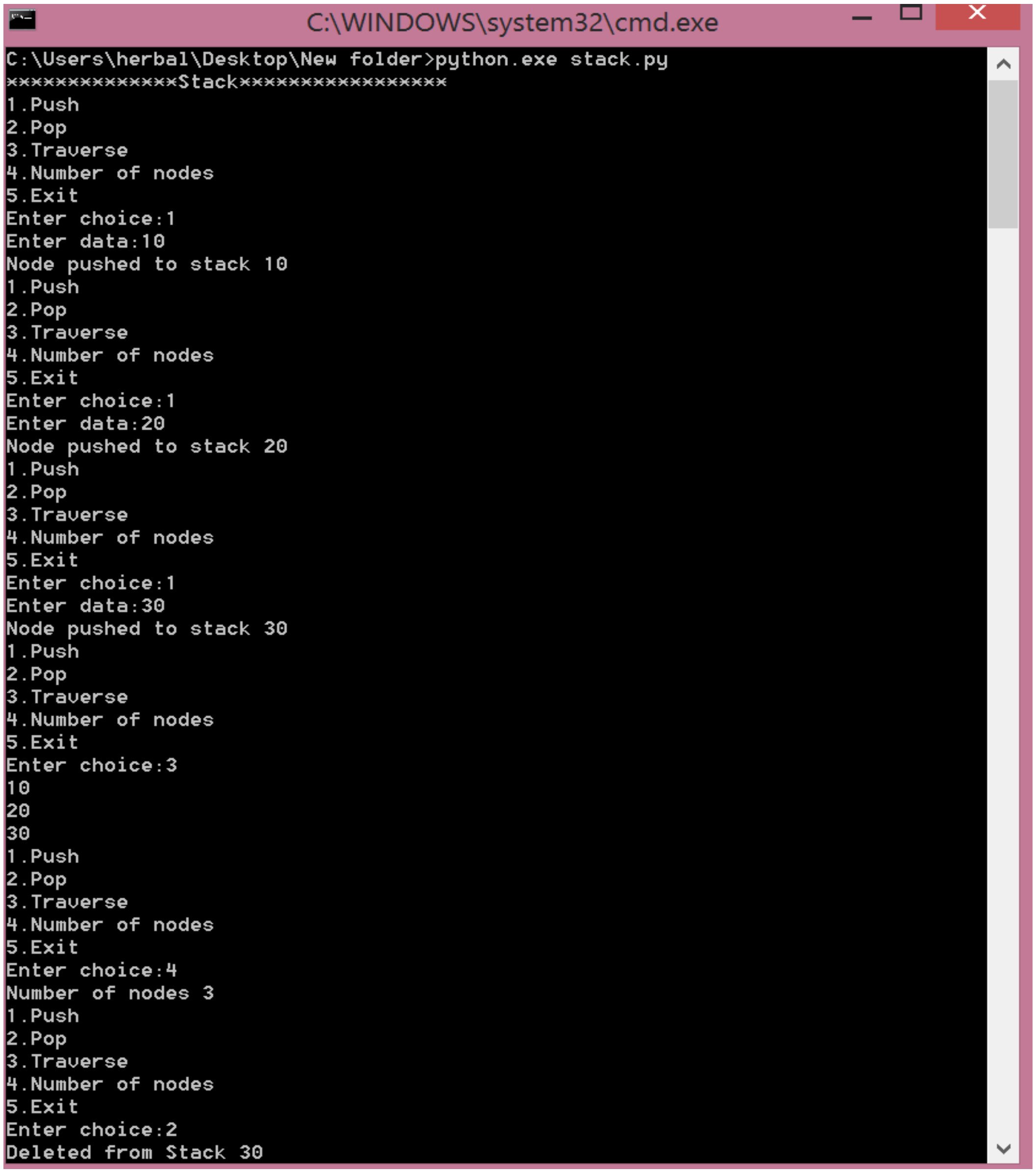
9.5 SOURCE CODE:

```
class Node:
   def __init__(self,data):
      self.data=data
      self.next=None

class Stack:
   def __init__(self):
      self.head=None
      self.ctr=0
```

