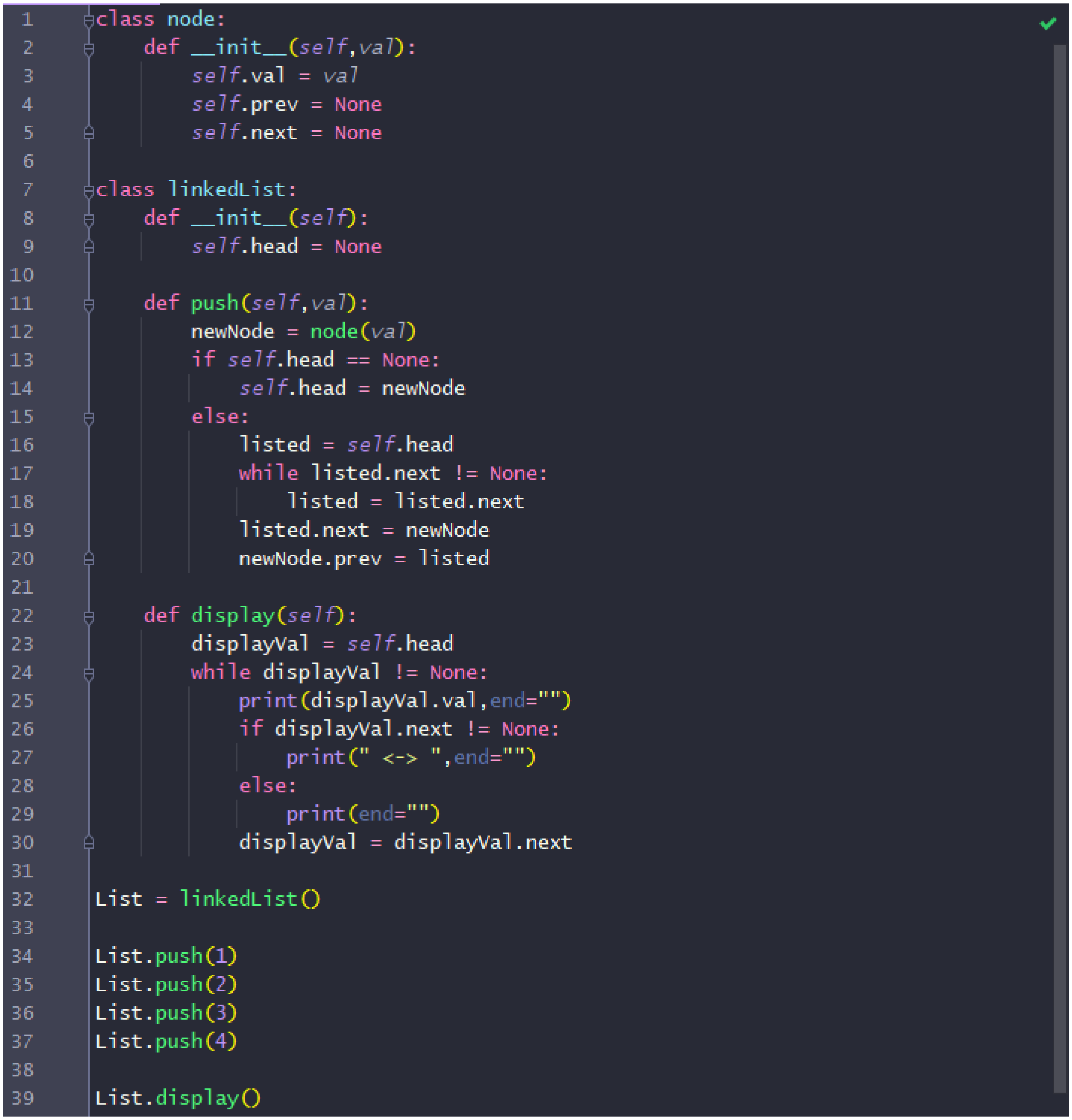
Linked list (Forward & Backward) and binary tree

Part 1 : Forward and backward linked list

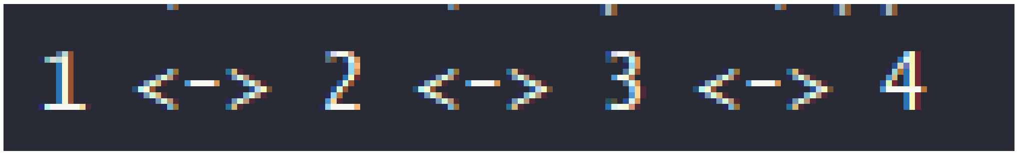
Exercise 1 : Create forward and backward linked list

