DATA STRUCTURES LABORARTORY
LAB MANUAL

Academic Year	**:**	**2019 - 2020**
Course Code	**:**	**ACSB05**
Regulations	**:**	**IARE – R18**
Semester	**:**	**III**
Branch	**:**	**CSE | IT | ECE | CE | ME**

Prepared by

Dr. J Sirisha Devi
Associate Professor

INSTITUTE OF AERONAUTICAL ENGINEERING
(Autonomous)
Dundigal, Hyderabad - 500 043

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500043

Program Outcomes	
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 (ME)	
PSO1	To produce engineering professional capable of synthesizing and analyzing mechanical systems including allied engineering streams.
PSO2	An ability to adopt and integrate current technologies in the design and manufacturing domain to enhance the employability.
PSO3	To build the nation, by imparting technological inputs and managerial skills to become Technocrats.

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad – 500043

ATTAINMENT OF PROGRAM OUTCOMES
& PROGRAM SPECIFIC OUTCOMES

S No	Experiment	Program Outcome Attained	Program Specific Outcomes Attained
1	BASICS OF PYTHON	PO1, PO2, PO3	PSO1, PSO3
2	SEARCHING TECHNIQUES	PO1, PO2, PO3	PSO1, PSO3
3	SORTING TECHNIQUES	PO1, PO2, PO3	PSO1, PSO3
4	IMPLEMENTATION OF STACK AND QUEUE	PO2, PO3	PSO1, PSO3
5	APPLICATIONS OF STACK	PO3, PO4	PSO1, PSO3
6	IMPLEMENTATION OF SINGLE LINKED LIST	PO2, PO3	PSO1, PSO3
7	IMPLEMENTATION OF CIRCULAR LINKED LIST	PO2, PO3	PSO1, PSO3
8	IMPLEMENTATION OF DOUBLE LINKED LIST	PO2, PO3	PSO1, PSO3
9	IMPLEMENTATION OF STACK USING LINKED LIST	PO3, PO4	PSO1, PSO3
10	IMPLEMENTATION OF QUEUE USING LINKED LIST	PO3, PO4	PSO1, PSO3
11	GRAPH TRAVERSAL TECHNIQUES	PO2, PO3	PSO1, PSO3
12	IMPLEMENTATION OF BINARY SEARCH TREES	PO2, PO3	PSO1, PSO3

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

III Semester: ME / CSE / IT / ECE / CE | IV Semester AE / EEE

Course Code	Category	Hours / Week			Credits	Maximum Marks		
		L	T	P	C	CIA	SEE	Total
ACSB05	Core	0	0	3	1.5	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 BASICS OF PYTHON

Write Python programs for the following:
a. To find the biggest of given n numbers using control statements and lists
b. To print the Fibonacci series using functions
c. To find GCD of two numbers

Week -2 SEARCHING TECHNIQUES

Write Python programs for implementing the following searching techniques to arrange a list of integers in ascending order.
a. Linear search
b. Binary search

Week -3 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 -4 IMPLEMENTATION OF STACK AND QUEUE

Write Python programs to for the following:
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

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

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 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 andin-order.
c. Count the number of nodes in the binary search tree.

LIST OF REFERENCE BOOKS:

1. Rance D. Necaise, “Data Structures and Algorithms using Python”, Wiley, John Wiley & Sons, INC., 2011.
2. Benjamin Baka, David Julian, “Python Data Structures and Algorithms”, Packt Publishing Ltd., 2017.

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
7. http://cse01-iiith.vlabs.ac.in/

WEEK – 1
BASICS OF PYTHON

OBJECTIVE:

a. Write a Python script to find the biggest of the given numbers using control statements and lists
b. Write a Python script to print the Fibonacci series using functions.
c. Write a Python script to find the GCD of two numbers.

RESOURCE:
Python 3.7.3

PROGRAM LOGIC:
Biggest of the given numbers using control statements and list:

1. Read a list of integers.
2. Assume the first number as maximum number.
3. Compare each number n with the maximum number and if n is bigger than max then change max with n.
4. Repeat this process for all numbers.
5. Return max

Fibonacci series using function:

1. Read number of terms n.
2. Send n to recursive method recur_fibo()
3. if n <= 1 then return n
4. otherwise return(recur_fibo(n-1) + recur_fibo(n-2))

GCD of two numbers:

1. Read two integers n1 and n2.
2. Send n to recursive method computeGCD(n1, n2).
3. Find the smaller number by checking if n1 > n2 then smaller = n2, otherwise smaller = n1
4. for each number i, compute if((n1 % i == 0) and (n2 % i == 0)) then gcd = i
5. return gcd

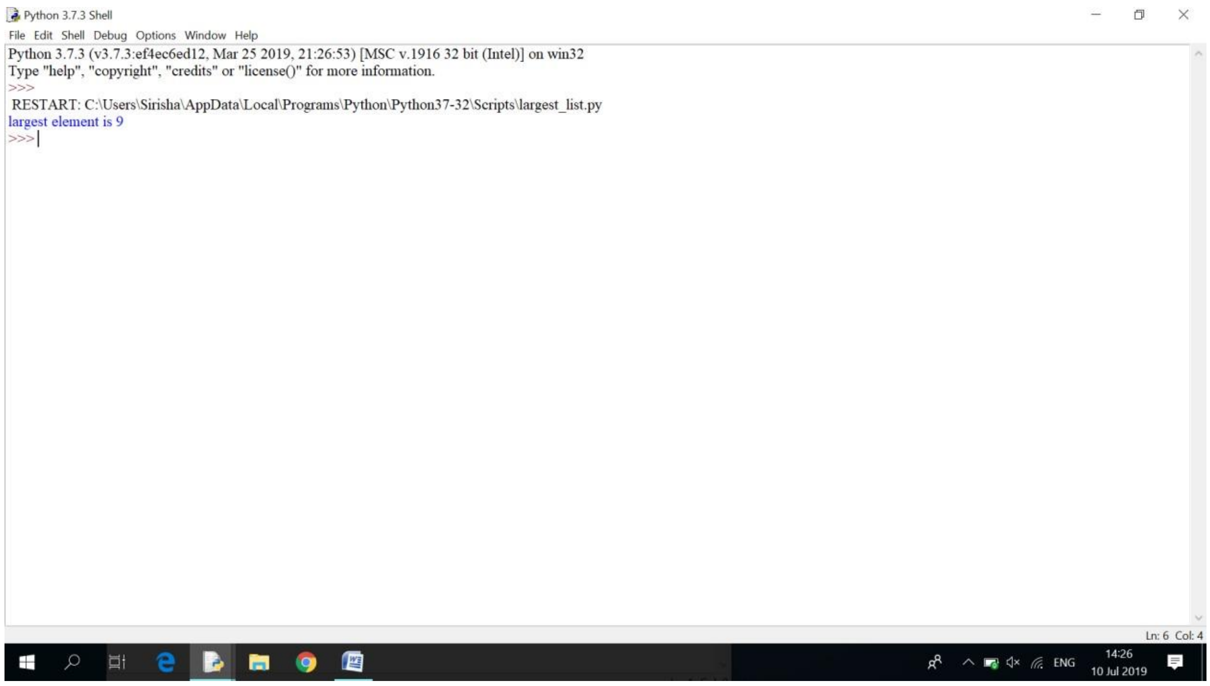
PROCEDURE:

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

SOURCE CODE:
Biggest of the given numbers using control statements and list:

```
def large(arr):
    n=len(arr)
    l=0
    for i in range(0,n-1):
        if arr[i]>l:
            l=arr[i]
    print("largest element is %d" %l)

# Driver code
arr=[3,2,4,1,5,8,6,9,7]
large(arr)
```

Output:

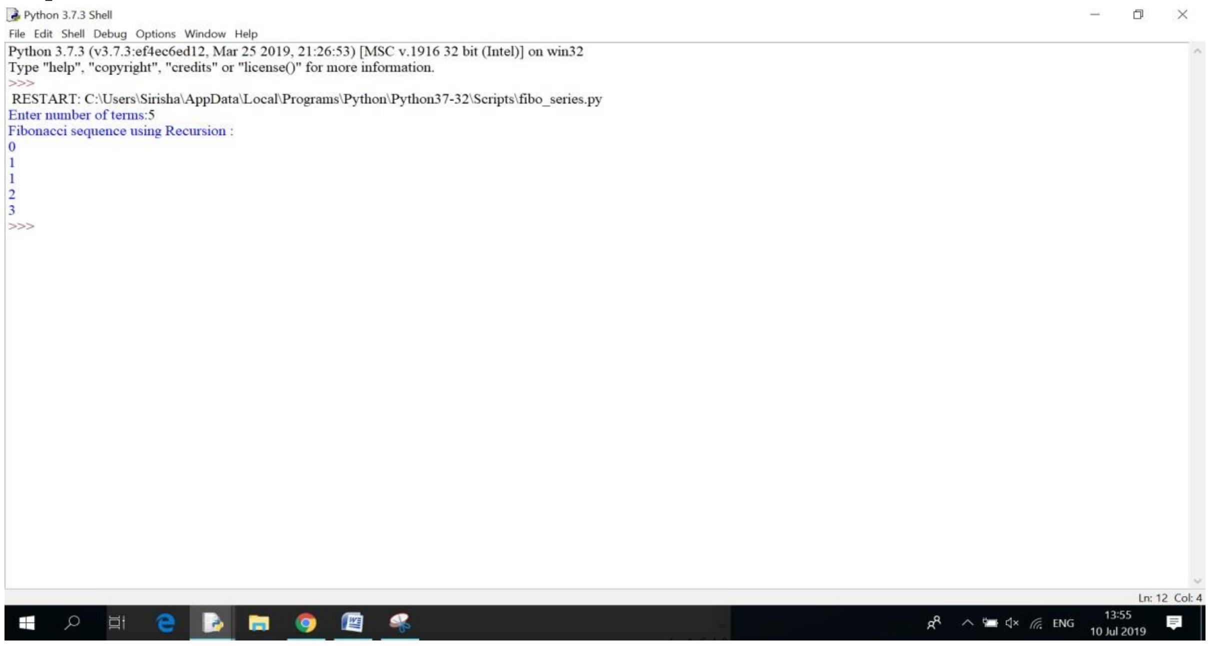


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\largest_list.py
largest element is 9
>>>
```

Fibonacci series using function:

```
def gen_seq(length):
   if(length <= 1):
      return length
   else:
      return (gen_seq(length-1) + gen_seq(length-2))

length = int(input("Enter number of terms:"))

print("Fibonacci sequence using Recursion :")
for iter in range(length):
   print(gen_seq(iter))
```

Output:

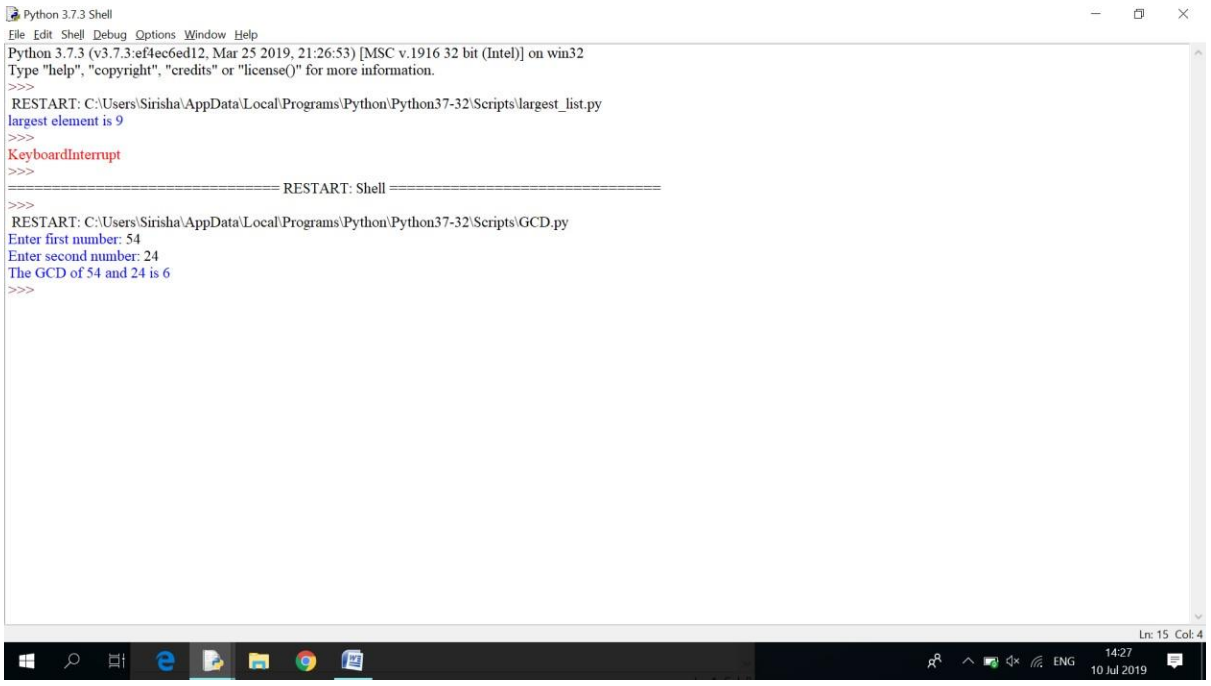


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\fibo_series.py
Enter number of terms:5
Fibonacci sequence using Recursion :
0
1
1
2
3
>>>
```

GCD of two numbers:

```
def computeGCD(x, y):

# choose the smaller number
    if x > y:
        smaller = y
    else:
        smaller = x
    for i in range(1, smaller+1):
        if((x % i == 0) and (y % i == 0)):
            gcd= i

    return gcd

#Driver Code
num1 = int(input("Enter first number: "))
num2 = int(input("Enter second number: "))

print("The GCD of", num1,"and", num2,"is", computeGCD(num1, num2))
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\largest_list.py
largest element is 9
>>>
KeyboardInterrupt
>>>
================================ RESTART: Shell ================================
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\GCD.py
Enter first number: 54
Enter second number: 24
The GCD of 54 and 24 is 6
>>>
```

PRE LAB VIVA QUESTIONS:

a. What is Python? What are the benefits of using Python?
b. How memory is managed in Python?
c. In Python what is slicing?
d. What are the different ways of accessing elements in a list?
e. State any five built-in functions used in lists?

LAB ASSIGNMENT:

a. Write a Python program to find the factors of a number?
b. Write a Python program to find the factorial of a number using recursion?

c. Write a Python program to check if the input number is prime or not?
d. Write a Python program to find the sum of natural numbers up to n using recursive function?
e. Write a Python program to display all the prime numbers within an interval?

POST LAB VIVA QUESTIONS:

a. What is the difference between list and tuple?
b. What are the built-in type does python provides?
c. State the built-in set operators?
d. Define class, object, attribute and method?
e. What is lambda in Python?

WEEK – 2
SEARCHING TECHNIQUES

OBJECTIVES:

a. Write a Python script to implement linear search technique.
b. Write a Python script to implement binary search technique.

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Linear search technique:

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.
    a. If x doesn't match with any of elements, return -1.

Binary search technique:

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.

PROCEDURE:

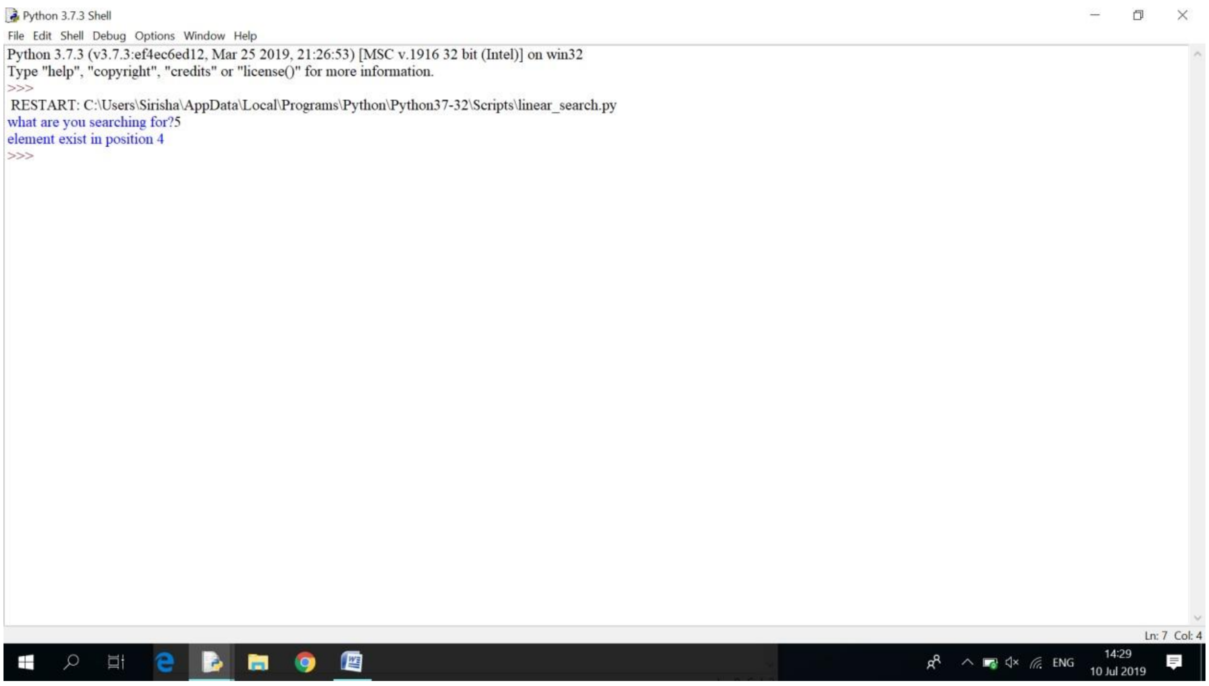
a. Create : Open a new file in Python shell, write a program and save the program with .py extension.
b. Execute : Go to Run -> Run module (F5)

SOURCE CODE:

Linear search technique:

```
def linear_search(obj, item):
    for i in range(len(obj)):
        if obj[i] == item:
            return i
    return -1

#Driver code
arr=[1,2,3,4,5,6,7,8]
x=int(input("what are you searching for?"))
result=linear_search(arr,x)

if result==-1:
    print ("element does not exist")
else:
    print ("element exist in position %d" %result)
```

Output:

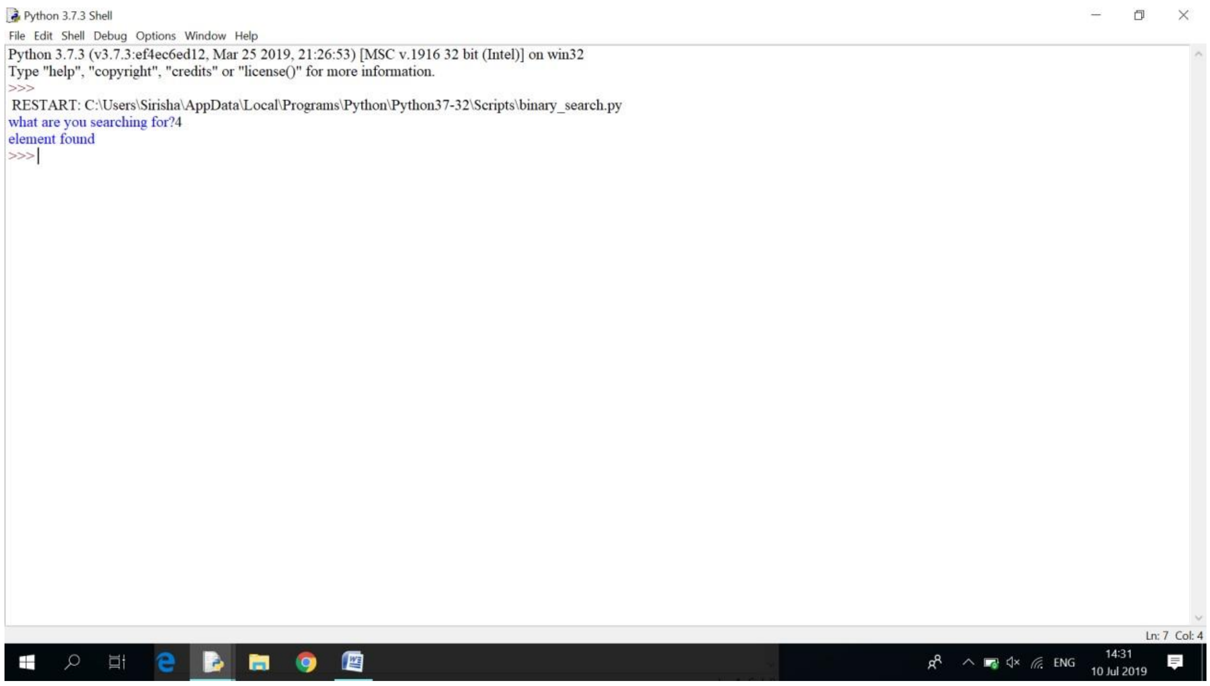


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\linear_search.py
what are you searching for?5
element exist in position 4
>>>
```

Binary search technique:

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

def binary_search(searchfor,array):
    lowerbound=0
    upperbound=len(array)-1
    found=False
    while found==False and lowerbound<=upperbound:
        midpoint=(lowerbound+upperbound)//2
        if array[midpoint]==searchfor:
            found =True
            return found
        elif array[midpoint]<searchfor:
            lowerbound=midpoint+1
        else:
            upperbound=midpoint-1
    return found

#Driver code
searchfor=int(input("what are you searching for?"))
if binary_search(searchfor,array):
    print ("element found")
else:
    print ("element not found")
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\binary_search.py
what are you searching for?4
element found
>>>|
```

PRE LAB VIVA QUESTIONS:

a. Define searching process?
b. How many types of searching are there?
c. Why binary search method is more efficient then liner search?
d. What is worse case?