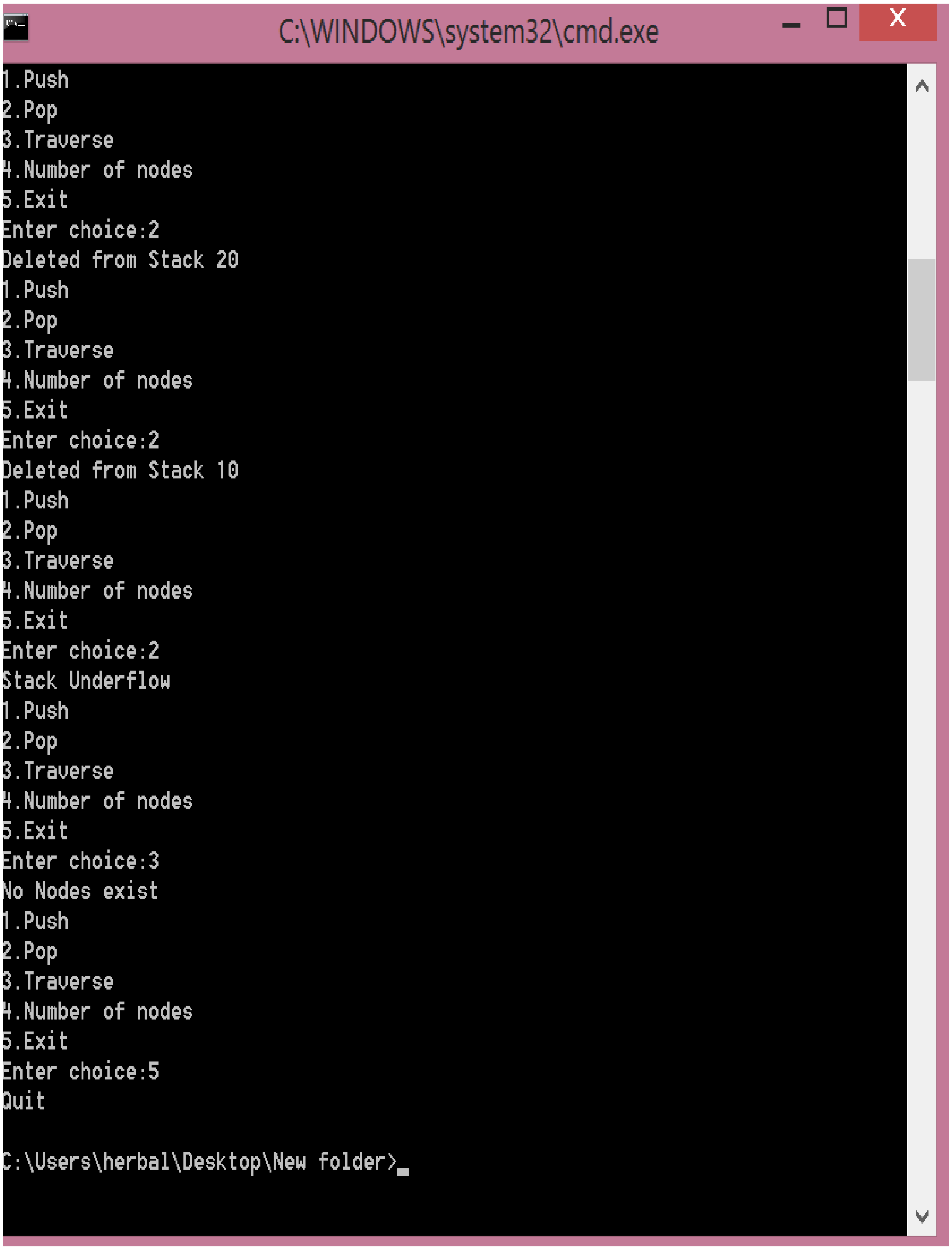
```
        self.top=None
    def Push(self,data):
        node=Node(data)
        if self.head==None:
            self.head=node
            self.top=node
        else:
            self.top.next=node
            self.top=node
        print("Node pushed to stack",data)
        self.ctr+=1
        return
    def Pop(self):
        if self.head==None:
            print("Stack Underflow")
        elif self.head==self.top:
            print("Deleted from Stack",self.head.data)
            self.head=self.top=None
            self.ctr-=1
        else:
            print("Deleted from Stack",self.top.data)
            temp=self.head
            while temp.next is not self.top:
                temp=temp.next
            temp.next=None
            self.top=temp
            self.ctr-=1
            return
    def Traverse(self):
        if self.head==None:
            print("No Nodes exist")
            return
       temp=self.head
      while temp is not None:
          print(temp.data)
          temp=temp.next

def Menu():
    print("1.Push\n2.Pop\n3.Traverse\n4.Number of nodes\n5.Exit")
    ch=int(input("Enter choice:"))
    return ch

s=Stack()
print("**************Stack*****************")
while True:
```

```
ch=Menu()
if ch==1:
    data=input("Enter data:")
    s.Push(data)
elif ch==2:
    s.Pop()
elif ch==3:
    s.Traverse()
elif ch==4:
    print("Number of nodes",s.ctr)
else:
    print('Quit')
    break
```

