**SOFTWARE DESIGN LABORATORY**

**PreLab**

**• Readings**

* 1. **Python DS: • Chapters 7 ,8 ,10 ,12 : Ref Lab3 Fundamentals Of Python Data Structures , Kenneth A. Lamber**

This chapter introduces the stack, a widely used collection in computer science. The stack is the most straightforward collection to explain and use. It does, however, have some intriguing uses, three of which are addressed later in this chapter. In this chapter we will be able to describe the features and behavior of stack and we’ll be able to recognize application where it is appropriate to use a stack. Also we can explain how the system call stack provides run-time support for recursive subroutines and be able to design and implement a backtracking algorithm that uses a stack.

**1.2 Python DS: Chapters 7, 10 : Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.al**

This chapter introduces the linked list, which is a data structure that can be used instead of an array-based sequence Array-based sequences and linked lists both preserve elements in a specific order, but they do so in quite different ways. When comparing array-based sequences and linked lists, this chapter will show how they trade off advantages and disadvantages. A numeric index cannot efficiently access elements of a linked list, and we can't know if anode is the second, fifth, or twentieth node in the list merely by looking at it. Linked lists, on the other hand, overcome the three drawbacks mentioned above for array-based sequences.

**• Answers to Questions Ref Lab3 Data Structures and Algorithms in Python Michael T. Goodrich, Roberto Tamassia et.**

**7.1 If (current\_node->next-next == NULL )** then the current node is the second last node.

**2. InLab • Write your Objectives (you can have your own objectives**

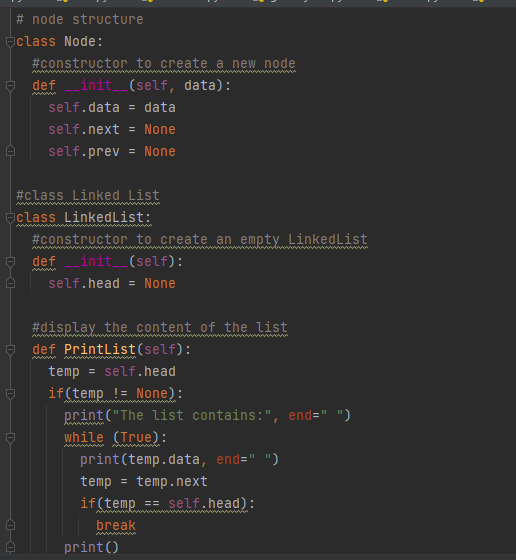
Implementation of Single / Circular / Doubly Linked List a.

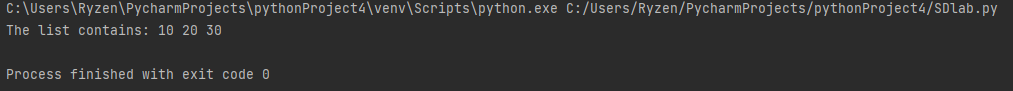
Write Python programs for the following operations on Single / Circular / Doubly Linked List.

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

b. To store a polynomial expression in memory using Single / Circular / Doubly Linked List.

In Python, circular doubly linked list can be represented as a class and a Node as a separate class. The LinkedList class contains a reference of Node class type.



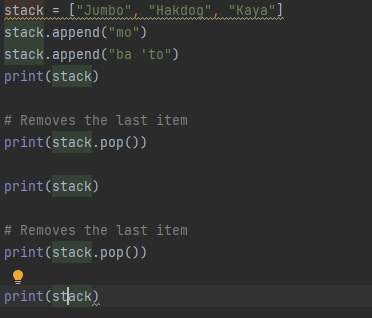
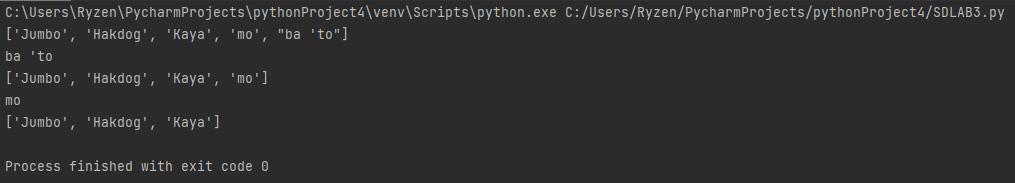


A circular doubly linked list can be traversed from any node of the list using a temp node. Keep on moving the temp node to the next one and displaying its content. Stop the traversal, after reaching the starting node.