LAB ASSIGNMENT:

a. 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.
b. Use linear search technique to search for a key value in a given list of characters and print the message found or not.

POST LAB VIVA QUESTIONS:

a. What do you understand by the term “linear search is unsuccessful”?
b. Efficiency of linear search?
c. What is the drawback of linear search?

WEEK – 3
SORTING TECHNIQUES

OBJECTIVES:

a. Write a Python script to implement bubble sort.
b. Write a Python script to implement insertion sort.
c. Write a Python script to implement selection sort.

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Bubble sort:

1. Starting with the first element(index = 0), compare the current element with the next element of the array.
2. If the current element is greater than the next element of the array, swap them.
3. If the current element is less than the next element, move to the next element. Repeat Step 1.

Insertion sort:

1. It is efficient for smaller data sets, but very inefficient for larger lists.
2. Insertion Sort is adaptive, that means it reduces its total number of steps if a partially sorted array is provided as input, making it efficient.
3. It is better than Selection Sort and Bubble Sort algorithms.
4. Its space complexity is less. Like bubble Sort, insertion sort also requires a single additional memory space.
5. It is a stable sorting technique, as it does not change the relative order of elements which are equal.

Selection sort:

1. Starting from the first element, we search the smallest element in the array, and replace it with the element in the first position.
2. We then move on to the second position, and look for smallest element present in the subarray, starting from index 1, till the last index.
3. We replace the element at the second position in the original array, or we can say at the first position in the subarray, with the second smallest element.
4. This is repeated, until the array is completely sorted.

PROCEDURE:

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

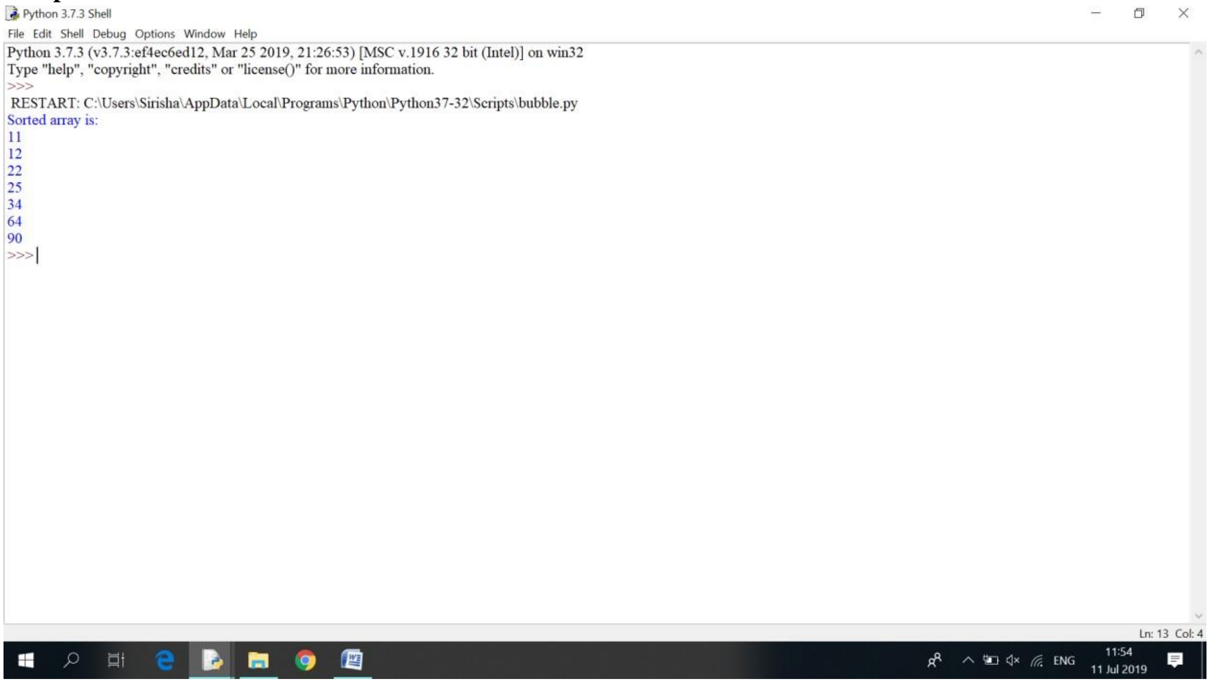
SOURCE CODE:

Bubble sort:

```
def bubbleSort(arr):
    n = len(arr)

    # Traverse through all array elements
    for i in range(n):

        # Last i elements are already in place
        for j in range(0, n-i-1):

            # traverse the array from 0 to n-i-1
            # Swap if the element found is greater
            # than the next element
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
# Driver code to test above
arr = [64, 34, 25, 12, 22, 11, 90]

bubbleSort(arr)

print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i])
```

Output:

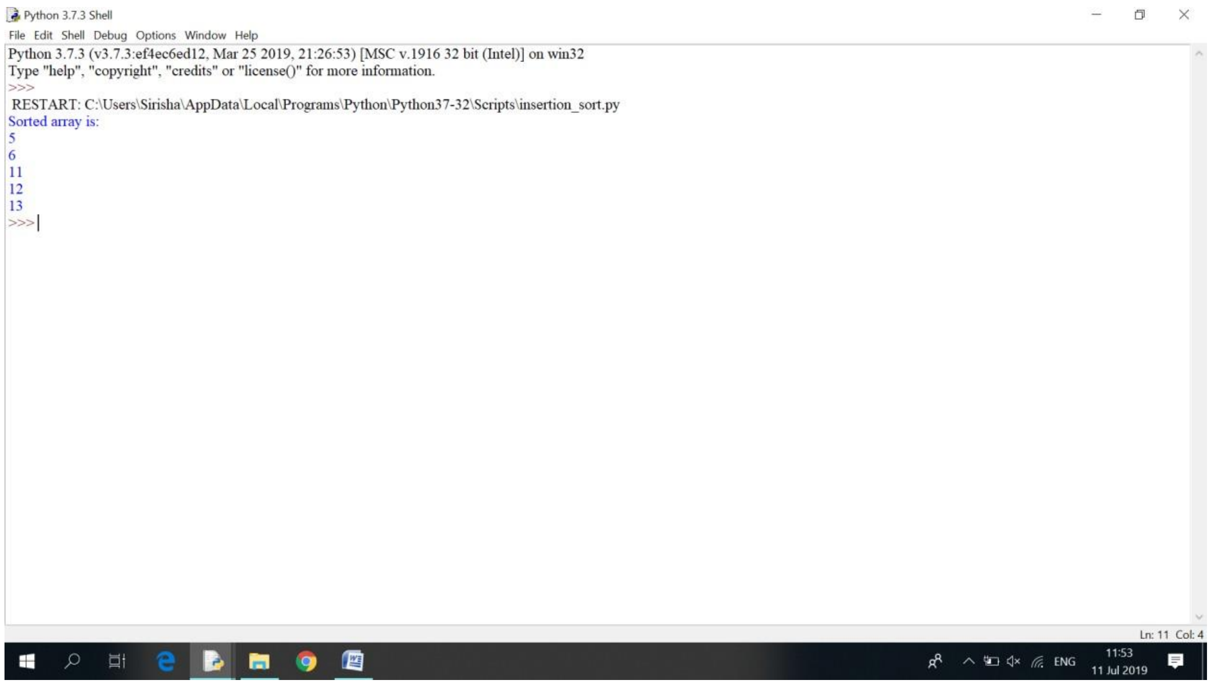


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\bubble.py
Sorted array is:
11
12
22
25
34
64
90
>>>|
```

Insertion sort:

```
def insertionSort(arr):

    # Traverse through 1 to len(arr)
    for i in range(1, len(arr)):
        key = arr[i]

        # Move elements of arr[0..i-1], that are
        # greater than key, to one position ahead
        # of their current position
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key


# Driver code to test above
arr = [12, 11, 13, 5, 6]
insertionSort(arr)
print ("Sorted array is:")
for i in range(len(arr)):
    print ("%d" %arr[i])
```

Output:

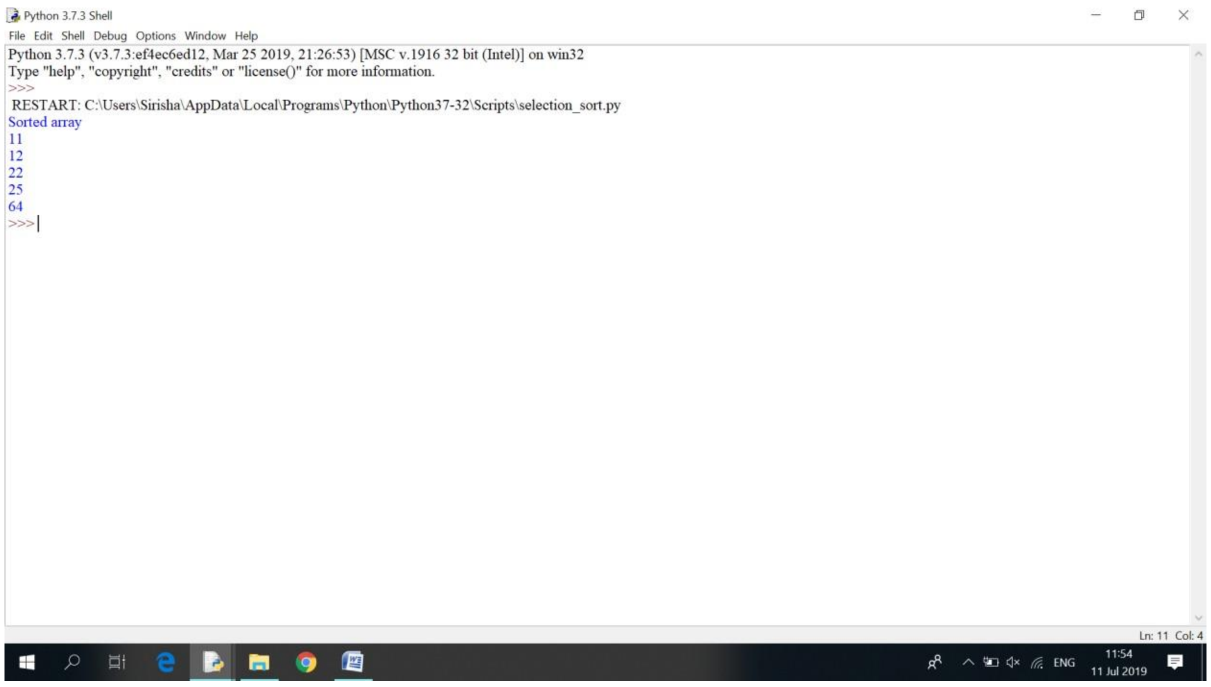


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\insertion_sort.py
Sorted array is:
5
6
11
12
13
>>>
```

Selection sort:

```
import sys
A = [64, 25, 12, 22, 11]

# Traverse through all array elements
for i in range(len(A)):

    # Find the minimum element in remaining
    # unsorted array
    min_idx = i
    for j in range(i+1, len(A)):
        if A[min_idx] > A[j]:
            min_idx = j

    # Swap the found minimum element with the first element

    A[i], A[min_idx] = A[min_idx], A[i]

# Driver code to test above
print ("Sorted array")
for i in range(len(A)):
    print("%d" %A[i])
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\selection_sort.py
Sorted array
11
12
22
25
64
>>>
```

PRE LAB VIVA QUESTIONS:

a. Explain the term sorting?
b. What are the different types of sorts in data structures?
c. Define the bubble sort?
d. Define the insertion sort?
e. Define the selection sort?

LAB ASSIGNMENT:

a. Formulate a program that implement Bubble sort, to sort a given list of integers in descending order.
b. Compose a program that implement Insertion sort, to sort a given list of integers in descending order.
c. Write a program that implement Selection sort, to sort a given list of integers in ascending order.
d. Formulate a program to sort N names using selection sort.
e. Write a program to sort N employee records based on their salary using insertion sort.
f. 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.

POST LAB VIVA QUESTIONS:

a. How many passes are required in selection sort?
b. Write the time complexity of insertion sort?
c. Write the time complexity of selection sort?
d. Write the time complexity of bubble sort?

WEEK – 4
IMPLEMENTATION OF STACKS AND QUEUES

OBJECTIVES:

a. Write a Python script to design and implement stack and its operations using list.
b. Write a Python script to design and implement queue and its operations using list.

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Stack and its operations using list:

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

Queue and its operations using list:

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

PROCEDURE:

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

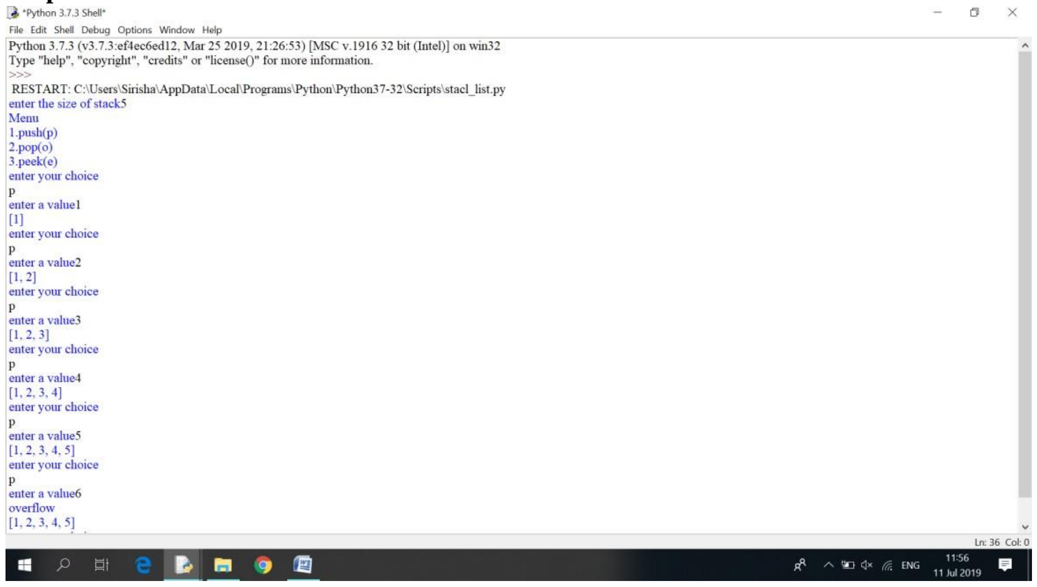
SOURCE CODE:

Stack and its operations using list:

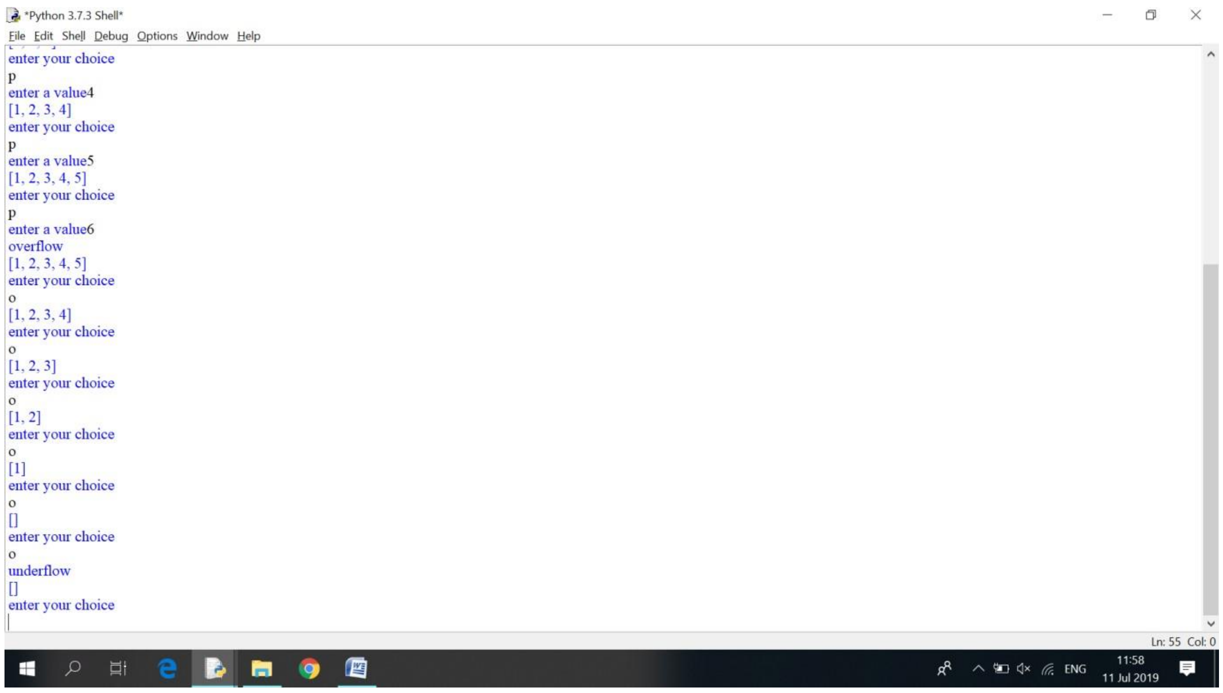
```
  # Function to create a stack. It initializes size of stack as 0
def createStack():
    stack = []
    return stack

# Stack is empty when stack size is 0
def isEmpty(stack):
    return len(stack) == 0

# Function to add an item to stack. It increases size by 1
def push(stack, item):
    if(len(stack)==size):
        print("overflow")
        return
    stack.append(item)

# Function to remove an item from stack. It decreases size by 1
def pop(stack):
    if   (isEmpty(stack)):
        print("underflow")
```

```
            return
        return stack.pop()

    #Function to know peek element
    def peek(stack):
        if(isEmpty(stack)):
            print("stack empty")
            return
        else:
            n=len(stack)
            print("peek element is: ",stack[n-1])

    #Function to display stack
    def display(stack):
        print(stack)

    # Driver program to test above functions
    stack = createStack()
    size=int(input("enter the size of stack"))


    print("Menu\n1.push(p)\n2.pop(o)\n3.peek(e)")

    choice=1
    while choice!='q':
        print("enter your choice")
        ch=input()
        choice=ch.lower()
        if choice=='p':
            push(stack,int(input("enter a value")))
            display(stack)
        elif choice=='o':
            pop(stack)
            display(stack)
        elif choice=='e':
            peek(stack)
        else:
            print("enter proper choice or q - quit")
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\stacl_list.py
enter the size of stack5
Menu
1.push(p)
2.pop(o)
3.peek(e)
enter your choice
p
enter a value1
[1]
enter your choice
p
enter a value2
[1, 2]
enter your choice
p
enter a value3
[1, 2, 3]
enter your choice
p
enter a value4
[1, 2, 3, 4]
enter your choice
p
enter a value5
[1, 2, 3, 4, 5]
enter your choice
p
enter a value6
overflow
[1, 2, 3, 4, 5]
```