Output:



```
C:\Users\herbal\Desktop\New folder>python.exe stack.py
**************Stack*****************
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:10
Node pushed to stack 10
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:20
Node pushed to stack 20
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:30
Node pushed to stack 30
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:3
10
20
30
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:4
Number of nodes 3
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Deleted from Stack 30
```



```
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Deleted from Stack 20
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Deleted from Stack 10
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Stack Underflow
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:3
No Nodes exist
1.Push
2.Pop
3.Traverse
4.Number of nodes
5.Exit
Enter choice:5
Quit

C:\Users\herbal\Desktop\New folder>_
```

9.7 PRE-LAB VIVA QUESTIONS:

1. What do you mean by stack overflow?
2. What are the basic operations of a stack?
3. How to implement stack?

9.8 LAB ASSIGNMENT:

1. Formulate a program to reverse a list of numbers using stack.
2. Write a program to find the factorial of a number using stack.
3. Develop a program to check a given expression is balanced or not using stack

9.9 POST-LAB VIVA QUESTIONS:

1. How to remove an element from stack?
2. How to insert an element using a stack?
3. Is it possible to store any number of data elements in stack?
4. What are the demerits of stack?

WEEK-10

IMPLEMENTATION OF QUEUE USING LINKED LIST

10.1 OBJECTIVE:

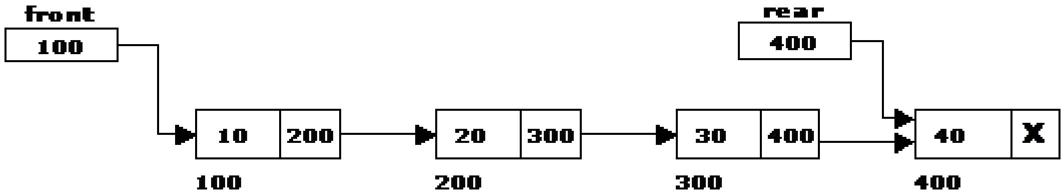
a. Write a Python program to implement queue using linked list

10.2 RESOURCES:

Python 3.4.0

10.3 PROGRAM LOGIC:

1. QUEUE: Queue is a linear data structure which works under the principle of first in first out. Basic operations: Insertion, deletion, display.
2. Inserion: if newnode ==NULL, display Queue is full. Else reading data and inserting at queue rear.
3. Deletion: if (front==NULL), display Queue is empty .Else printing element at queue front
4. Display: if (front==NULL) ,display No elements in the queue .Else printing the elements from front to rear.



