```
#IMPLEMENTATION OF BINARY TREE USING LINKED LIST IN PYTHON
#INCLUDES INORDER, PREORDER AND POSTORDER TRAVERSALS.
class Node:
    def __init__(self,data):
        self.data = data
        self.left = None
        self.right = None

class BinaryTree:
    def __init__(self):
        #Represent the root of binary tree
        self.root = None

    #Function to add new node to the binary tree
    def insertNode(self, data):
        #Create a new node
        newNode = Node(data)

        #Check whether tree is empty
        if(self.root == None):
            self.root = newNode
            return;
        else:
            queue = []
            #Add root to the queue if root is not empty
            queue.append(self.root)

            while(True):
                #the first elements of the queue are popped
                #as they will be respective left and right children
                node = queue.pop(0)
```

```
            #If node has both left and right child, add both the child to queue
            if(node.left != None and node.right != None):
                queue.append(node.left)
                queue.append(node.right)
            else:
                #If node has no left child, make newNode as left child
                if(node.left == None):
                    node.left = newNode
                    queue.append(node.left)
                else:
                    #If node has left child but no right child, make newNode as right child
                    node.right = newNode
                    queue.append(node.right)
                break;

    #inorder() will perform inorder traversal on binary search tree
    def inorderTraversal(self, node):

        #Check if tree is empty
        if(self.root == None):
            print("Tree is empty")
            return
        else:
            if(node.left != None):
                self.inorderTraversal(node.left)
            print(node.data,end=" ")
            if(node.right!= None):
                self.inorderTraversal(node.right)
    '''
    #preorder() will perform preorder traversal on binary search tree
    def preorderTraversal(self, node):
```

```
    #Check whether tree is empty
    if(self.root == None):
        print("Tree is empty")
        return
    else:
        print(node.data)
        if(node.left != None):
            self.inorderTraversal(node.left)
        if(node.right!= None):
            self.inorderTraversal(node.right)
'''
'''
#postorder() will perform inorder traversal on binary search tree
def postorderTraversal(self, node):
    #Check whether tree is empty
    if(self.root == None):
        print("Tree is empty")
        return
    else:
        if(node.left != None):
            self.inorderTraversal(node.left)
        if(node.right!= None):
            self.inorderTraversal(node.right)
        print(node.data)
'''
bt = BinaryTree()
#Adding nodes to the binary tree
bt.insertNode('A')
print("Thank you for downloading from Techprodezza!")
#A will become root node of the tree
```

```
print("Binary tree after insertion: ")
#Binary after inserting nodes
bt.inorderTraversal(bt.root)

bt.insertNode('B')
bt.insertNode('C')
#B will become left child and C will become right child of root node A
print("\nBinary tree after insertion: ")
#Binary after inserting nodes
bt.inorderTraversal(bt.root)

bt.insertNode('D')
bt.insertNode('E')
#D will become left child and E will become right child of node B
print("\nBinary tree after insertion: ")
#Binary after inserting nodes
bt.inorderTraversal(bt.root)

bt.insertNode('F')
#F will become the left child
print("\nBinary tree after insertion: ")
#Binary after inserting nodes
bt.inorderTraversal(bt.root)
```