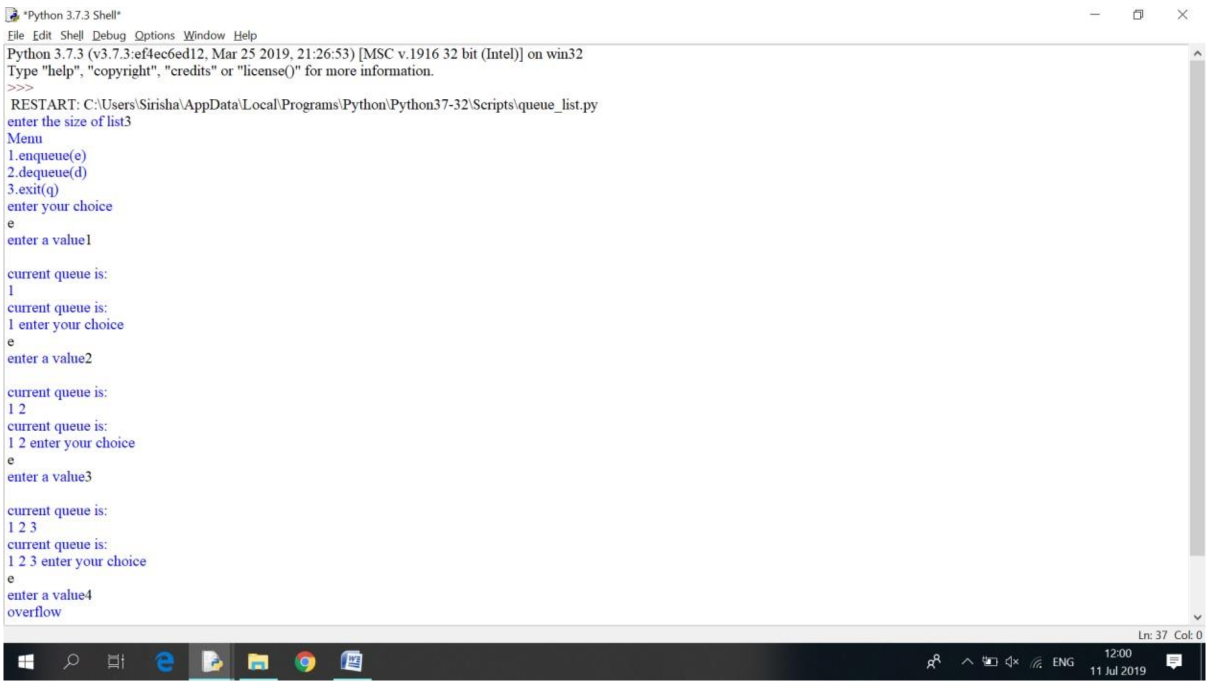
```
enter your choice
p
enter a value4
[1, 2, 3, 4]
enter your choice
p
enter a value5
[1, 2, 3, 4, 5]
enter your choice
p
enter a value6
overflow
[1, 2, 3, 4, 5]
enter your choice
o
[1, 2, 3, 4]
enter your choice
o
[1, 2, 3]
enter your choice
o
[1, 2]
enter your choice
o
[1]
enter your choice
o
[]
enter your choice
o
underflow
[]
enter your choice
```

Queue and its operations using list:

```
def enqueue(a,item):
    global r
    global f

    if r==-1 and f==-1:
        r=0
        f=0
        a.insert(r,item)

    elif r==(n-1):
        print("overflow")
        return
    else:
        r+=1
        a.insert(r,item)

    display(a)

def dequeue(a):
    global r
    global f
    if r==(n-1) and f==(n-1):
        item=a[f]
        r=-1
        f=-1
    elif r==-1 and f==-1:
        print("underflow")
        return
    else:
        item=a[f]
        f+=1
    print("deleted item is:",item)
    display(a)
```

```
def display(a):
    print("\ncurrent queue is:")
    for i in range(f,r+1):
        if f==-1 and r==-1:
            print("Queue is empty!")
            return
        print(a[i],end=" ")

#DC
n=int(input("enter the size of list"))
a=[]
r=-1
f=-1
print("Menu\n1.enqueue(e)\n2.dequeue(d)\n3.exit(q)")

choice=1
while choice!='q':
    print("enter your choice")
    ch=input()
    choice=ch.lower()
    if choice=='e':
        enqueue(a,int(input("enter a value")))
        display(a)
    elif choice=='d':
        dequeue(a)
        display(a)

    else:
        print("enter proper choice")
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\queue_list.py
enter the size of list3
Menu
1.enqueue(e)
2.dequeue(d)
3.exit(q)
enter your choice
e
enter a value1

current queue is:
1
current queue is:
1 enter your choice
e
enter a value2

current queue is:
1 2
current queue is:
1 2 enter your choice
e
enter a value3

current queue is:
1 2 3
current queue is:
1 2 3 enter your choice
e
enter a value4
overflow
```

PRE LAB VIVA QUESTIONS:

a. What is stack?
b. What are the operations performed on stack?

c. How stacks are implemented?
d. What are the applications of stack?
e. What is recursion?
f. Define “Top of stack”.
g. How to implement stack?
h. Define a queue?
i. Define the condition “overflow”.
j. Define the condition “underflow”.
k. Define a queue.
l. Which principle is followed in queue?
m. List out the applications of queue?

LAB ASSIGNMENT

a. Write a program to implement stack and its operations using arrays.
b. Formulate a program to reverse a list of numbers using stack.
c. Write a program to find the factorial of a number using stack.
d. Develop a program to check a given expression is balanced or not using stack
e. Compose a program to implement Queue operations using arrays.
f. Formulate a program to implement circular queue operations using arrays.
g. Write a program to implement a priority queue?

POST LAB VIVA QUESTIONS:

a. Write the time complexity of PUSH operation?
b. Write the time complexity of POP operation?
c. List out the applications of stack?
d. How to remove an element from stack?
e. How to insert an element into a stack?
f. Write the time complexity to insert an element into a queue?
g. Write the time complexity to delete an element from a queue?
h. List out the advantage of circular queue over linear queue?
i. Define a priority queue?
1. Define DEQUE?

WEEK – 5
APPLICATIONS OF STACKS

OBJECTIVES:

a. Write a Python script that uses stack operations to convert infix expression to postfix expression.
b. Write a Python script that uses stack operations for evaluating the postfix expression.

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Infix expression to postfix expression:

Let, X is an arithmetic expression written in infix notation. This algorithm finds the equivalent postfix expression Y.

```
1. Push “(“onto Stack, and add “)” to the end of X.
2. Scan X from left to right and repeat Step 3 to 6 for each element of X until the Stack is empty.
3. If an operand is encountered, add it to Y.
4. If a left parenthesis is encountered, push it onto Stack.
5. If an operator is encountered ,then:
      1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) which has the
         same precedence as or higher precedence than operator.
      2. Add operator to Stack.
         [End of If]
6. If a right parenthesis is encountered ,then:
      1. Repeatedly pop from Stack and add to Y each operator (on the top of Stack) until a left
         parenthesis is encountered.
      2. Remove the left Parenthesis.
         [End of If]
         [End of If]
7. END.
```

Evaluation of the postfix expression:

```
1. Create a stack to store operands (or values).
2. Scan the given expression and do following for every scanned element.
1. If the element is a number, push it into the stack
2. If the element is a operator, pop operands for the operator from stack. Evaluate the operator and push
   the result back to the stack
3. When the expression is ended, the number in the stack is the final answer
```

PROCEDURE:

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

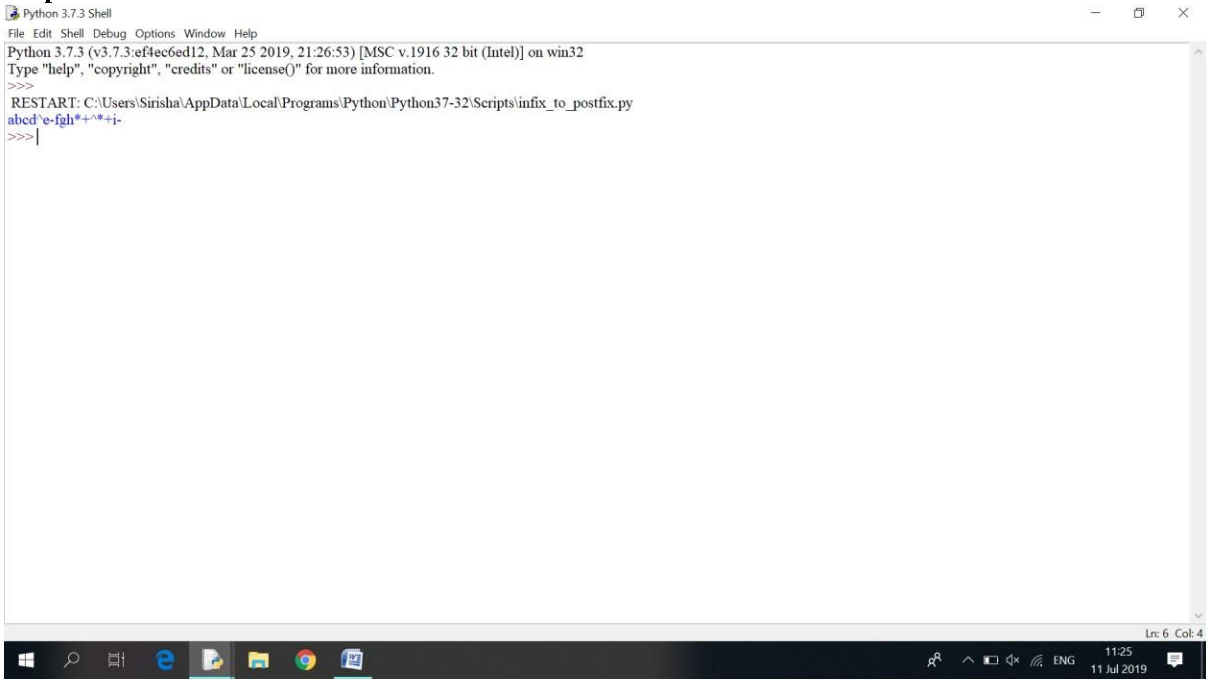
SOURCE CODE:

Infix expression to postfix expression:

```
import string
class Conversion:

  # Constructor to initialize the class variables
  def_init_(self, capacity):
    self.top = -1
    self.capacity = capacity
    # This array is used a stack
    self.array = []
    # Precedence setting
    self.output = []
    self.precedence = {'+':1, '-':1, '*':2, '/':2, '^':3}
```

```
# check if the stack is empty
def isEmpty(self):
    return True if self.top == -1 else False

# Return the value of the top of the stack
def peek(self):
    return self.array[-1]

# Pop the element from the stack
def pop(self):
    if not self.isEmpty():
        self.top -= 1
        return self.array.pop()
    else:
        return "$"

# Push the element to the stack
def push(self, op):
    self.top += 1
    self.array.append(op)

# A utility function to check is the given character
# is operand
def isOperand(self, ch):
    return ch.isalpha()

# Check if the precedence of operator is strictly
# less than top of stack or not
def notGreater(self, i):
    try:
        a = self.precedence[i]
        b = self.precedence[self.peek()]
        return True if a <= b else False
    except KeyError:
        return False

# The main function that converts given infix expression
# to postfix expression
def infixToPostfix(self, exp):

    # Iterate over the expression for conversion
    for i in exp:
        # If the character is an operand,
        # add it to output
        if self.isOperand(i):
            self.output.append(i)

        # If the character is an '(', push it to stack
        elif i == '(':
            self.push(i)

        # If the scanned character is an ')', pop and
        # output from the stack until and '(' is found
        elif i == ')':
            while( (not self.isEmpty()) and self.peek() != '('):
                a = self.pop()
                self.output.append(a)
            if (not self.isEmpty() and self.peek() != '('):
                return -1
                else:
                    self.pop()

            # An operator is encountered
            else:
                while(not self.isEmpty() and self.notGreater(i)):
                    self.output.append(self.pop())
                self.push(i)

        # pop all the operator from the stack
        while not self.isEmpty():
            self.output.append(self.pop())

        result= "".join(self.output)
        print(result)
# Driver program to test above function
exp = "a+b*(c^d-e)^(f+g*h)-i"
obj = Conversion(len(exp))
obj.infixToPostfix(exp)
```

Output:

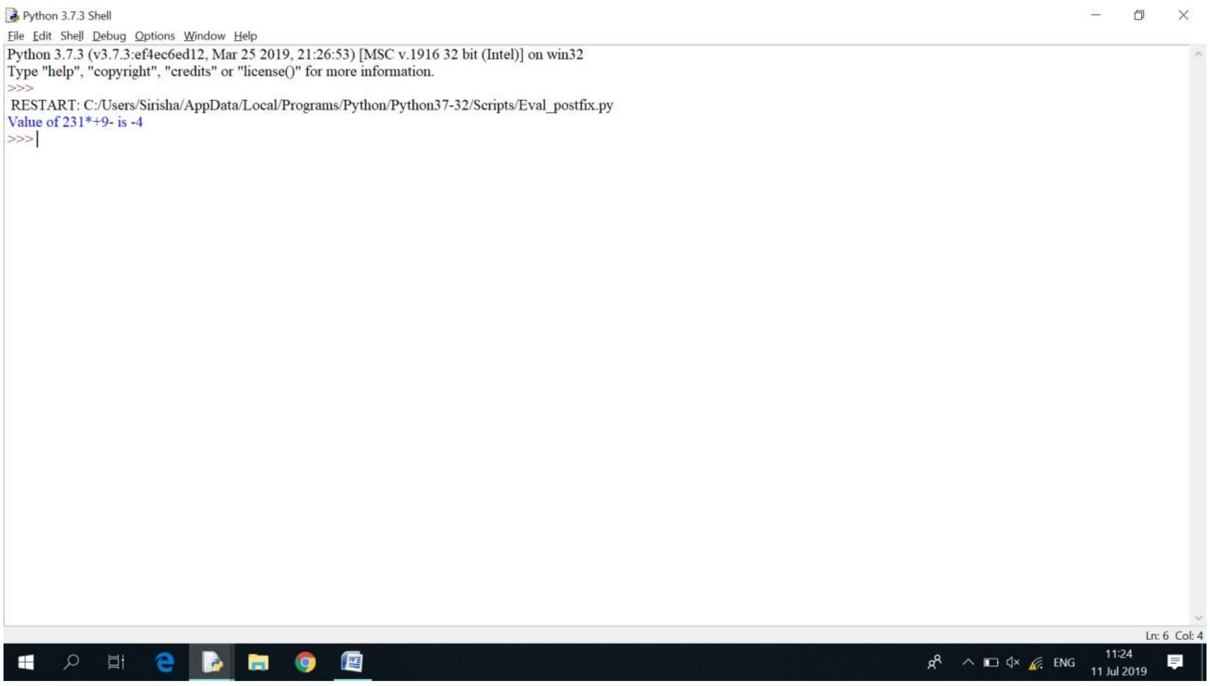


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\infix_to_postfix.py
abcd^e-fgh*+^*+i-
>>>
```

Evaluation of the postfix expression:

```
class Evaluate:

    # Constructor to initialize the class variables
    def_init_(self, capacity):
        self.top = -1
        self.capacity = capacity
        # This array is used a stack
        self.array = []

    # check if the stack is empty
    def isEmpty(self):
        return True if self.top == -1 else False

    # Return the value of the top of the stack
    def peek(self):
        return self.array[-1]

    # Pop the element from the stack
    def pop(self):
        if not self.isEmpty():
            self.top -= 1
            return self.array.pop()
        else:
            return "$"

    # Push the element to the stack
    def push(self, op):
        self.top += 1
        self.array.append(op)


    # The main function that converts given infix expression
    # to postfix expression
    def evaluatePostfix(self, exp):

        # Iterate over the expression for conversion
        for i in exp:

            # If the scanned character is an operand
            # (number here) push it to the stack
            if i.isdigit():
                self.push(i)

            # If the scanned character is an operator,
            # pop two elements from stack and apply it.
            else:
                val1 = self.pop()
                val2 = self.pop()
                self.push(str(eval(val2 + i + val1)))

        return int(self.pop())

# Driver program to test above function
exp = "231*+9-"
obj = Evaluate(len(exp))
print ("Value of {0} is {1}".format(exp, obj.evaluatePostfix(exp)))
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:/Users/Sirisha/AppData/Local/Programs/Python/Python37-32/Scripts/Eval_postfix.py
Value of 231*+9- is -4
>>>
```

PRE-LAB VIVA QUESTIONS:

a. What is an expression?
b. Which operator is having highest priority?
c. Give an example for prefix expression?
d. Give an example for postfix expression?

LAB ASSIGNMENT:

a. Formulate a program to convert infix expression into postfix expression.
b. Write a program to evaluate any postfix expression.
c. Compose a program to convert infix expression into prefix expression.
d. Write a program to convert prefix expression into postfix expression.
e. Write a program to evaluate any prefix expression.

POST-LAB VIVA QUESTIONS:

a. What is the output of the following expression: 2 3 4 5 + * -
b. What is the advantage of postfix expression?
c. What is the maximum difference between number of operators and operands?
d. Which expression doesn’t require parenthesis?
e. What is the output of the following expression: + * - 2 3 4 5

WEEK – 6
IMPLEMENTATION OF SINGLE LINKED LIST

OBJECTIVES:

Write Python programs for the following operations on Single Linked List.

(i) Creation (ii) insertion (iii) deletion (iv) traversal

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Single Linked List: (i) creation (ii) insertion (iii) deletion (iv) traversal

(i) Creation

```
1.  first=new node;{create the 1st node of the list pointed by first};
2.  Read(Data(first));
3.  NEXT(First)=NULL;
4.  Far a First;   [point Far to the First]
5.  For I=1 to N-1 repeat steps 6 to 10
6.  X=new node;
7.  Read(Data(X))
8.  NEXT(X)=NULL;
9.  NEXT(Far)=X; {connect the nodes}
10. Far=X;[shift the pointer to the last node of the list]
11. [end of For Loop]
12. END
```

(ii) Insertion

Empty list case: When list is empty, which is indicated by (head == NULL) condition, the insertion is quite simple. Algorithm sets both head and tail to point to the new node.

Add first: In this case, new node is inserted right before the current head node.

It can be done in two steps:

1. Update the next link of a new node, to point to the current head node.
2. Update head link to point to the new node.

Add last: In this case, new node is inserted right after the current tail node.

It can be done in two steps:

1. Update the next link of the current tail node, to point to the new node
2. Update tail link to point to the new node.

General case: In general case, new node is always inserted between **two nodes, which are already in the list. Head and tail links are not updated in this case.**

Such an insert can be done in two steps:

1. Update link of the "previous" node, to point to the new node.
2. Update link of the new node, to point to the "next" node.

(iii) Deletion

List has only one node: When list has only one node, which is indicated by the condition, that the head points to the same node as the tail, the removal is quite simple. Algorithm disposes the node, pointed by head (or tail) and sets both head and tail to *NULL*.

Remove first: In this case, first node (current head node) is removed from the list.

It can be done in two steps:

1. Update head link to point to the node, next to the head.
2. Dispose removed node.

Remove last: In this case, last node (current tail node) is removed from the list. This operation is a bit trickier, than removing the first node, because algorithm should find a node, which is previous to the tail first.

It can be done in three steps:

1. Update tail link to point to the node, before the tail. In order to find it, list should be traversed first, beginning from the head.
2. Set next link of the new tail to NULL.
3. Dispose removed node.

General case: In general case, node to be removed is always located between **two list nodes. Head and tail links are not updated in this case.**

Such a removal can be done in two steps:

1. Update next link of the previous node, to point to the next node, relative to the removed node.
2. Dispose removed node.

(iv) Traversal