**Implementation of Stack and Queue**

**a. Design and implement Stack and its operations using List.**

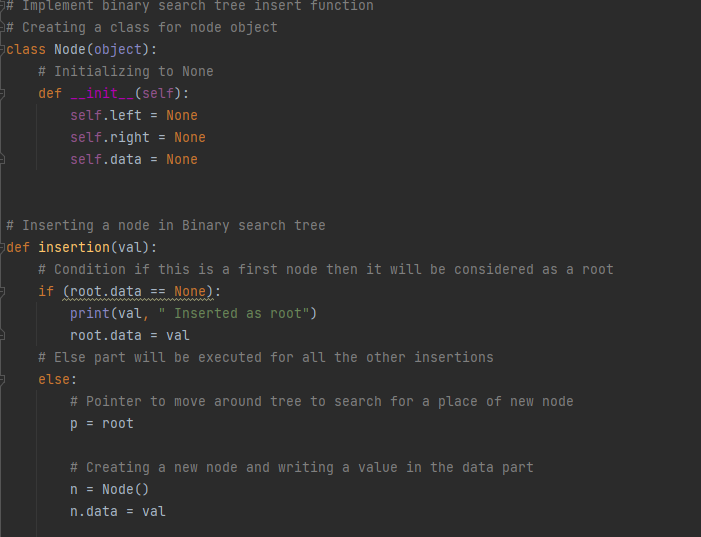
**b. Design and implement Queue and its operations using Lis**

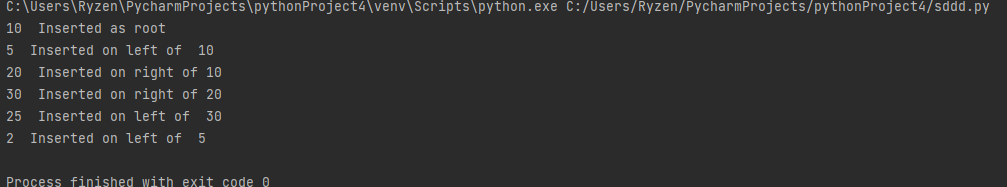
**Stack** works on the principle of “Last-in, first-out”. Also, the inbuilt functions in Python make the code short and simple. To add an item to the top of the list, i.e., to push an item, we use **append ()** function and to pop out an element we use **pop()** function. These functions work quiet efficiently and fast in end operations.  

**Implementation of Binary Search Tree**

1. **Create a binary search tree.**
2. **B. Traverse the above binary search tree recursively in pre-order, post-order and in-order.**
3. **c. Count the number of nodes in the binary search tree.**

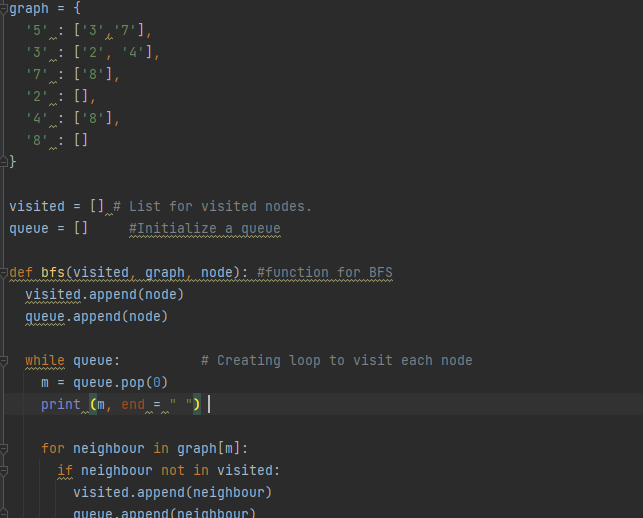
In order to insert a node in binary search tree, we need to make sure that none of the properties of binary search tree is violated while inserting a node.  
Here, the each call to the insertion method will insert one node to a binary search tree. The first call to insertion will make that node as a root node. All the other insertions will take place considering the value of root node.  
The code below shows the insertion function which places the node in a tree such that none of the properties of binary search tree is violated.

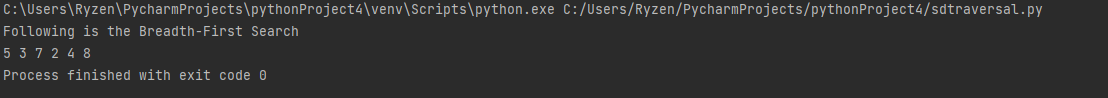




**Implementation of Traversal Algorithm for Breadth first traversal**

1. **Create a traversal algorithm for Breadth first traversal**





We will first generate the graph in the preceding code, for which we will apply the breadth-first search. After that, we'll make two lists: one to keep track of the graph's visited nodes, and another to keep track of the nodes in the queue. After the above process, we will declare a function with the parameters as visited nodes, the graph itself and the node respectively. And inside a function, we will keep appending the visited and queue lists. Then we'll execute the while loop for the queue of nodes to visit, and then remove and print the same node as it's visited. Finally, we'll use the for loop to check for unvisited nodes before appending them to the visited and queue lists. We'll ask the user to create the bfs function with the first node we want to visit as the driver code.