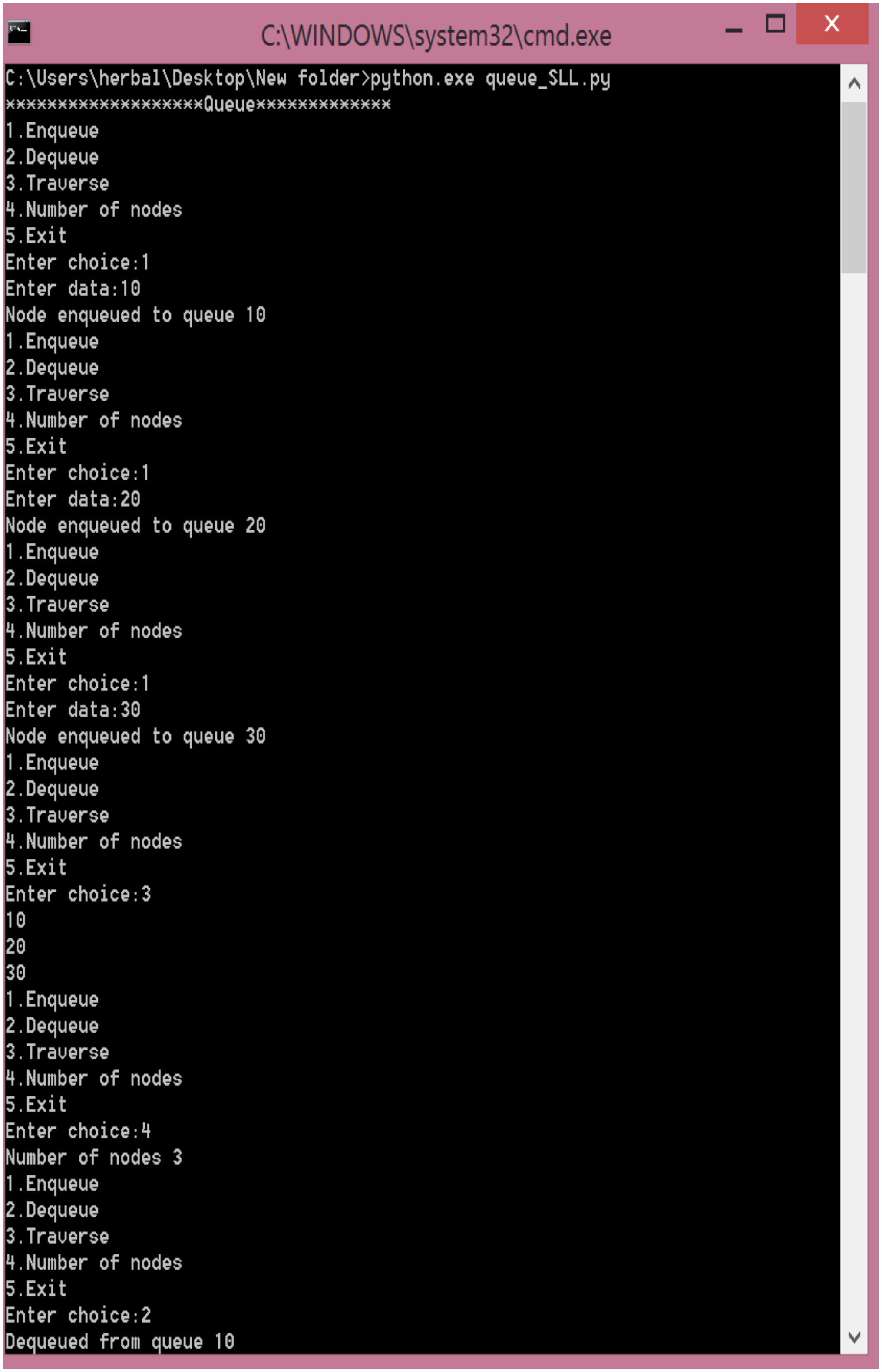
10.4 PROCEDURE:

1. Create: open Python GUI write a program after that save the program with .py extension.
2. Execute: F5

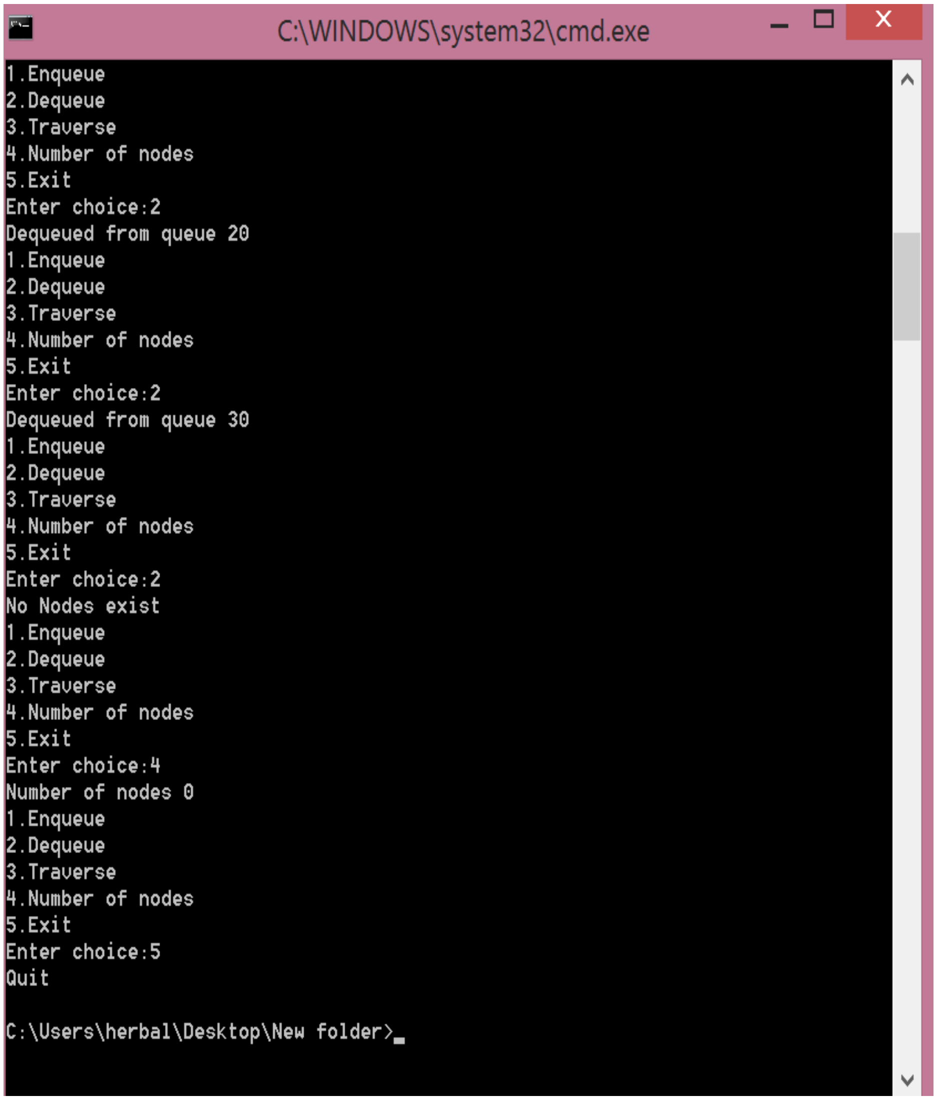
10.5 SOURCE CODE:

```
class Node:
    def __init__(self,data):
        self.data=data
        self.next=None

class Queue:
    def __init__(self):
        self.front=None
        self.ctr=0
        self.rear=None
    def Enqueue(self,data):
        node=Node(data)
        if self.front==None:
            self.front=node
            self.rear=node
        else:
            self.rear.next=node
            self.rear=node
    print("Node enqueued to queue",data)
    self.ctr+=1
    return
  def Dequeue(self):
    if self.front==None:
      print("No Nodes exist")
    else:
      print("Dequeued from queue",self.front.data)
      self.front=self.front.next
      self.ctr-=1
    return
 def Traverse(self):
   if self.front==None:
     print("No Nodes exist")
     return
   temp=self.front
   while temp is not None:
     print(temp.data)
     temp=temp.next

def Menu():
  print("1.Enqueue\n2.Dequeue\n3.Traverse\n4.Number of nodes\n5.Exit")
  ch=int(input("Enter choice:"))
  return ch

print("******************Queue*************")
s=Queue()
while True:
  ch=Menu()
  if ch==1:
    data=input("Enter data:")
    s.Enqueue(data)
  elif ch==2:
    s.Dequeue()
  elif ch==3:
    s.Traverse()
  elif ch==4:
    print("Number of nodes",s.ctr)
  else:
    print('Quit')
    break
```