Code:

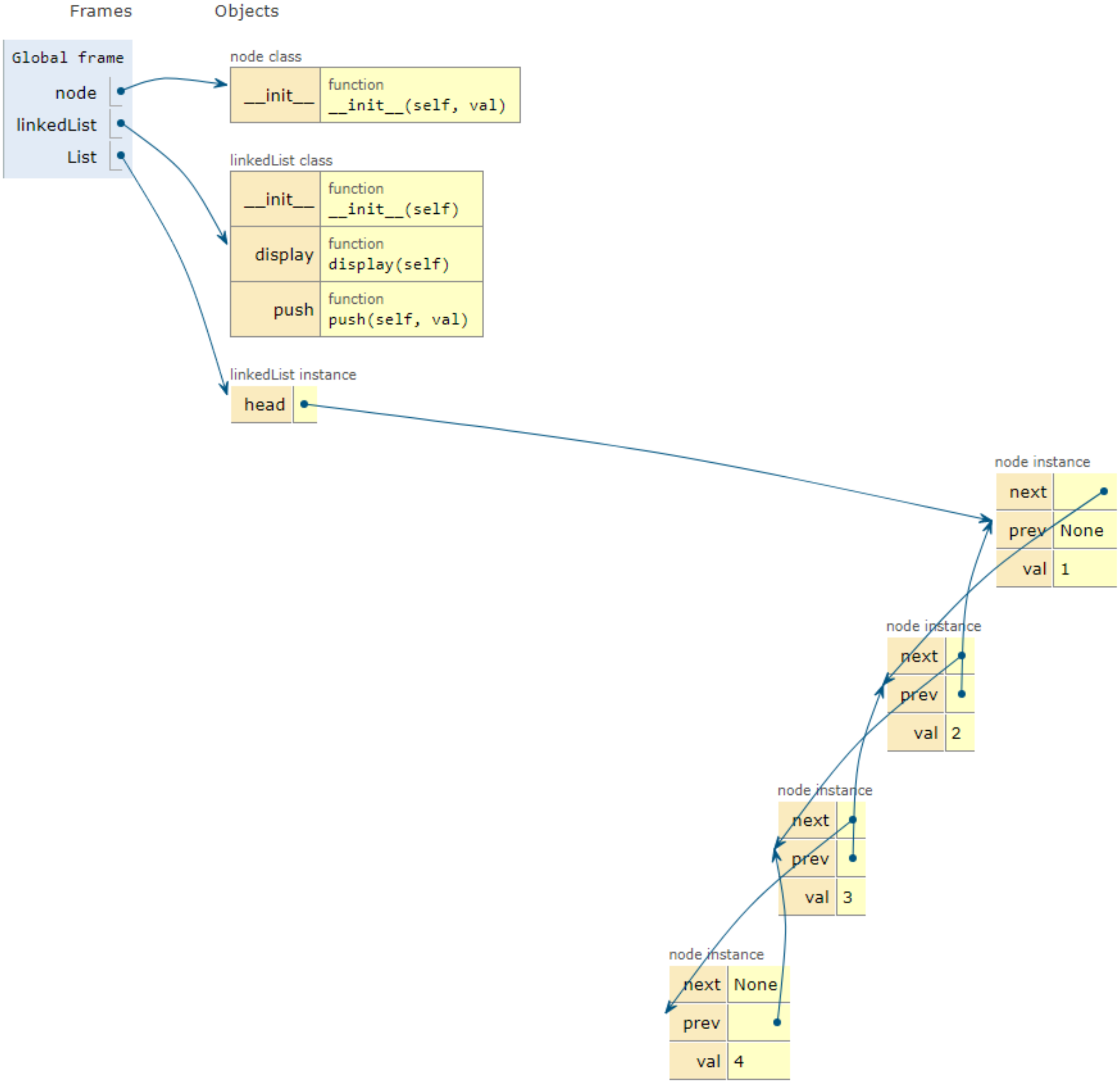


```
class node:
    def __init__(self,val):
        self.val = val
        self.prev = None
        self.next = None

class linkedList:
    def __init__(self):
        self.head = None

    def push(self,val):
        newNode = node(val)
        if self.head == None:
            self.head = newNode
        else:
            listed = self.head
            while listed.next != None:
                listed = listed.next
            listed.next = newNode
            newNode.prev = listed

    def display(self):
        displayVal = self.head
        while displayVal != None:
            print(displayVal.val,end="")
            if displayVal.next != None:
                print(" <-> ",end="")
            else:
                print(end="")
            displayVal = displayVal.next

List = linkedList()

List.push(1)
List.push(2)
List.push(3)
List.push(4)

List.display()
```