```
1.   If First=NULL then {print “List empty” STOP};
2.   count=0;
3.   ptr=First; {point ptr to the 1st node}
4.   While ptr<> NULL repeat Steps 5 to 6
5.   count=count+1;
6.   ptr=NEXT(ptr) [shift ptr to the next node]
7.   print (‘Number of nodes=’, count)
8.   END
```

PROCEDURE:

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

SOURCE CODE:

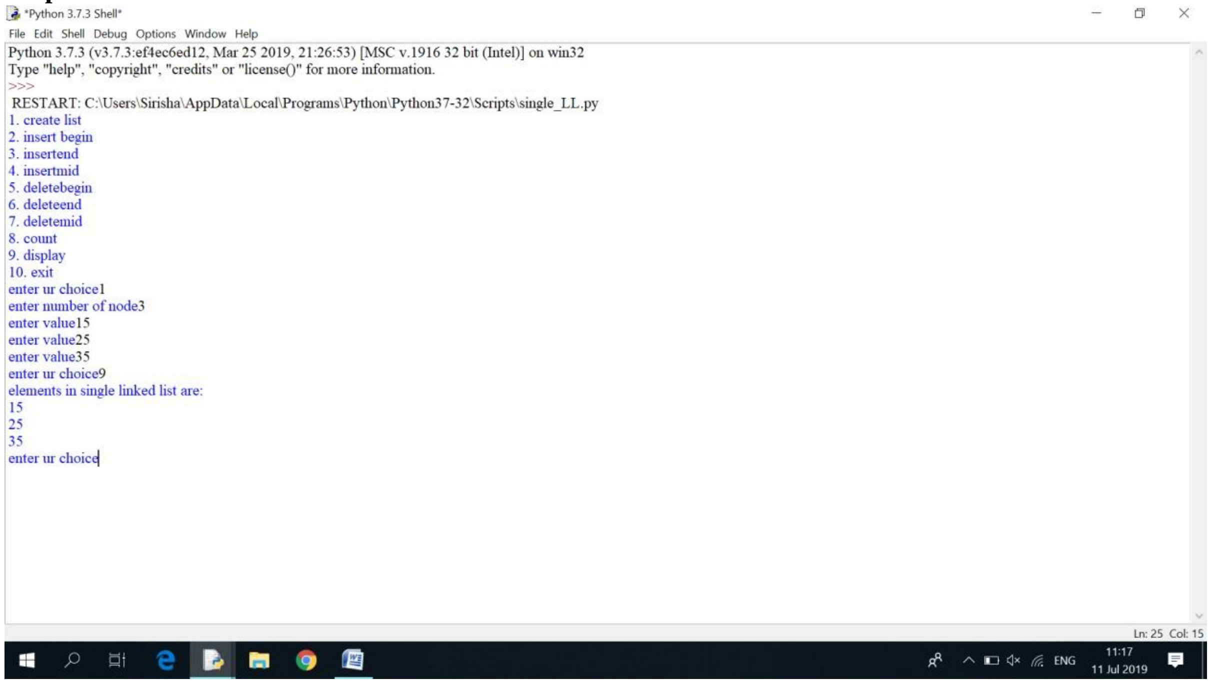
```
class Node:
    def_init_(self,data):
        self.data=data
        self.next=None

class Sll:
    def_init_(self):
        self.start=None
    def createlist(self):
        n=int(input("enter number of node"))
        for i in range(n):
```

```
        data=int(input("enter value"))
        newnode=Node(data)
        if self.start==None:
            self.start=newnode
        else:
            temp=self.start
            while temp.next!=None:
                temp=temp.next
            temp.next=newnode
def insertend(self):
    n=int(input("enter value"))
    newnode=Node(n)
    if self.start==None:
        self.start=newnode
    else:
        temp=self.start
        while temp.next!=None:
            temp=temp.next
        temp.next=newnode
def insertmid(self):
    n=int(input("enter value"))
    newnode=Node(n)
    pos=int(input("enter position"))
    c=self.count()
    if self.start==None:
        self.start=newnode
    else:
        if pos>1 and pos<=c:
            temp=self.start
            prev=temp
            i=1
            while i<pos:
                prev=temp
                temp=temp.next
                i=i+1
        prev.next=newnode
        newnode.next=temp

def count(self):
    nc=0
    temp=self.start
    while temp!=None:
        nc+=1
        temp=temp.next
    print("number of nodes=%d" %nc)
    return nc

def deletemid(self):
    count=1
    if self.start==None:
        print("empty")
    else:
        position=int(input("enter position"))
        c=self.count()
        if position>c:
            print("check position")
        if position>1 and position<c:
            temp=prev=self.start
            while count<position:
                rev=temp
```

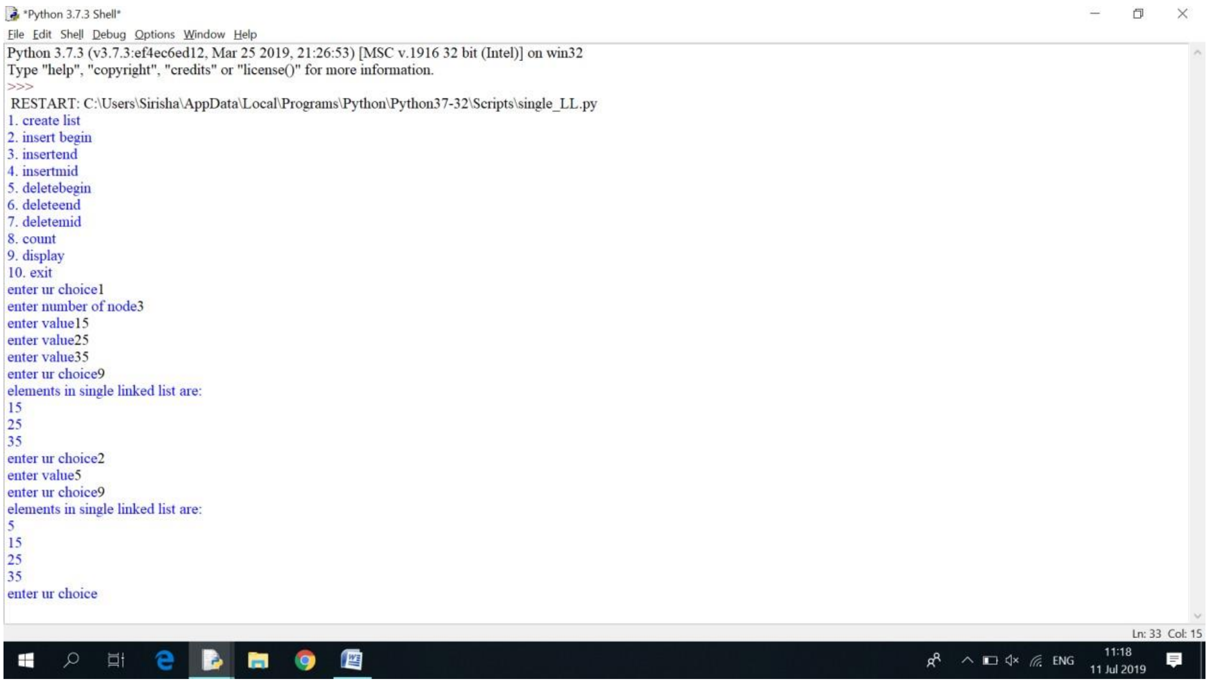
```
                    temp=temp.next
                    count=count+1
                prev.next=temp.next
                del temp
            else:
                print("check position")

    def deleteend(self):
        global prev
        if self.start==None:
            print("empty")
        else:
            temp=self.start
            prev=self.start
            while temp.next!=None:
                prev=temp
                temp=temp.next
            prev.next=None
            del temp

    def insertbegin(self):
        n=int(input("enter value"))
        newnode=Node(n)
        if self.start==None:
            self.start=newnode
        else:
            temp=self.start
            newnode.next=temp
            self.start=newnode

    def deletebegin(self):
        global prev
        if self.start==None:
            print("empty")
        else:
            temp=self.start
            newstart=self.start.next
            del temp
            self.start=newstart

    def display(self):
        print("elements in single linked list are:")
        if self.start==None:
            print("empty")
        else:
            temp=self.start
            print("%d" %(temp.data))
            while temp.next!=None:
                temp=temp.next
                print("%d" %(temp.data))

### OUTSIDE CLASS
def menu():
    print("1. create list \n2. insert begin \n3. insertend \n4. insertmid \n5. deletebegin \n6. deleteend \n7.
deletemid \n8. count \n9. display \n10. exit")

def stop():
    print("u r about to terminate program")
    exit()
```

```
s=Sll()

def default():
    print("check ut input")



menu()
while True:
    menu={
    1: s.createlist,
    2: s.insertbegin,
    3: s.insertend,
    4: s.insertmid,
    5: s.deletebegin,
    6: s.deleteend,
    7: s.deletemid,
    8: s.count,
    9: s.display,
    10: stop}
    option=int(input("enter ur choice"))
    menu.get(option)()
```

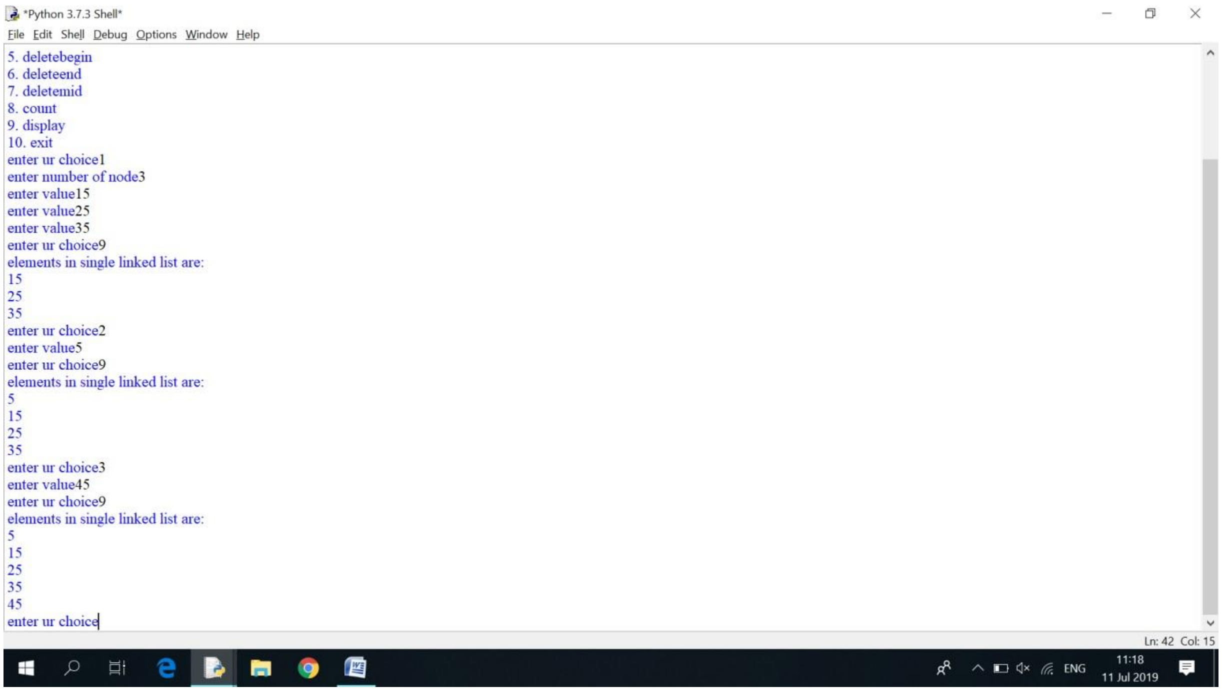
Output:



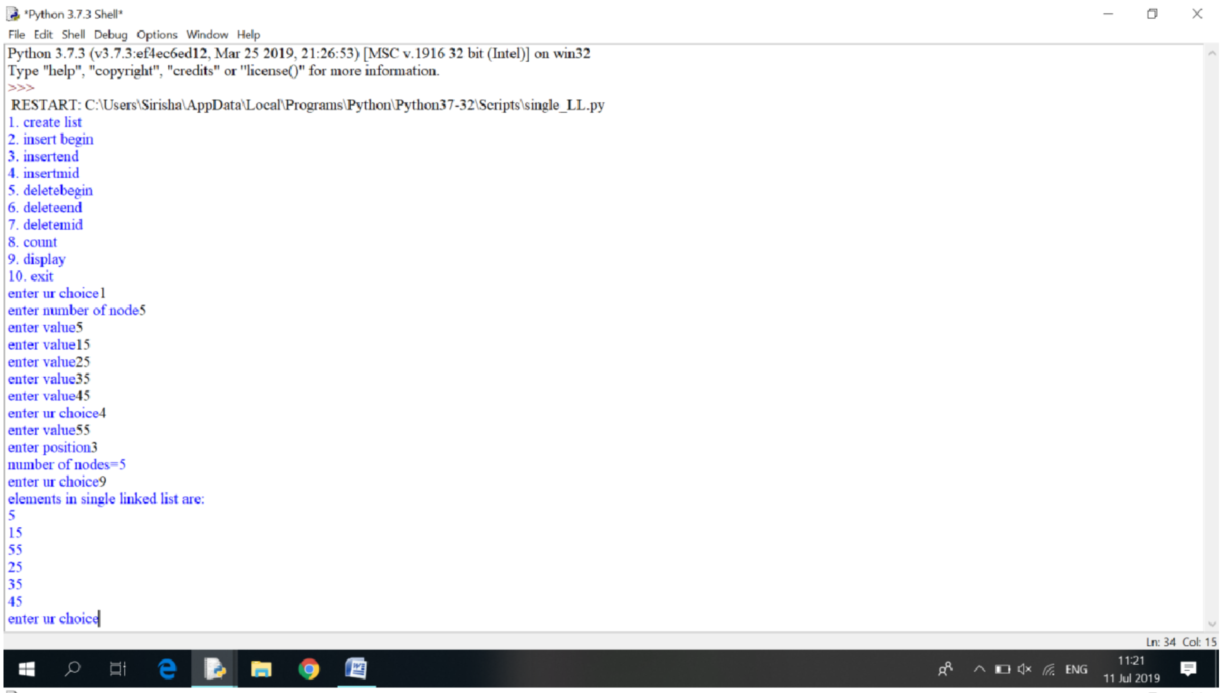
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\single_LL.py
1. create list
2. insert begin
3. insertend
4. insertmid
5. deletebegin
6. deleteend
7. deletemid
8. count
9. display
10. exit
enter ur choice1
enter number of node3
enter value15
enter value25
enter value35
enter ur choice9
elements in single linked list are:
15
25
35
enter ur choice
```