10.6 INPUT/OUTPUT:-



```
C:\Users\herbal\Desktop\New folder>python.exe queue_SLL.py
******************Queue*************
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:10
Node enqueued to queue 10
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:20
Node enqueued to queue 20
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:30
Node enqueued to queue 30
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:3
10
20
30
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:4
Number of nodes 3
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Dequeued from queue 10
```



```
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Dequeued from queue 20
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Dequeued from queue 30
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
No Nodes exist
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:4
Number of nodes 0
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:5
Quit

C:\Users\herbal\Desktop\New folder>_
```

10.7 PRE-LAB VIVA QUESTIONS:

1. Which principle is followed in queue?
2. What are the applications of queue?

10.8 LAB ASSIGNMENT:

1. Write a program to implement Queue operations using linked list.
2. Formulate a program to implement circular queue operations using arrays.
3. Write a program to implement a priority queue?

10.9 POST-LAB VIVA QUESTIONS:

1. What is the advantage of circular queue over linear queue?
2. Where priority queues are used?
3. What is DEQUE?

WEEK-11

GRAPH TRAVERSAL TECHNIQUES

11.1 OBJECTIVE:

a. To write a Python program to implement depth first search.
b. To write a Python program to implement breadth first search.

11.2 RESOURCES:

Python GUI

11.3 PROGRAM LOGIC:

1. Take the graph as input and find the adjacency list
2. Start at a random vertex and visit all nodes using depth first search (DFS) and then breadth first search (BFS).
3. Use stack for DFS and queue for BFS.

11.4 PROCEDURE:

1. Create: open Python shell write a program after that save the program with .py extension.
2. Execute: F5

11.5 SOURCE CODE:

Depth First Search Program

```
import defaultdict

class Graph:

    # Constructor
    def __init__(self):

        self.graph = defaultdict(list)

    def addEdge(self,u,v):
        self.graph[u].append(v)

    def DFSUtil(self,v,visited):

        visited[v]= True
        print (v),

        for i in self.graph[v]:
            if visited[i] == False:
                self.DFSUtil(i, visited)

    def DFS(self,v):

        visited = [False]*(len(self.graph))
        self.DFSUtil(v,visited)
```

```
   g = Graph()
   g.addEdge(0, 1)
   g.addEdge(0, 2)
   g.addEdge(1, 2)
   g.addEdge(2, 0)
   g.addEdge(2, 3)
   g.addEdge(3, 3)

   print ("Following is DFS from (starting from vertex 2)")
   g.DFS(2)
```

Breadth First Search Program

```
import defaultdict

class Graph:
    def __init__(self):

        self.graph = defaultdict(list)


    def addEdge(self,u,v):
        self.graph[u].append(v)

    def BFS(self, s):

        visited = [False]*(len(self.graph))

        queue = []

        queue.append(s)
        visited[s] = True

        while queue:

            s = queue.pop(0)
            print (s)

            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True

g = Graph()
g.addEdge(0, 1)
g.addEdge(0, 2)
g.addEdge(1, 2)
g.addEdge(2, 0)
g.addEdge(2, 3)
g.addEdge(3, 3)
```