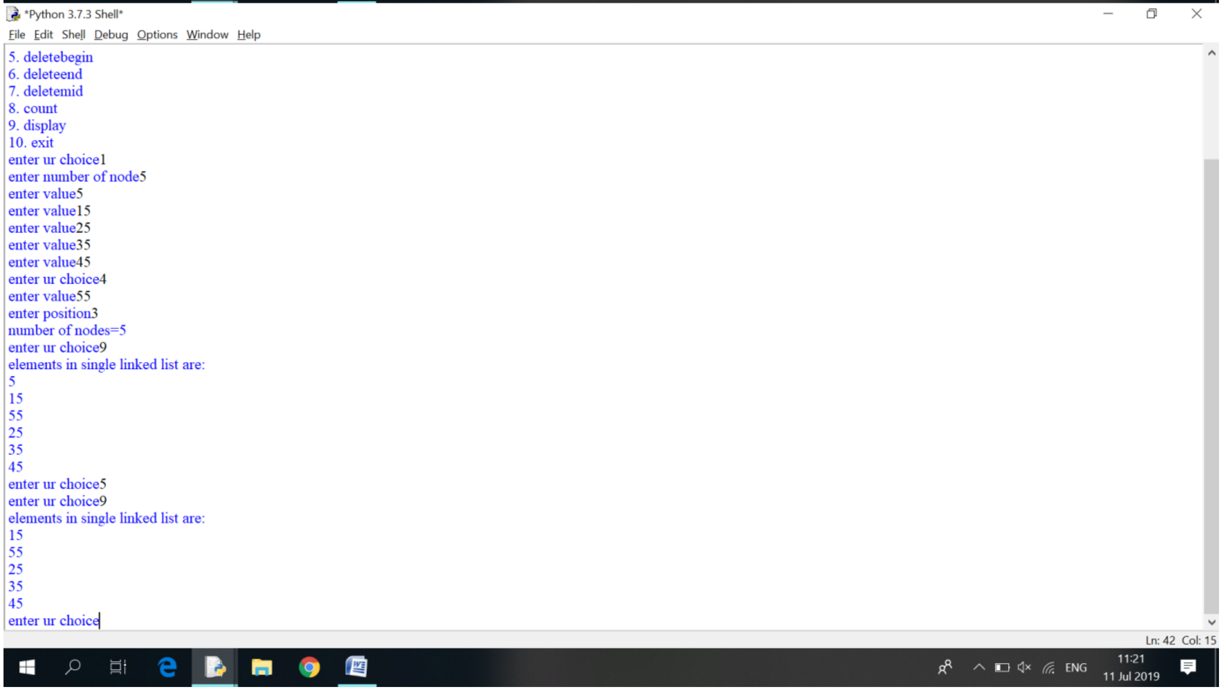
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\single_LL.py
1. create list
2. insert begin
3. insertend
4. insertmid
5. deletebegin
6. deleteend
7. deletemid
8. count
9. display
10. exit
enter ur choice1
enter number of node3
enter value15
enter value25
enter value35
enter ur choice9
elements in single linked list are:
15
25
35
enter ur choice2
enter value5
enter ur choice9
elements in single linked list are:
5
15
25
35
enter ur choice
```



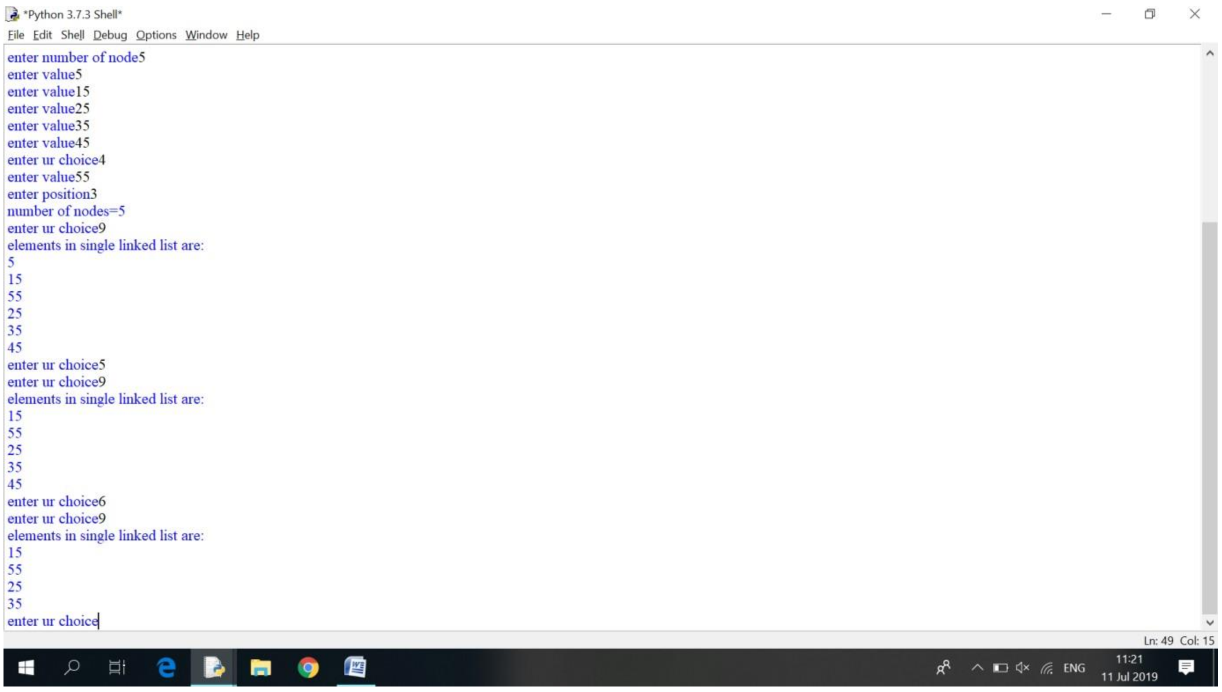
```
5. deletebegin
6. deleteend
7. deletemid
8. count
9. display
10. exit
enter ur choice1
enter number of node3
enter value15
enter value25
enter value35
enter ur choice9
elements in single linked list are:
15
25
35
enter ur choice2
enter value5
enter ur choice9
elements in single linked list are:
5
15
25
35
enter ur choice3
enter value45
enter ur choice9
elements in single linked list are:
5
15
25
35
45
enter ur choice
```



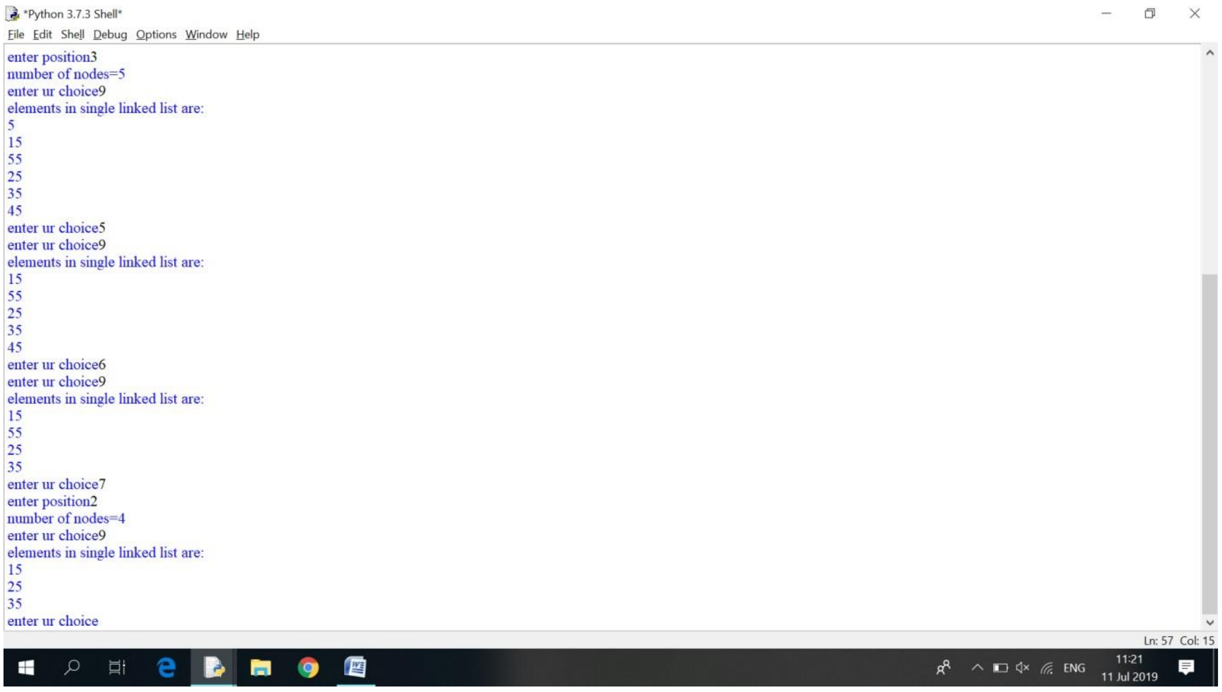
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\single_LL.py
1. create list
2. insert begin
3. insertend
4. insertmid
5. deletebegin
6. deleteend
7. deletemid
8. count
9. display
10. exit
enter ur choice1
enter number of node5
enter value5
enter value15
enter value25
enter value35
enter value45
enter ur choice4
enter value55
enter position3
number of nodes=5
enter ur choice9
elements in single linked list are:
5
15
55
25
35
45
enter ur choice
```



```
5. deletebegin
6. deleteend
7. deletemid
8. count
9. display
10. exit
enter ur choice1
enter number of node5
enter value5
enter value15
enter value25
enter value35
enter value45
enter ur choice4
enter value55
enter position3
number of nodes=5
enter ur choice9
elements in single linked list are:
5
15
55
25
35
45
enter ur choice5
enter ur choice9
elements in single linked list are:
15
55
25
35
45
enter ur choice
```



```
enter number of node5
enter value5
enter value15
enter value25
enter value35
enter value45
enter ur choice4
enter value55
enter position3
number of nodes=5
enter ur choice9
elements in single linked list are:
5
15
55
25
35
45
enter ur choice5
enter ur choice9
elements in single linked list are:
15
55
25
35
45
enter ur choice6
enter ur choice9
elements in single linked list are:
15
55
25
35
enter ur choice
```



```
enter position3
number of nodes=5
enter ur choice9
elements in single linked list are:
5
15
55
25
35
45
enter ur choice5
enter ur choice9
elements in single linked list are:
15
55
25
35
45
enter ur choice6
enter ur choice9
elements in single linked list are:
15
55
25
35
enter ur choice7
enter position2
number of nodes=4
enter ur choice9
elements in single linked list are:
15
25
35
enter ur choice
```

PRE-LAB VIVA QUESTIONS:

a. What is linked list?
b. What type of memory allocation is used in linked list?
c. How many self referential pointers are used in single linked list?
d. What is double linked list?
e. Which node contains NULL pointer in a single linked list?
f. How many nodes you can have in a single linked list?
g. What are the components of a polynomial expression?

LAB ASSIGNMENT:

a. Formulate a program to create a singly linked list and perform insertion, deletion and traversing operations on a singly linked list.
b. Write a program to merge two linked list?
c. Compose a program to print odd nodes of a linked list?

d. Write a program to divide the linked list into two parts into odd and even list?
e. Formulate a program to convert a single linked to circular linked list?
f. Compose a program to store and add two polynomial expressions in memory using linked list.

POST-LAB VIVAQUESTIONS:

a. What is the time complexity to insert a node at the beginning of linked list?
b. What is the time complexity to traverse a linked list?
c. How many modifications are required to delete a node at the beginning?
d. How many modifications are required to insert a node in the middle of the linked list?
e. What are the types of linked list?
f. What are the applications of a linked list?

WEEK – 7
IMPLEMENTATION OF CIRCULAR LINKED LIST

OBJECTIVE:

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

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Circular Linked List:

(i) Creation

```
Init_circular_linked_list(key)
      z= new node
      z.data=key
      z.next=z
      c=new circular_linked_list
      c.last=z
      return c
```

(ii) Insertion

```
Insert_after(n,a)
      n.next=a.next
      a.next=n
insert_at_last(L,n)
      n.next=L.last.next
      L.last.next=n
      L.last=n
```

(iii) Deletion

```
Delete(L,n)
      temp=L.last
      while temp.next!=n
            temp=temp.next
      if n==L.last
            if n.next==n
                  L.last=NULL
            else
                  temp.next=n.next
                  L.last=temp
      else
            temp.next=n.next
```

(iv) Traversal

```
Node temp = this.last;
 print temp.data
 temp = temp.next;

 while(temp != this.last) {
  print temp.data
  temp = temp.next;
```

PROCEDURE:

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

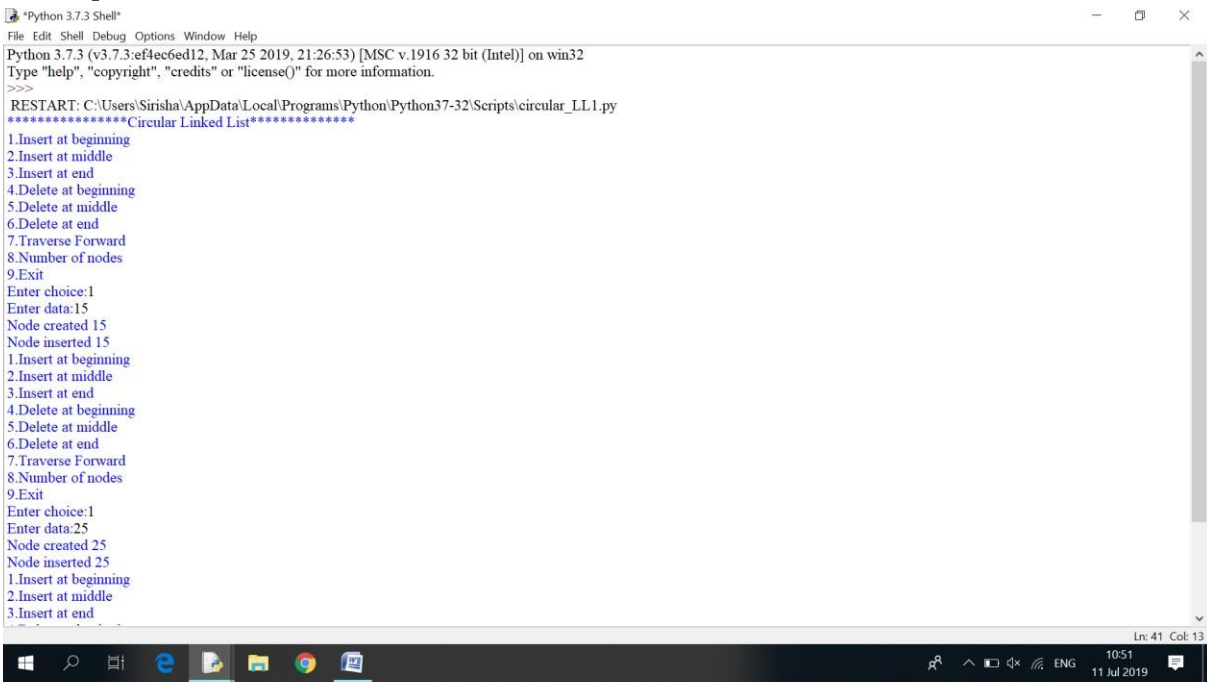
SOURCE CODE:

```
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:
```

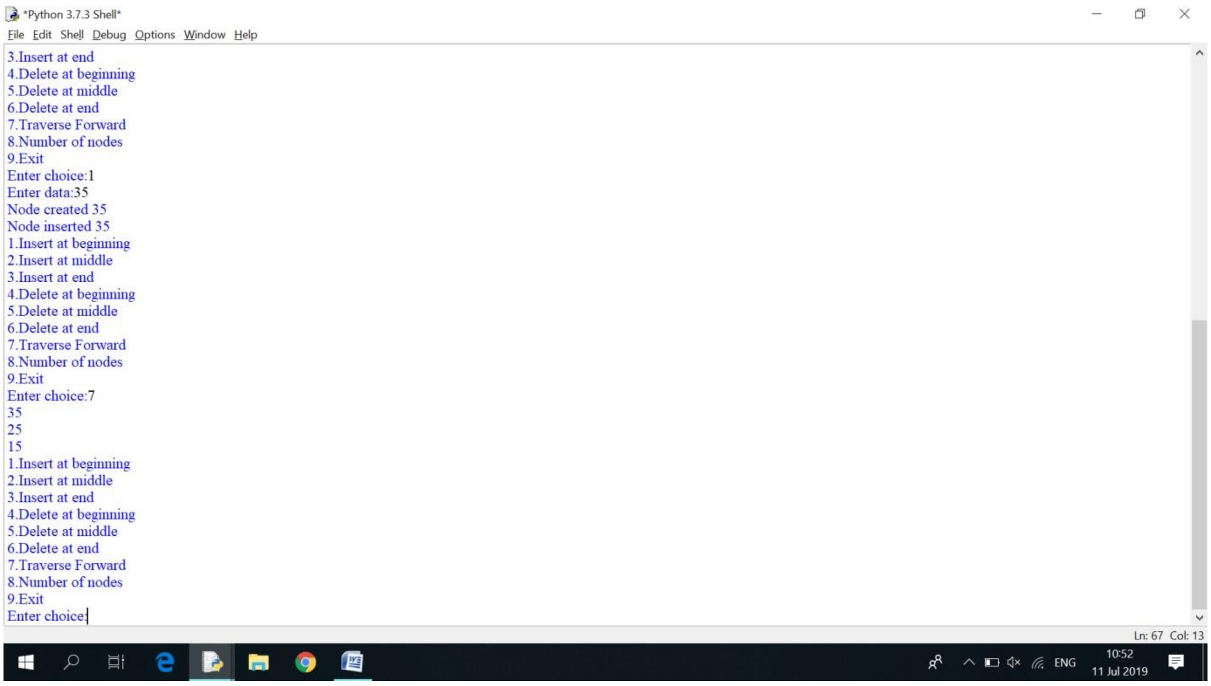
```
                temp=temp.next
                i+=1
            node.next=temp.next
            temp.next=node
            self.ctr+=1
            print("Node Insered",data)
        return
    def delete_beg(self):
        if self.head==None:
            print("No Nodes exist")
        elif self.ctr==1:
            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
```

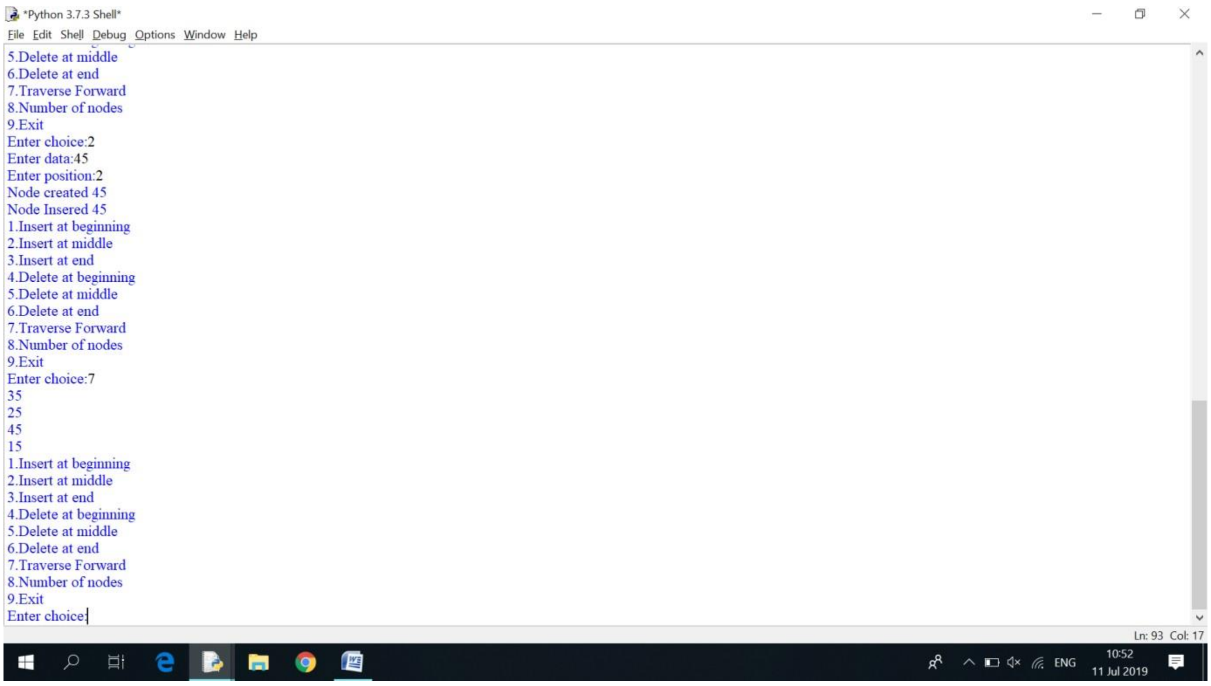
Output:



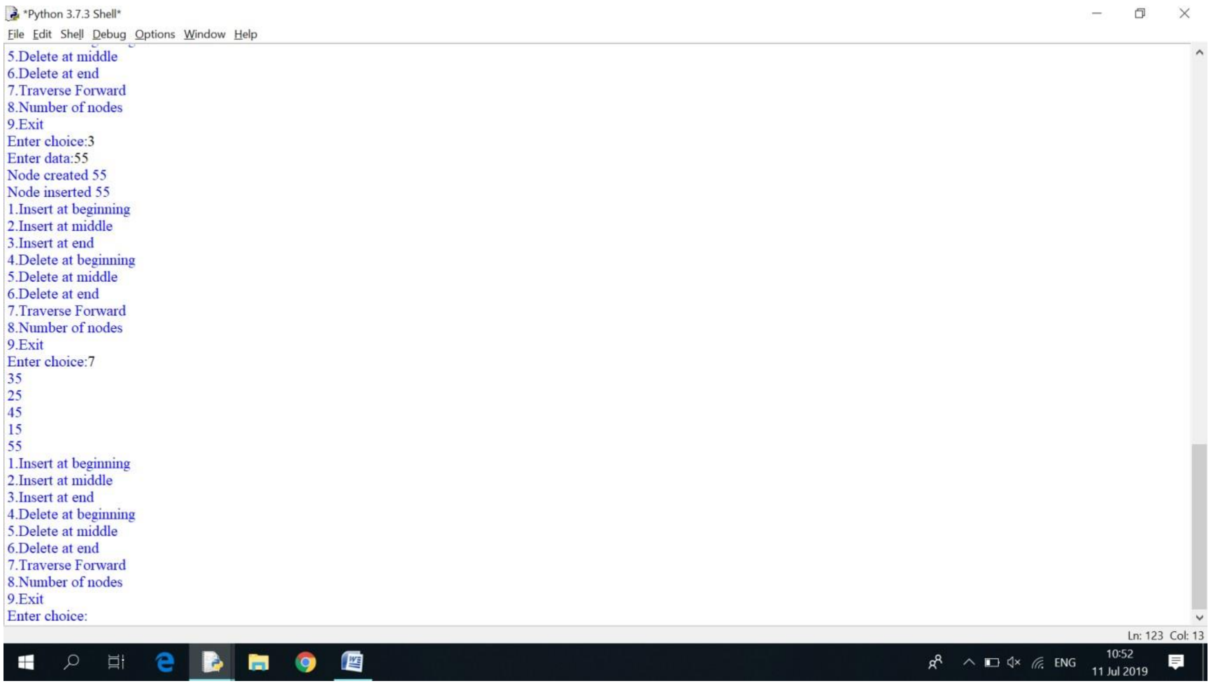
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\circular_LL1.py
****************Circular Linked List**************
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:15
Node created 15
Node inserted 15
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:25
Node created 25
Node inserted 25
1.Insert at beginning
2.Insert at middle
3.Insert at end
```



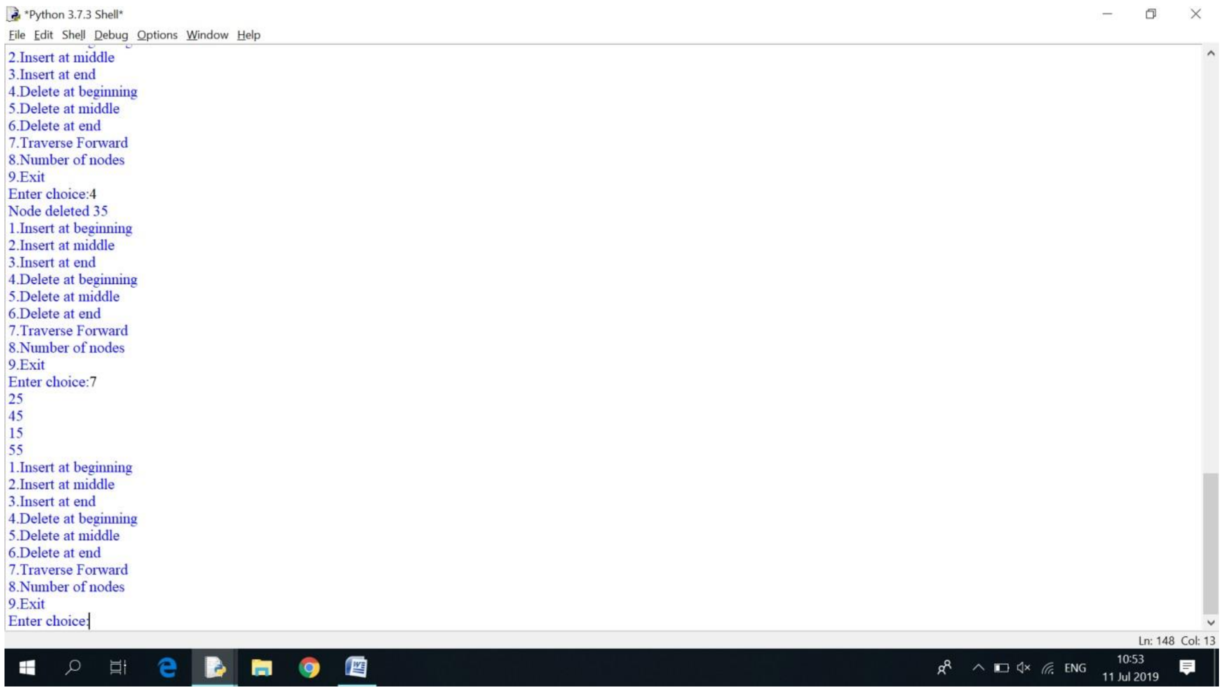
```
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:1
Enter data:35
Node created 35
Node inserted 35
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:7
35
25
15
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.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:2
Enter data:45
Enter position:2
Node created 45
Node Insered 45
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
35
25
45
15
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:
```



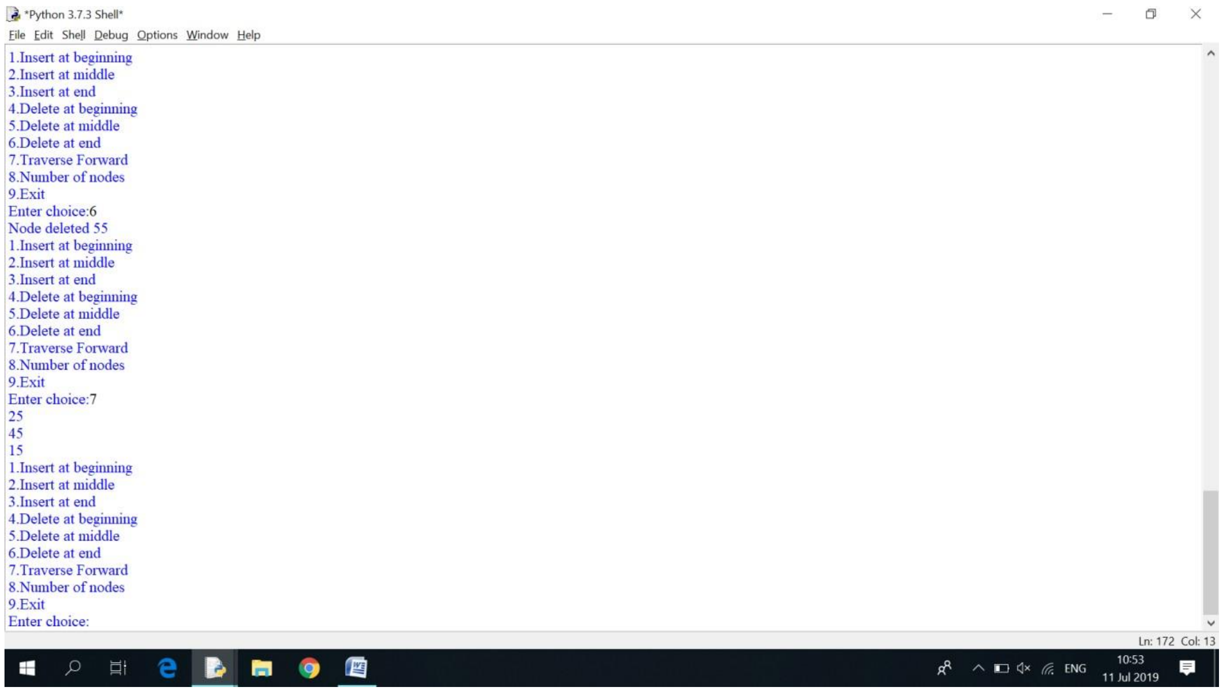
```
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:3
Enter data:55
Node created 55
Node inserted 55
1.Insert at beginning
2.Insert at middle
3.Insert at end
4.Delete at beginning
5.Delete at middle
6.Delete at end
7.Traverse Forward
8.Number of nodes
9.Exit
Enter choice:7
35
25
45
15
55
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:
```


Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
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 35
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
25
45
15
55
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:
```

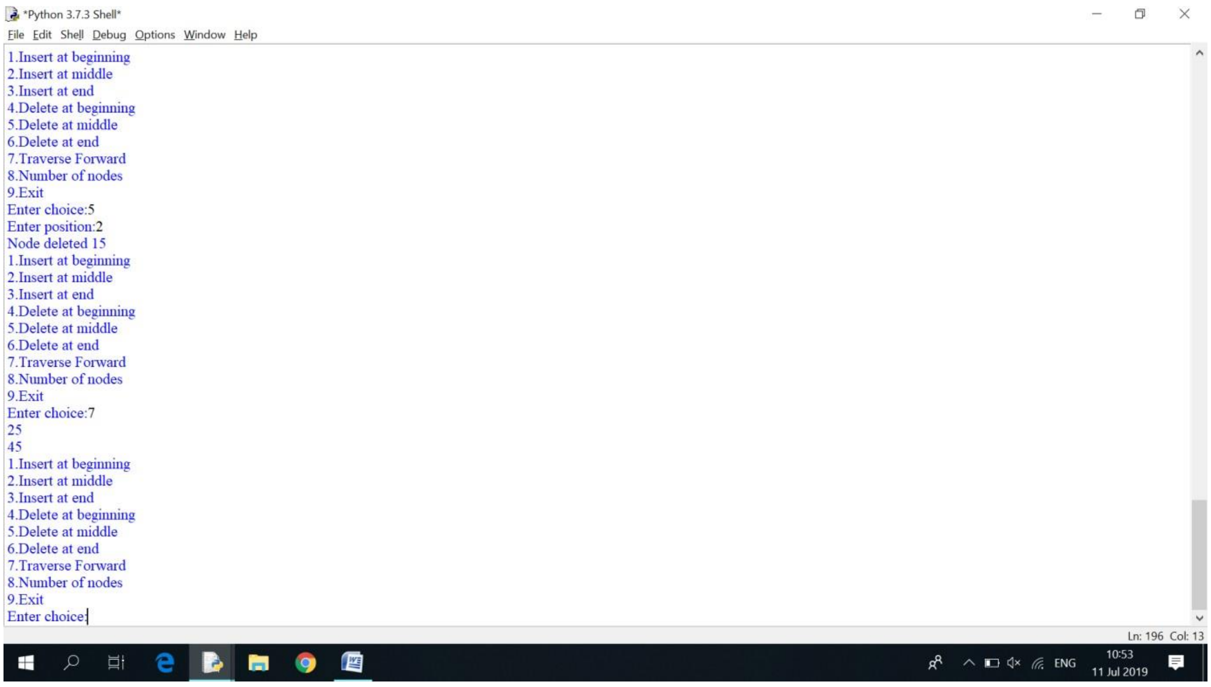
Ln: 148 Col: 13


Python 3.7.3 Shell

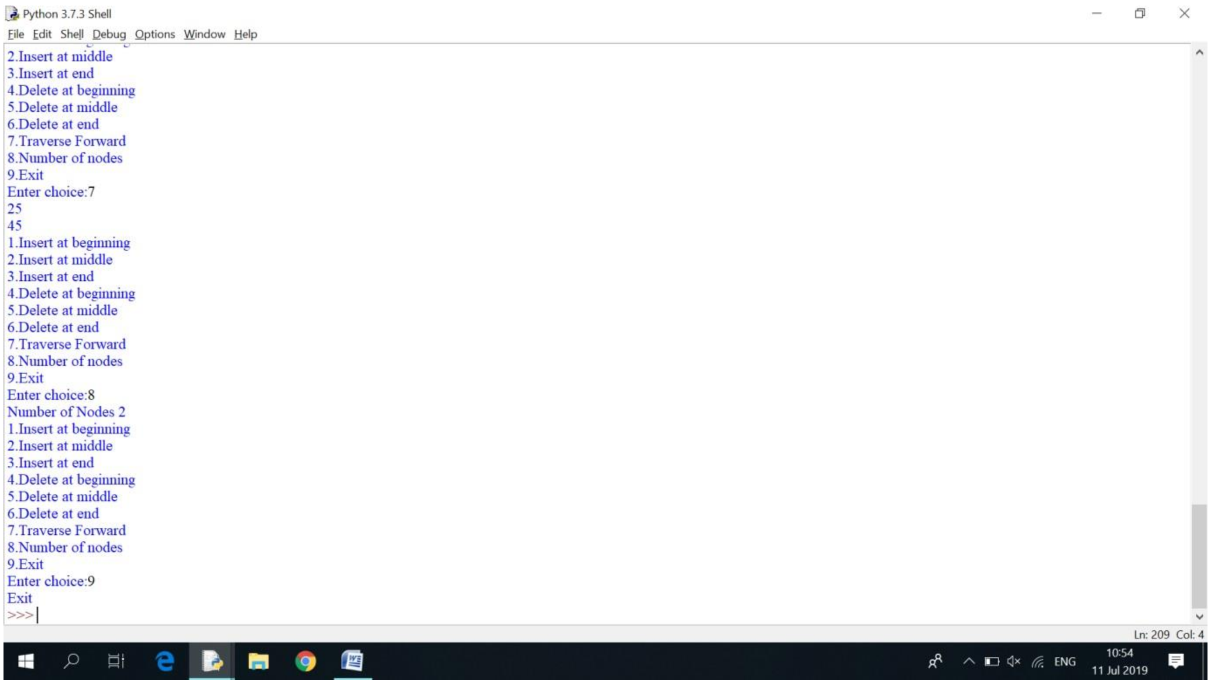
File Edit Shell Debug Options Window Help

```
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 55
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
25
45
15
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:
```

Ln: 172 Col: 13



```
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:2
Node deleted 15
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
25
45
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.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
25
45
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 2
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
>>>
```

PRE-LAB VIVA QUESTIONS:

a. What is circular linked list?
b. What type of memory allocation is used in linked circular list?
c. How many self referential pointers are used in circular single linked list?
d. What is double linked list?
e. Which node contains NULL pointer in a circular single linked list?
f. How many nodes you can have in a circular single linked list?

LAB ASSIGNMENT:

a. Formulate a program to create a circular singly linked list and perform insertion, deletion and traversing operations on a singly linked list.
b. Write a program to merge two linked list?

c. Compose a program to print odd nodes of a circular linked list?
d. Write a program to divide the circular linked list into two parts into odd and even list?
e. Formulate a program to convert a single linked to circular linked list?

POST-LAB VIVA QUESTIONS:

a. What is the time complexity to insert a node at the beginning of circular linked list?
b. What is the time complexity to traverse a circular linked list?
c. How many modifications are required to delete a node at the beginning?
d. How many modifications are required to insert a node in the middle of the circular linked list?
e. What are the types of linked list?
f. What are the applications of a circular linked list?

WEEK – 8
IMPLEMENATION OF DOUBLE LIKED LIST

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

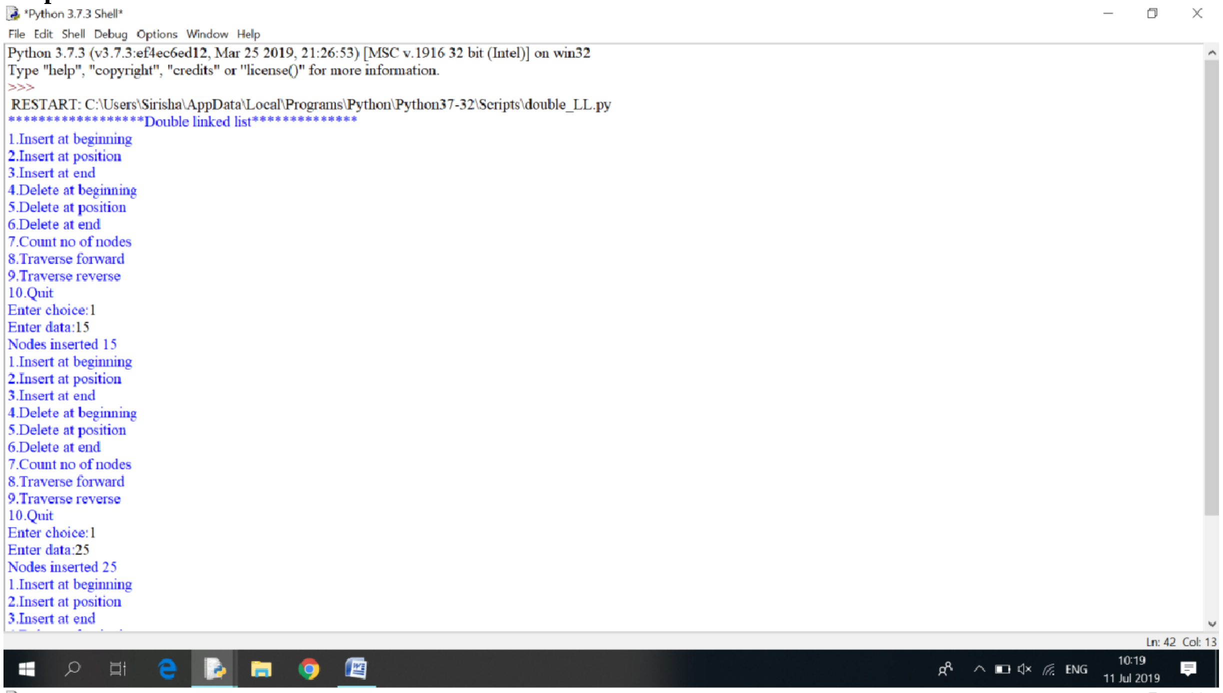
RESOURCE:
Python 3.7.3

PROGRAM LOGIC:
Double Linked List
(i) Creation
(ii) Insertion
(iii) Deletion
(iv) Traversal in both ways

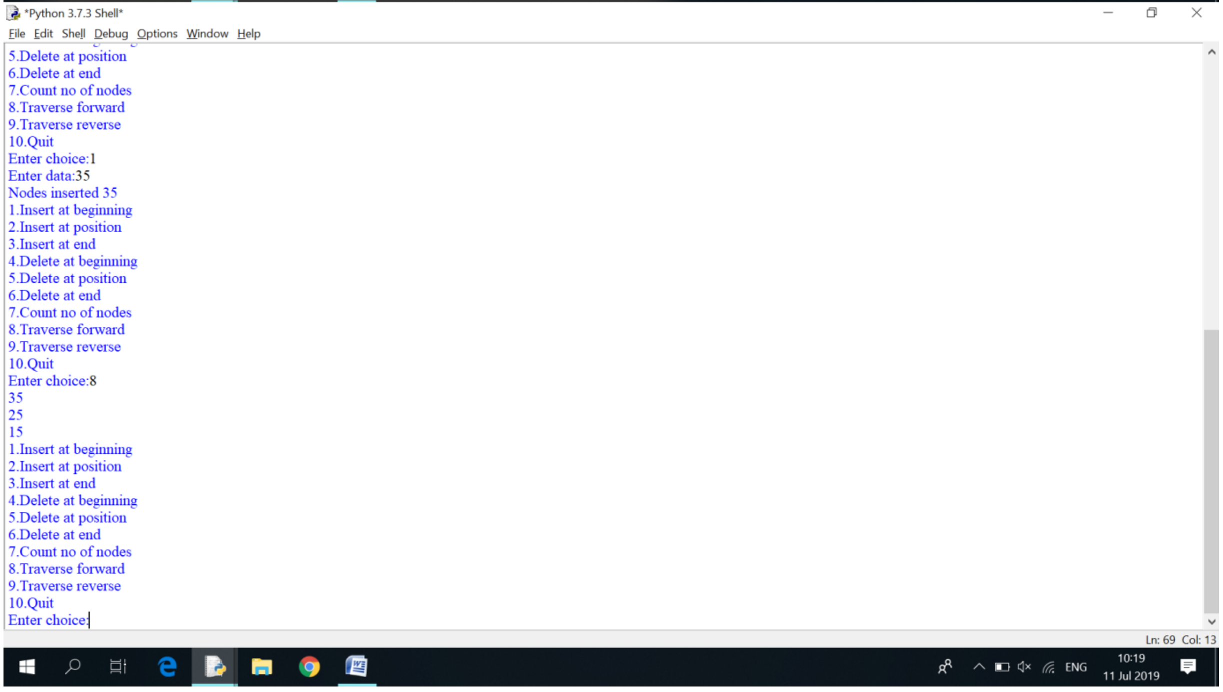
PROCEDURE:
a. Create : Open a new file in Python shell, write a program and save the program with .py extension.
b. Execute : Go to Run -> Run module (F5)

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
```

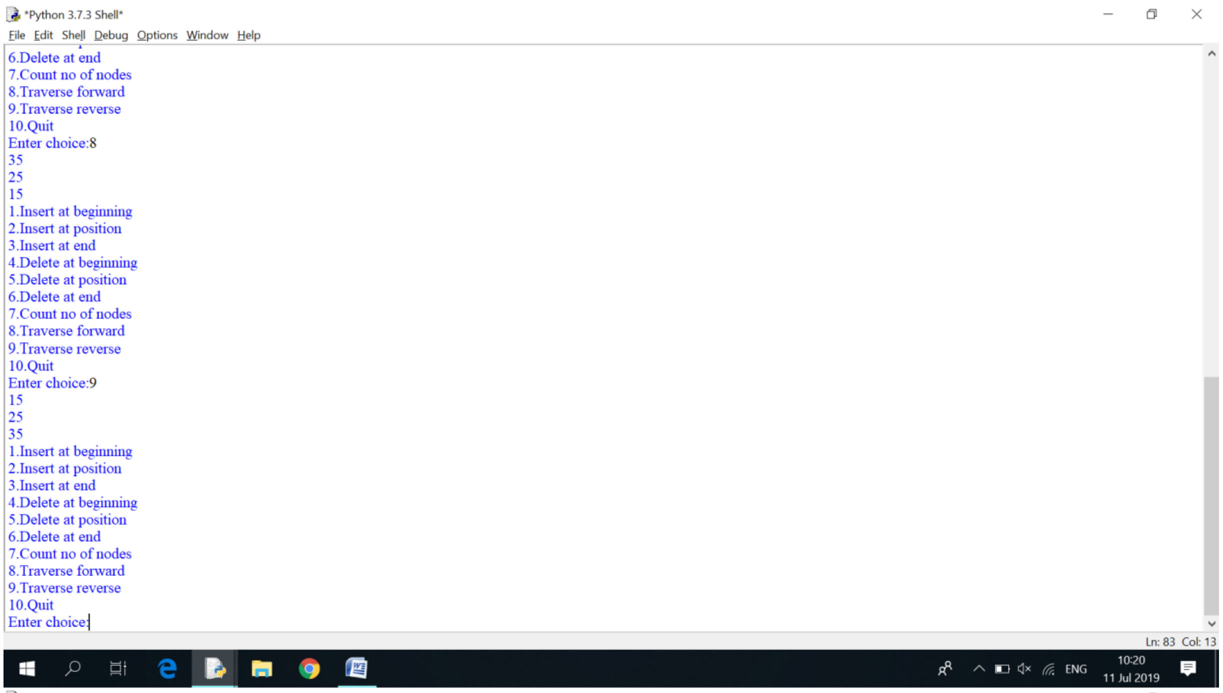
Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\double_LL.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:15
Nodes inserted 15
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:25
Nodes inserted 25
1.Insert at beginning
2.Insert at position
3.Insert at end
```



```
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:35
Nodes inserted 35
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
35
25
15
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:
```

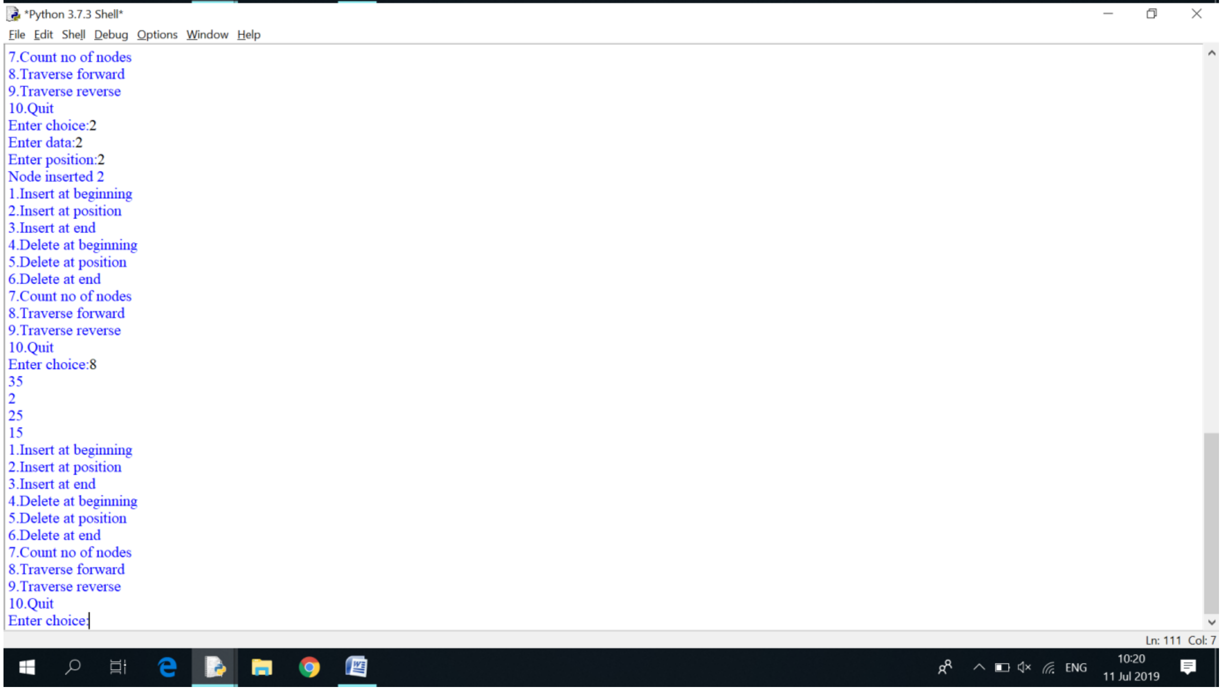


Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
6.Delete at end
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:8
35
25
15
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
15
25
35
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:
```

Ln: 83 Col: 13

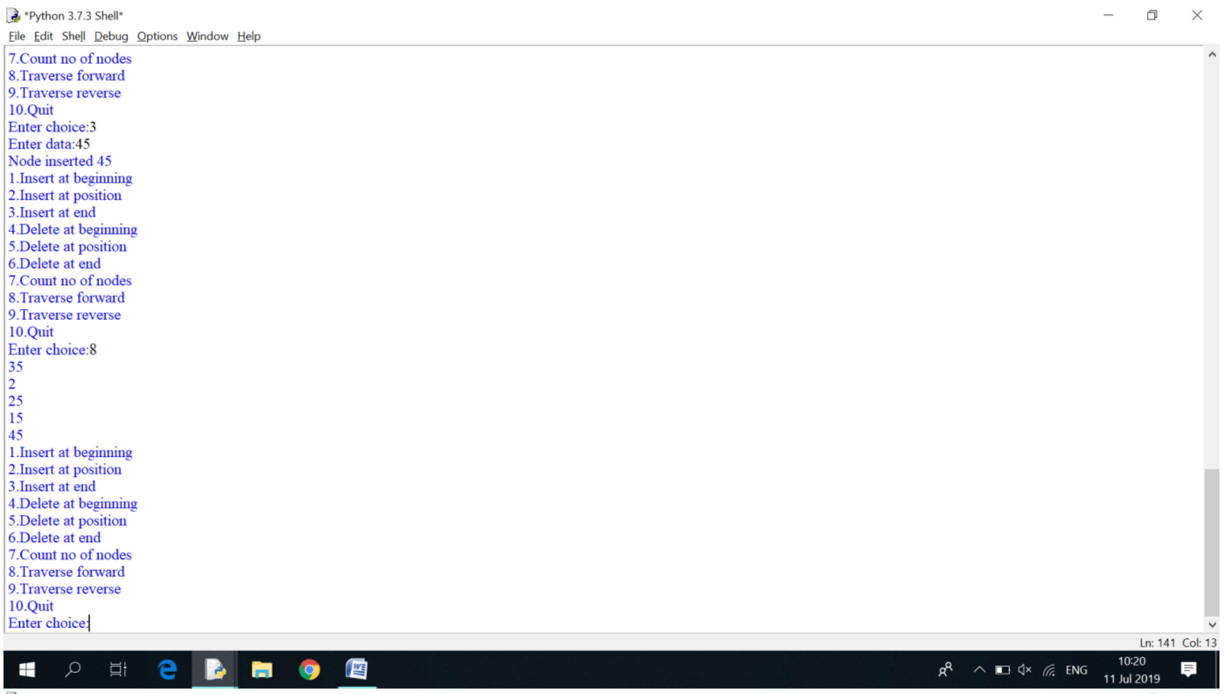


Python 3.7.3 Shell

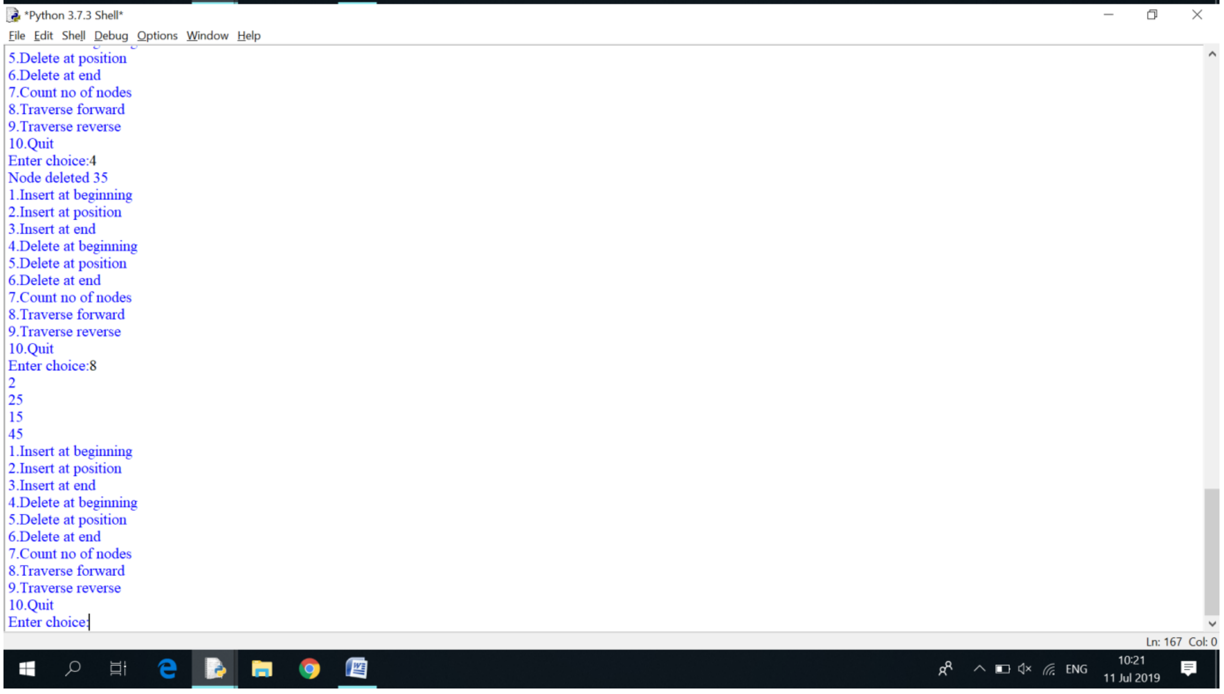
File Edit Shell Debug Options Window Help

```
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:2
Enter data:2
Enter position:2
Node inserted 2
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
35
2
25
15
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:
```

Ln: 111 Col: 7



```
7.Count no of nodes
8.Traverse forward
9.Traverse reverse
10.Quit
Enter choice:3
Enter data:45
Node inserted 45
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
35
2
25
15
45
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.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 35
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
2
25
15
45
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:
```

PRE-LAB VIVA QUESTIONS:

a. What is double linked list
b. How to represent a node in double linked list
c. Differentiate between single and double linked list

LAB ASSIGNMENT:

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

POST-LAB VIVA QUESTIONS:

a. How to represent double linked list?

b. How will you traverse double linked list?
c. List the advantages of double linked list over single list?

WEEK – 9
IMPLEMENTATION OF STACK USING LINKED LIST

OBJECTIVE:
Write a Python script to implement Stack using linked list.

RESOURCE:
Python 3.7.3

PROGRAM LOGIC:

create()

```
Define a 'Node' structure with two members data and next.
Define a Node pointer 'top' and set it to NULL.
Implement the main method by displaying Menu with list of operations and make suitable function calls
in the main method.
```

push(value) - Inserting an element into the Stack

```
Create a newNode with given value.
Check whether stack is Empty (top == NULL)
If it is Empty, then set newNode → next = NULL.
If it is Not Empty, then set newNode → next = top.
Finally, set top = newNode.
```

pop() - Deleting an Element from a Stack

```
Check whether stack is Empty (top == NULL).
If it is Empty, then display "Stack is Empty!!! Deletion is not possible!!!" and terminate the function
If it is Not Empty, then define a Node pointer 'temp' and set it to 'top'.
Then set 'top = top → next'.
Finally, delete 'temp'. (free(temp)).
```

display() - Displaying stack of elements

```
Check whether stack is Empty (top == NULL).
If it is Empty, then display 'Stack is Empty!!!' and terminate the function.
If it is Not Empty, then define a Node pointer 'temp' and initialize with top.
Display 'temp → data --->' and move it to the next node. Repeat the same until temp reaches to the first
node in the stack. (temp → next != NULL).
Finally! Display 'temp → data ---> NULL'.
```

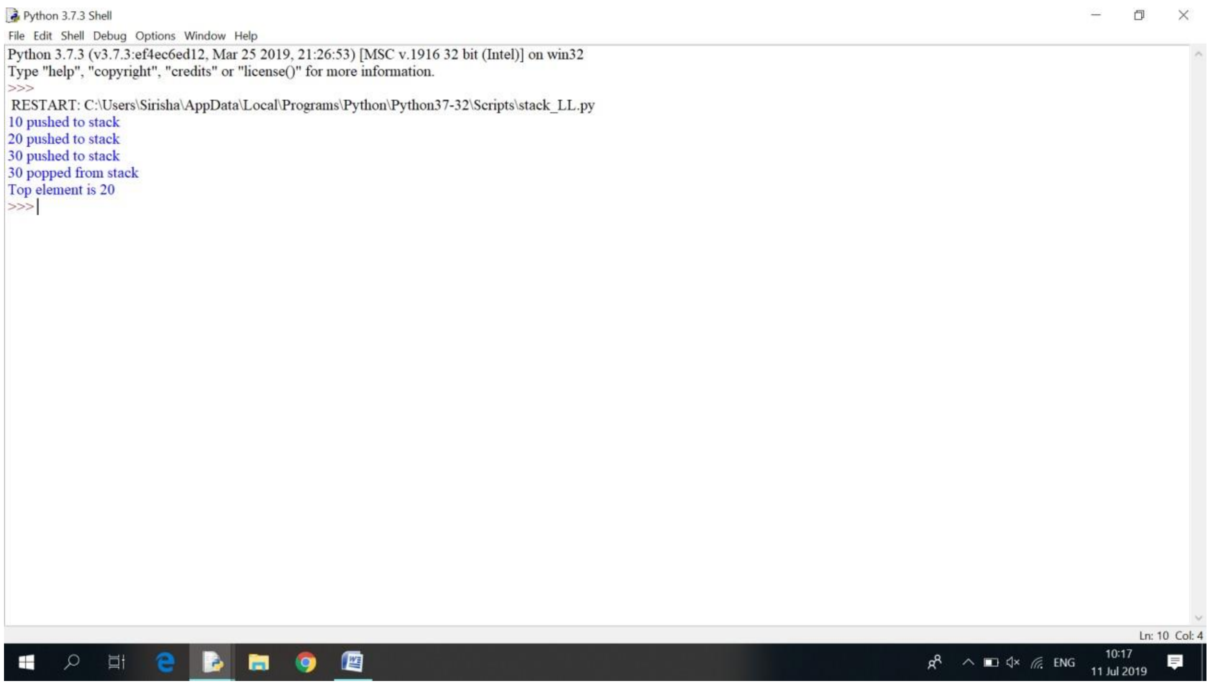
PROCEDURE:

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

SOURCE CODE:

```
class StackNode:

    # Constructor to initialize a node
    def_init_(self, data):
        self.data = data
        self.next = None

class Stack:

    # Constructor to initialize the root of linked list
    def_init_(self):
        self.root = None
    def isEmpty(self):
        return True if self.root is None else False

    def push(self, data):
        newNode = StackNode(data)
        newNode.next = self.root
        self.root = newNode
        print ("%d pushed to stack" %(data))

    def pop(self):
        if (self.isEmpty()):
            return float("-inf")
        temp = self.root
        self.root = self.root.next
        popped = temp.data
        return popped

    def peek(self):
        if self.isEmpty():
            return float("-inf")
        return self.root.data

# Driver program to test above class
stack = Stack()
stack.push(10)
stack.push(20)
stack.push(30)

print ("%d popped from stack" %(stack.pop()))
print ("Top element is %d " %(stack.peek()))
```

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\stack_LL.py
10 pushed to stack
20 pushed to stack
30 pushed to stack
30 popped from stack
Top element is 20
>>>
```

PRE-LAB VIVA QUESTIONS:

a. What do you mean by stack overflow?
b. What are the basic operations of a stack?
c. How to implement stack?

LAB ASSIGNMENT:

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

POST-LAB VIVA QUESTIONS:

a. How to remove an element from stack?
b. How to insert an element using a stack?
c. Is it possible to store any number of data elements in stack?
d. What are the demerits of stack?

WEEK – 10
IMPLEMENTATION OF QUEUE USING LINKED LIST

OBJECTIVE:

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

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Queue using linked list:

Create():

Define a 'Node' structure with two members data and next.
Define two Node pointers 'front' and 'rear' and set both to NULL.
Implement the main method by displaying Menu of list of operations and make suitable function calls in the main method to perform user selected operation.

enQueue(value) - Inserting an element into the Queue

Create a newNode with given value and set 'newNode → next' to NULL.
Check whether queue is Empty (rear == NULL)
If it is Empty then, set front = newNode and rear = newNode.
If it is Not Empty then, set rear → next = newNode and rear = newNode.

deQueue() - Deleting an Element from Queue

Check whether queue is Empty (front == NULL).
If it is Empty, then display "Queue is Empty!!! Deletion is not possible!!!" and terminate from the function
If it is Not Empty then, define a Node pointer 'temp' and set it to 'front'.
Then set 'front = front → next' and delete 'temp' (free(temp)).

display() - Displaying the elements of Queue

Check whether queue is Empty (front == NULL).
If it is Empty then, display 'Queue is Empty!!!' and terminate the function.
If it is Not Empty then, define a Node pointer 'temp' and initialize with front.
Display 'temp → data --->' and move it to the next node. Repeat the same until 'temp' reaches to 'rear' (temp → next != NULL).
Finally! Display 'temp → data ---> NULL'.

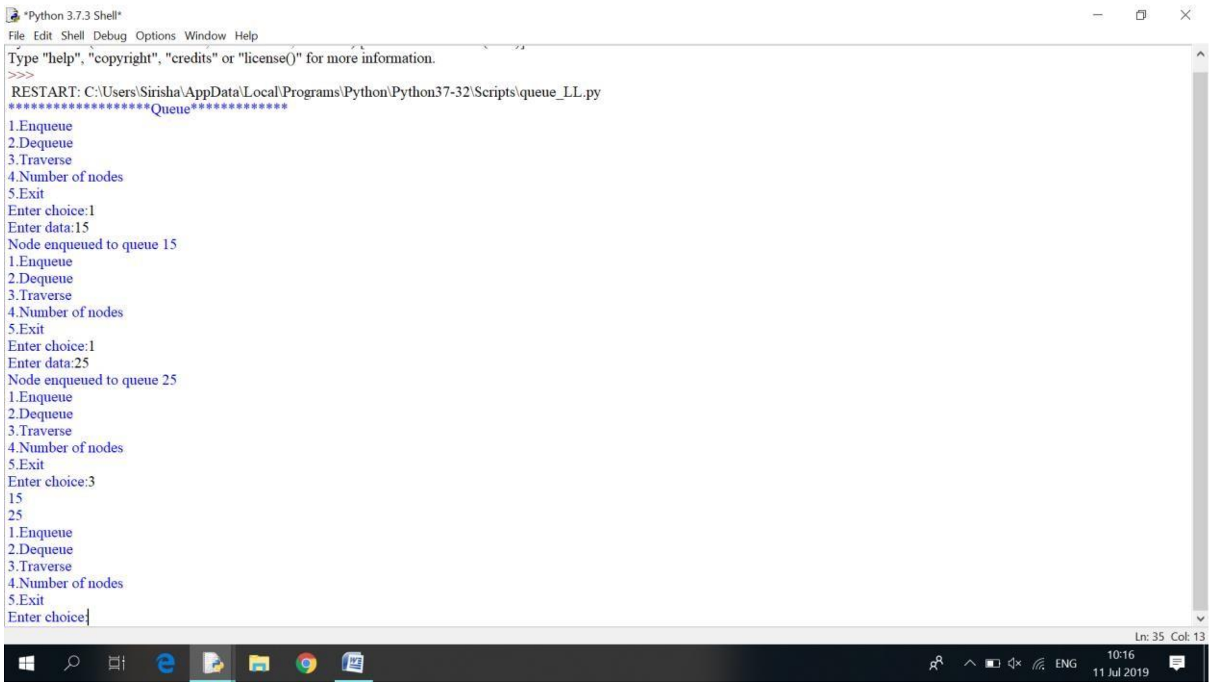
PROCEDURE:

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

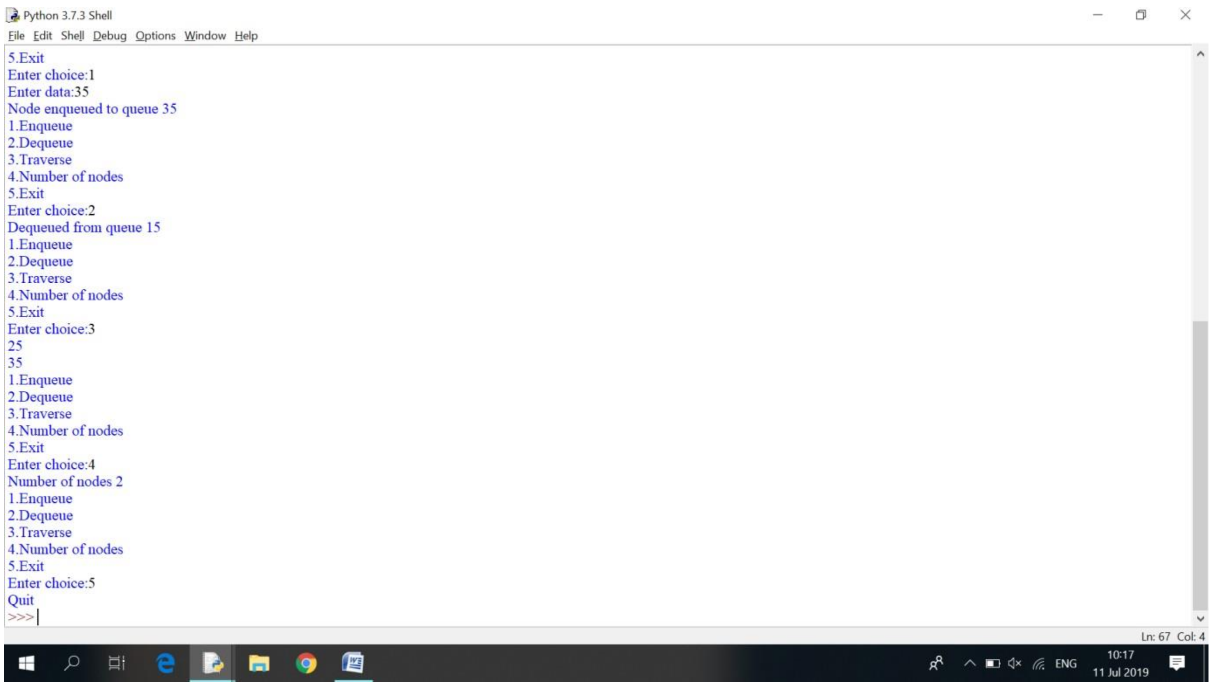
SOURCE CODE:

```
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
```

Output:



```
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\queue_LL.py
*******************Queue*************
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:15
Node enqueued to queue 15
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:1
Enter data:25
Node enqueued to queue 25
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:3
15
25
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:
```



```
5.Exit
Enter choice:1
Enter data:35
Node enqueued to queue 35
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:2
Dequeued from queue 15
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:3
25
35
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:4
Number of nodes 2
1.Enqueue
2.Dequeue
3.Traverse
4.Number of nodes
5.Exit
Enter choice:5
Quit
>>>
```

PRE-LAB VIVA QUESTIONS:

a. Which principle is followed in queue?
b. What are the applications of queue?