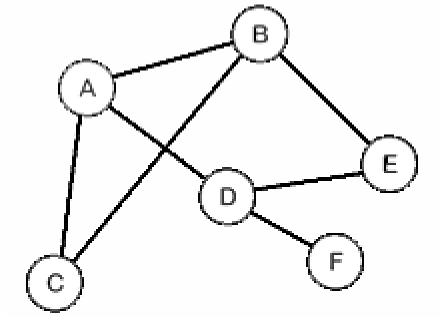
```
print  ("Following is Breadth First Traversal (starting from vertex 2)")
g.BFS(2)
```

11.7 PRE-LAB VIVA QUESTIONS:

1. What is graph?
2. List various way of representations of graph?
3. How many graph traversal algorithms are there?

11.8 LAB ASSIGNMENT:

1. Find DFS traversal of the following graph



2. Deduce the time complexity of DFS algorithm

11.9 POST-LAB VIVA QUESTIONS:

1. Applications of graph traversals?
2. Define minimum spanning tree?
3. What is the time complexity of DFS?

WEEK-12

IMPLEMENTATION OF BINARY SEARCH TREE

12.1 OBJECTIVE:

To write a Python program to implement binary search tree creation, traversal and count node.

12.2 RESOURCES:

Python 3.4.0

12.3 PROGRAM LOGIC:

1. The left sub tree of a node contains smaller nodes than a root node.
2. The right sub tree of a node contains greater nodes than a root node.
3. Both the left and right sub trees must also be binary search trees.
4. There are three types of tree traversals: Preorder, Postorder, and Inorder.

Pre-order traversal

Algorithm:

```
1. Visit the root (we will print it when we visit to show the order of visiting)
2. Traverse the left subtree in pre-order
3. Traverse the right subtree in pre-order
```

In-order traversal

Visit the root node in between the left and right node (in)

Algorithm:

```
1. Traverse the left subtree in in-order
2. Visit the root (we will print it when we visit to show the order of visiting)
3. Traverse the right subtree in in-order
```

Post-order traversal

Visit the root node after (post) visiting the left and right subtree.

Algorithm:

```
1. Traverse the left subtree in in-order
2. Traverse the right subtree in in-order
3. Visit the root (we will print it when we visit to show the order of visiting)
```

Maximum depth or Height of a tree

Algorithm:

```
 maxDepth()
1. If tree is empty then return 0
2. Else
    (a) Get the max depth of left subtree recursively  i.e.,
         call maxDepth( tree->left-subtree)
    (a) Get the max depth of right subtree recursively  i.e.,
         call maxDepth( tree->right-subtree)
    (c) Get the max of max depths of left and right
```

```
          subtrees and add 1 to it for the current node.
          max_depth = max(max dept of left subtree,
                          max depth of right subtree)
                          + 1
     (d) Return max_depth
```

Count number of leaf nodes in a binary tree

A node is a leaf node if both left and right child nodes of it are NULL.

Algorithm

```
getLeafCount(node)
1) If node is NULL then return 0.
2) Else If left and right child nodes are NULL return 1.
3) Else recursively calculate leaf count of the tree using below formula.
 Leaf count of a tree=Leaf count of left sub tree + leaf count of right sub tree
```

12.4 PROCEDURE:

1. Create: open python shell write a program after that save the program with .py extension.
2. Execute: F5

12.5 SOURCE CODE:

```
class Node:

    def __init__(self,info): #constructor of class
        self.info = info  #information for node
        self.left = None  #left leef
        self.right = None #right leef
        self.level = None #level none defined
    def __str__(self):
        return str(self.info) #return as string

class searchtree:
    def __init__(self): #constructor of class
        self.root = None
    def create(self,val):  #create binary search tree nodes
        if self.root == None:
            self.root = Node(val)
        else:
            current = self.root
            while 1:
                if val < current.info:
                    if current.left:
                        current = current.left
                    else:
                        current.left = Node(val)
                        break;
                elif val > current.info:
                    if current.right:
                        current = current.right
                    else:
                       current.right = Node(val)
                       break;
                 else:
                    break

      def bft(self): #Breadth-First Traversal
          self.root.level = 0
          queue = [self.root]
          out = []
          current_level = self.root.level
          while len(queue) > 0:
             current_node = queue.pop(0)
             if current_node.level > current_level:
                current_level += 1
                out.append("\n")
             out.append(str(current_node.info) + " ")
             if current_node.left:
                current_node.left.level = current_level + 1
                queue.append(current_node.left)
             if current_node.right:
                current_node.right.level = current_level + 1
                queue.append(current_node.right)
          result= "".join(out)
          print  (result)
      def inorder(self,node):
           if node is not None:
              self.inorder(node.left)
              print (node.info)
              self.inorder(node.right)
      def preorder(self,node):
           if node is not None:
              print (node.info)
              self.preorder(node.left)
              self.preorder(node.right)
      def postorder(self,node):
           if node is not None:
              self.postorder(node.left)
              self.postorder(node.right)
              print (node.info)

tree = searchtree()
arr = [8,3,1,6,4,7,10,14,13]
for i in arr:
    tree.create(i)
print ('Breadth-First Traversal')
tree.bft()
print ('Inorder Traversal')
tree.inorder(tree.root)
print ('Preorder Traversal')
tree.preorder(tree.root)
print ('Postorder Traversal')
tree.postorder(tree.root)
```

Output:

```
Breadth-First Traversal
8
3 10
1 6 14
4 7 13
Inorder Traversal
1
3
4
6
7
8
10
13
14
Preorder Traversal
8
3
1
6
4
7
10
14
13
Postorder Traversal
1
4
7
6
3
13
14
10
8
```

Count the number of nodes in the binary search tree.

```
class BinaryTree:
    def __init__(self, data):
        self.data = data
        self.left = None
        self.right = None
    def insert_left(self, new_data):
        if self.left == None:
            self.left = BinaryTree(new_data)
        else:
            t = BinaryTree(new_data)
            t.left = self.left
            self.left = t
    def insert_right(self, new_data):
        if self.right == None:
            self.right = BinaryTree(new_data)
        else:
            t = BinaryTree(new_data)
            t.right = self.right
            self.right = t
    def get_left(self):
        return self.left
    def get_right(self):
        return self.right
    def set_data(self, data):
        self.data = data
    def get_data(self):
        return self.data
def size(my_tree):
      if not my_tree:
            return 0
      return 1 + size(my_tree.get_left()) + size(my_tree.get_right())

a = BinaryTree(1)
a.insert_left(2)
a.insert_right(3)
print(size(a))
```

Output:

```
No of nodes: 3
```

12.7 PRE-LAB VIVA QUESTIONS:

1. Define tree traversal and mention types of traversal?
2. Define a tree?
3. Define height of a tree?
4. Define depth of a tree?
5. Define degree of a node?
6. Define Degree of a tree?
7. Define Terminal node or leaf node?
8. Define Non-terminal node?
9. Define Sibling?
10. Define Binary Tree?
11. Write the properties of Binary Tree?
12. Find the minimum and maximum height of a binary tree?

12.8 LAB ASSIGNMENT:

1. Formulate a program to create a Binary Tree of integers?
2. Write a recursive program, for traversing a binary tree in preorder, inorder and postorder?
3. Compose a non-recursive program, for traversing a binary tree in preorder, inorder and postorder?
4. Write a program to check balance property of a tree?

12.9 POST-LAB VIVA QUESTIONS:

1. Write the balance factor of a Binary Tree?
2. What are the data structures used for Binary Trees?
3. Define a Complete Binary Tree?
4. List out the applications of Binary Tree?
5. Write the two approaches for Binary Tree Traversal?
6. Write the various operation performed in the binary search tree?
7. Write the three approaches for inserting data into general trees?
8. Define pre-order traversal.
9. Define post-order traversal.
10. Define in-order traversal.