LAB ASSIGNMENT:

a. Write a program to implement Queue operations using linked list.
b. Formulate a program to implement circular queue operations using arrays.
c. Write a program to implement a priority queue?

POST-LAB VIVA QUESTIONS:

a. What is the advantage of circular queue over linear queue?
b. Where priority queues are used?
c. What is DEQUE?

WEEK – 11
IMPLEMENTATION OF QUEUE USING LINKED LIST

OBJECTIVE:

a. Write a Python script to implement depth first search
b. Write a Python script to implement breadth first search

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Depth first search

1. Define a Stack of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and push it on to the Stack.
3. Visit any one of the non-visited adjacent vertices of a vertex which is at the top of stack and push it on to the stack.
4. Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
5. When there is no new vertex to visit then use back tracking and pop one vertex from the stack.
6. Repeat steps 3, 4 and 5 until stack becomes Empty.
7. When stack becomes Empty, then produce final spanning tree by removing unused edges from the graph

Breadth first search

1. Define a Queue of size total number of vertices in the graph.
2. Select any vertex as starting point for traversal. Visit that vertex and insert it into the Queue.
3. Visit all the non-visited adjacent vertices of the vertex which is at front of the Queue and insert them into the Queue.
4. When there is no new vertex to be visited from the vertex which is at front of the Queue then delete that vertex.
5. Repeat steps 3 and 4 until queue becomes empty.
6. When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

PROCEDURE:

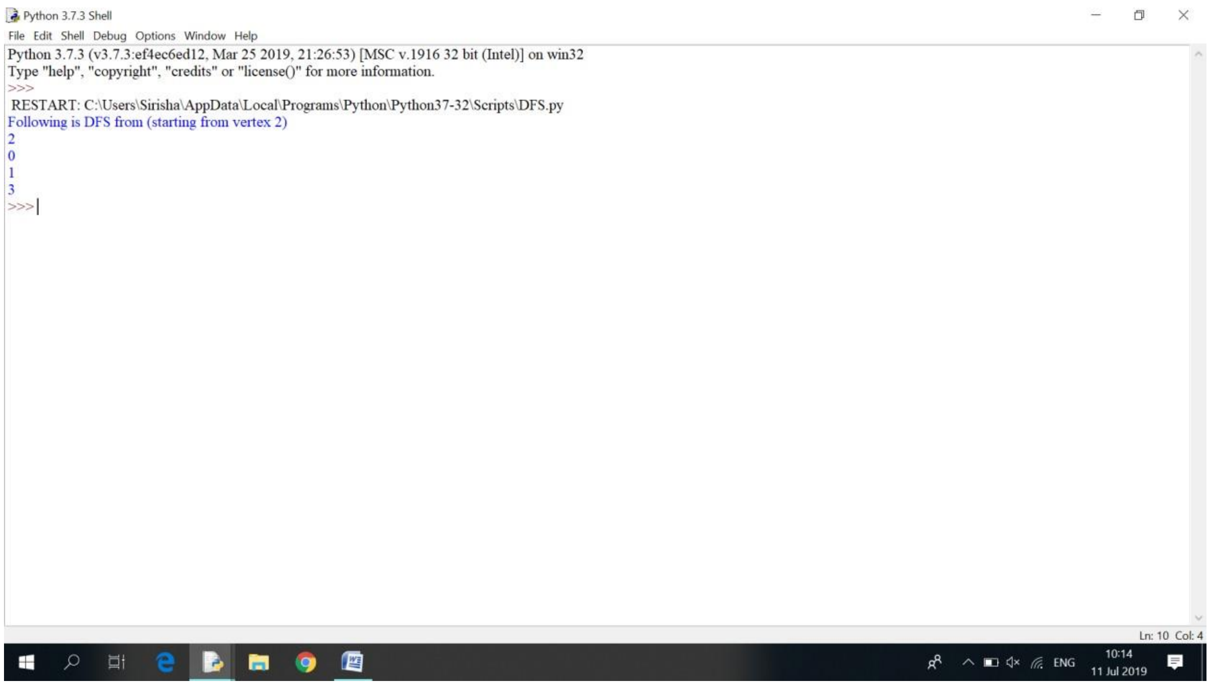
a. Create : Open a new file in Python shell, write a program and save the program with .py extension.
b. Execute : Go to Run -> Run module (F5)

SOURCE CODE:

Depth first search

```
from collections import defaultdict
class Graph:

    # Constructor
    def_init_(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)
    # A function used by DFS
    def DFSUtil(self,v,visited):

        # Mark the current node as visited and print it
        visited[v]= True
        print (v),

        # Recur for all the vertices adjacent to this vertex
        for i in self.graph[v]:
            if visited[i] == False:
                self.DFSUtil(i, visited)


    # The function to do DFS traversal. It uses
    # recursive DFSUtil()
    def DFS(self,v):

        # Mark all the vertices as not visited
        visited = [False]*(len(self.graph))

        # Call the recursive helper function to print
        # DFS traversal
        self.DFSUtil(v,visited)


# Driver code
# Create a graph given in the above diagram
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)
```

Output:

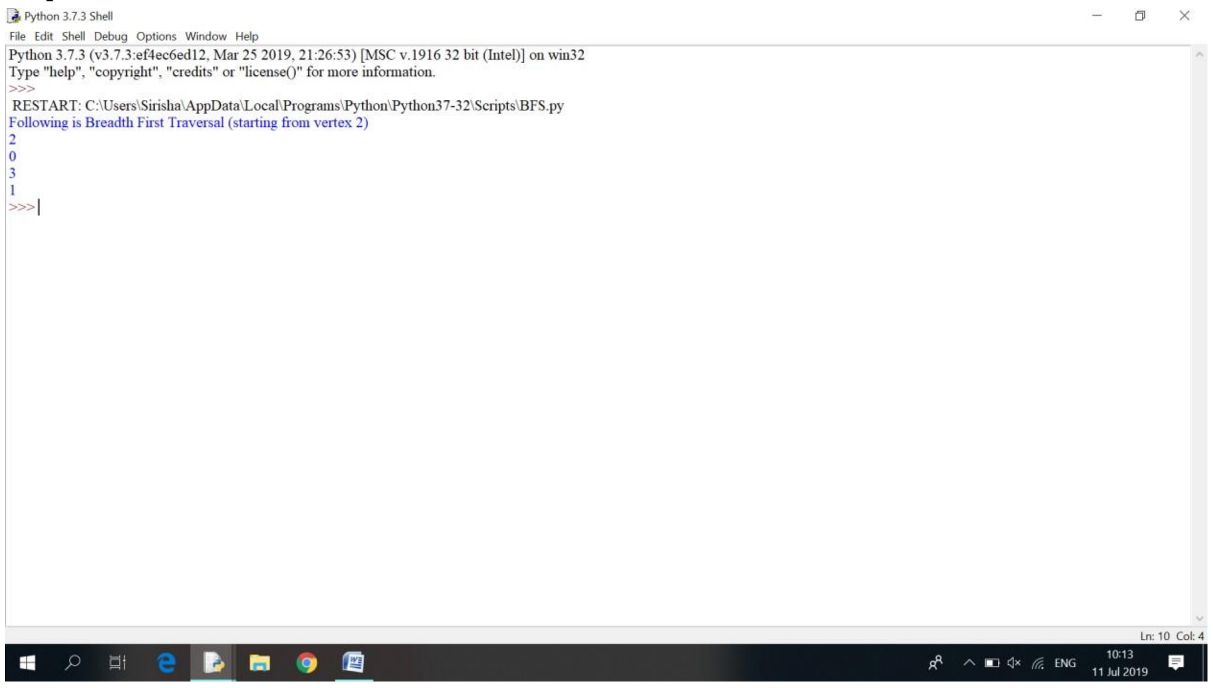


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\DFS.py
Following is DFS from (starting from vertex 2)
2
0
1
3
>>>
```

Breadth first search

```
from collections import defaultdict
class Graph:

    # Constructor
    def_init_(self):

        # default dictionary to store graph
        self.graph = defaultdict(list)

    # function to add an edge to graph
    def addEdge(self,u,v):
        self.graph[u].append(v)

    # Function to print a BFS of graph
    def BFS(self, s):

        # Mark all the vertices as not visited
        visited = [False]*(len(self.graph))

        # Create a queue for BFS
        queue = []

        # Mark the source node as visited and enqueue it
        queue.append(s)
        visited[s] = True

        while queue:

            # Dequeue a vertex from queue and print it
            s = queue.pop(0)
            print (s)
            # Get all adjacent vertices of the dequeued
            # vertex s. If a adjacent has not been visited,
            # then mark it visited and enqueue it
            for i in self.graph[s]:
                if visited[i] == False:
                    queue.append(i)
                    visited[i] = True


# Driver code
# Create a graph given in the above diagram
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)
```

Output:



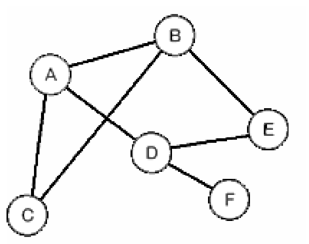
```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\BFS.py
Following is Breadth First Traversal (starting from vertex 2)
2
0
3
1
>>>
```

PRE-LAB VIVA QUESTIONS:

a. What is graph?
b. List various ways of representations of graph?
c. How many graph traversal algorithms are there?

LAB ASSIGNMENT:

a. Find DFS traversal of the following graph



b. Deduce the time complexity of DFS algorithm

POST-LAB VIVA QUESTIONS:

a. What is the advantage of circular queue over linear queue?
b. Where priority queues are used?
c. What is DEQUE?

WEEK – 12
BASICS OF PYTHON

OBJECTIVE:

Write a Python script 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.

RESOURCE:

Python 3.7.3

PROGRAM LOGIC:

Binary search tree:

```
 Create():
   If root == NULL
      return NULL;
   If number == root->data
      return root->data;
   If number < root->data
      return search(root->left)
   If number > root->data
      return search(root->right)
 Inorder(tree):
   1. Traverse the left subtree, i.e., call Inorder(left-subtree)
   2. Visit the root.
   3. Traverse the right subtree, i.e., call Inorder(right-subtree)
 Preorder(tree):
   1. Visit the root.
   2. Traverse the left subtree, i.e., call Preorder(left-subtree)
   3. Traverse the right subtree, i.e., call Preorder(right-subtree)
 Postorder(tree):
   1. Traverse the left subtree, i.e., call Postorder(left-subtree)
   2. Traverse the right subtree, i.e., call Postorder(right-subtree)
   3. Visit the root.
 Number of nodes in BST:
   CountNodes(node x)
   set n=1  //global variable
   If x=NULL
             return 0
   If(x->left!=NULL)
      n=n+1
   CountNode(x->left)
   If(x->right!=NULL)
      n=n+1
   CountNode(x->right)
   return n
```

PROCEDURE:

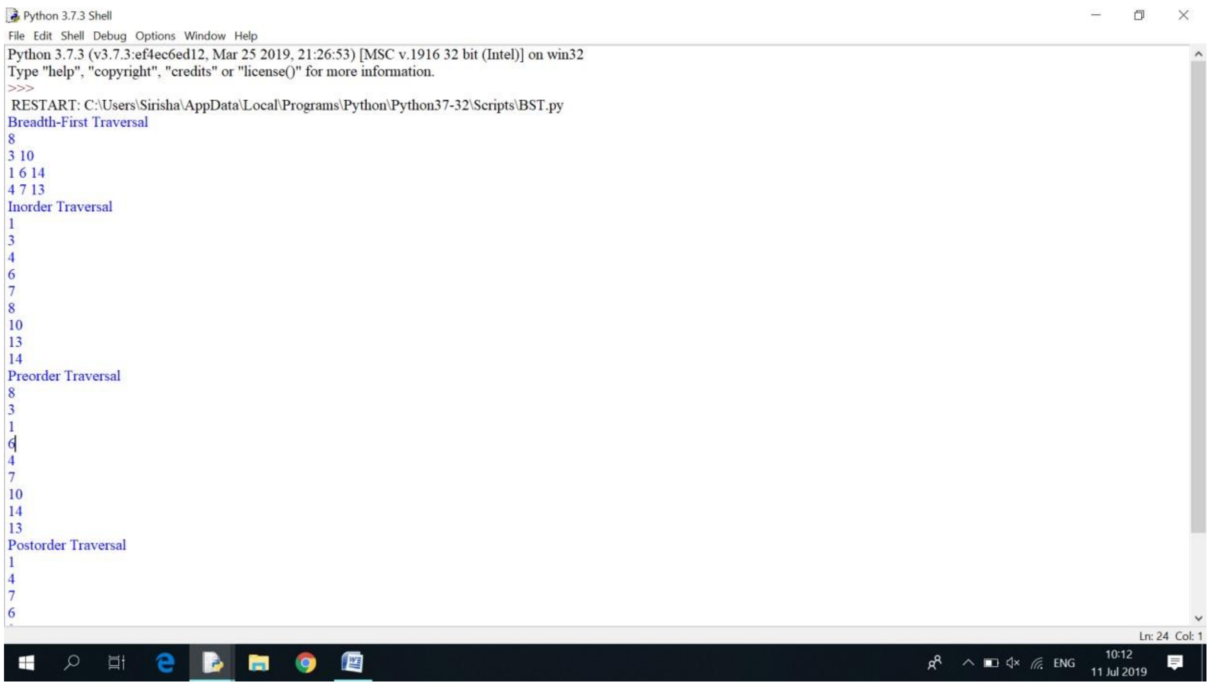
a. Create : Open a new file in Python shell, write a program and save the program with .py extension.
b. Execute : Go to Run -> Run module (F5)

SOURCE CODE:

Binary search tree:

```
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)

 #Driver  code
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:

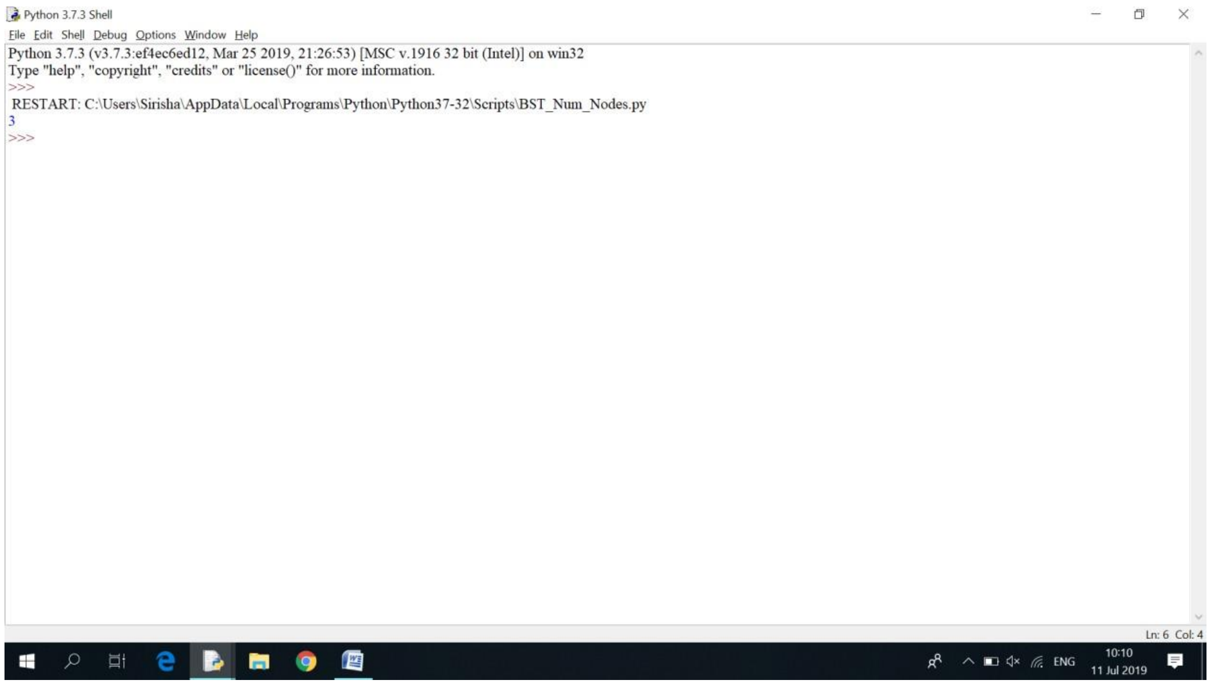


```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\BST.py
Breadth-First Traversal
8
3 10
1 6 14
4 7 13
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
```

Count the number of nodes in BST:

```
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())

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

Output:



```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:26:53) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
 RESTART: C:\Users\Sirisha\AppData\Local\Programs\Python\Python37-32\Scripts\BST_Num_Nodes.py
3
>>>
```

PRE-LAB VIVA QUESTIONS:

a. Define tree traversal and mention types of traversal?
b. Define a tree?
c. Define height of a tree?
d. Define depth of a tree?
e. Define degree of a node?
f. Define Degree of a tree?
g. Define Terminal node or leaf node?
h. Define Non-terminal node?
i. Define Sibling?
j. Define Binary Tree?
k. Write the properties of Binary Tree?
l. Find the minimum and maximum height of a binary tree?

LAB ASSIGNMENT:

a. Formulate a program to create a Binary Tree of integers?
b. Write a recursive program, for traversing a binary tree in preorder, inorder and postorder?
c. Compose a non-recursive program, for traversing a binary tree in preorder, inorder and postorder?

d. Write a program to check balance property of a tree?

POST-LAB VIVA QUESTIONS:

a. Write the balance factor of a Binary Tree?
b. What is a spanning Tree?
c. Define a Complete Binary Tree?
d. List out the applications of Binary Tree?
e. Write the two approaches for Binary Tree Traversal?
f. Write the various operations performed in the binary search tree?
g. List out few of the Application of tree data-structure?
h. Define pre-order traversal.
i. Define post-order traversal.
j. Define in-order traversal.