Result :



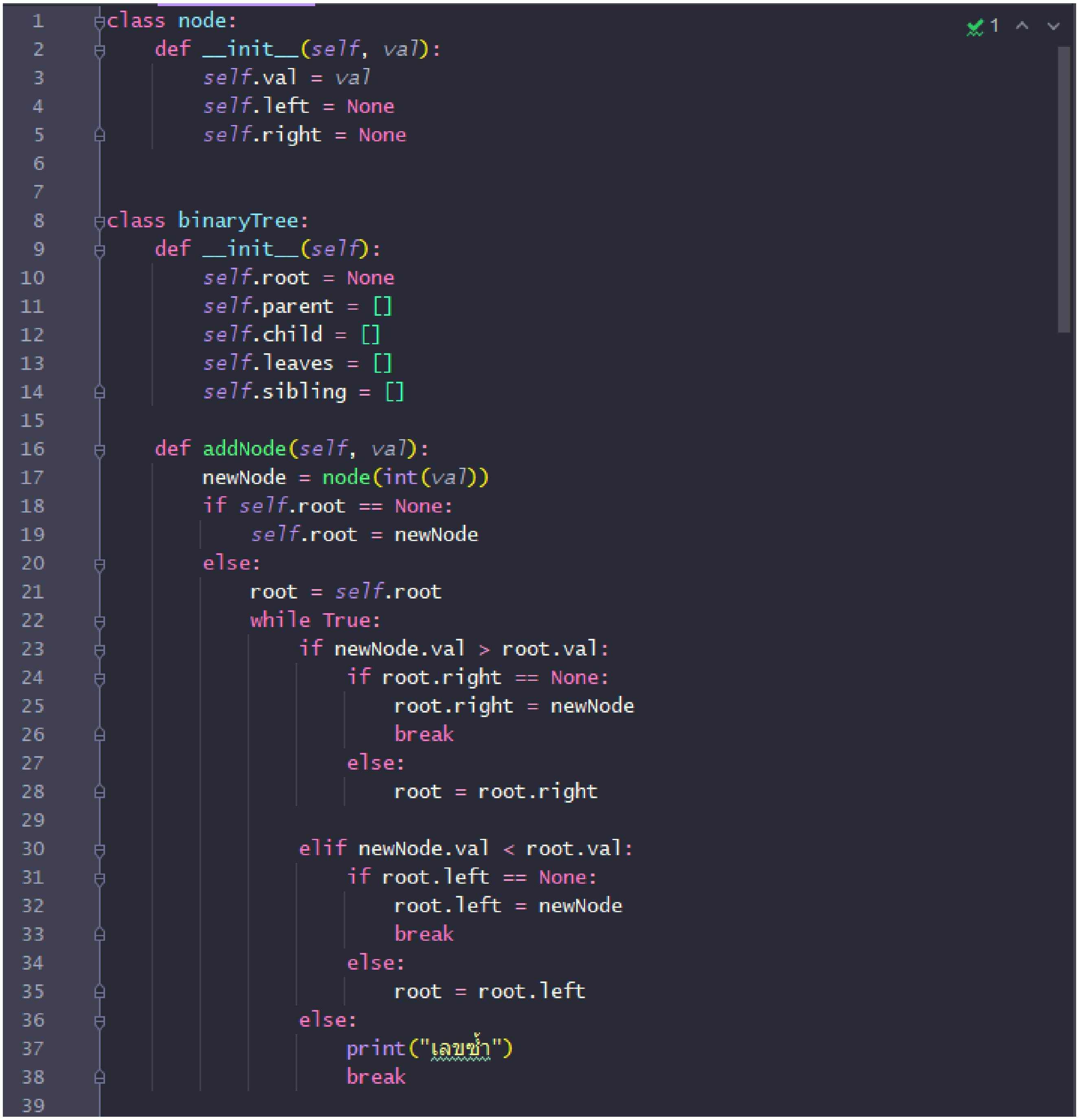
```
1 <-> 2 <-> 3 <-> 4
```


Frames
Objects
Global frame
node
linkedList
List
node class
__init__
function
__init__(self, val)
linkedList class
__init__
function
__init__(self)
display
function
display(self)
push
function
push(self, val)
linkedList instance
head
node instance
next
prev None
val 1
node instance
next
prev
val 2
node instance
next
prev
val 3
node instance
next None
prev
val 4

Part 2 : Binary tree

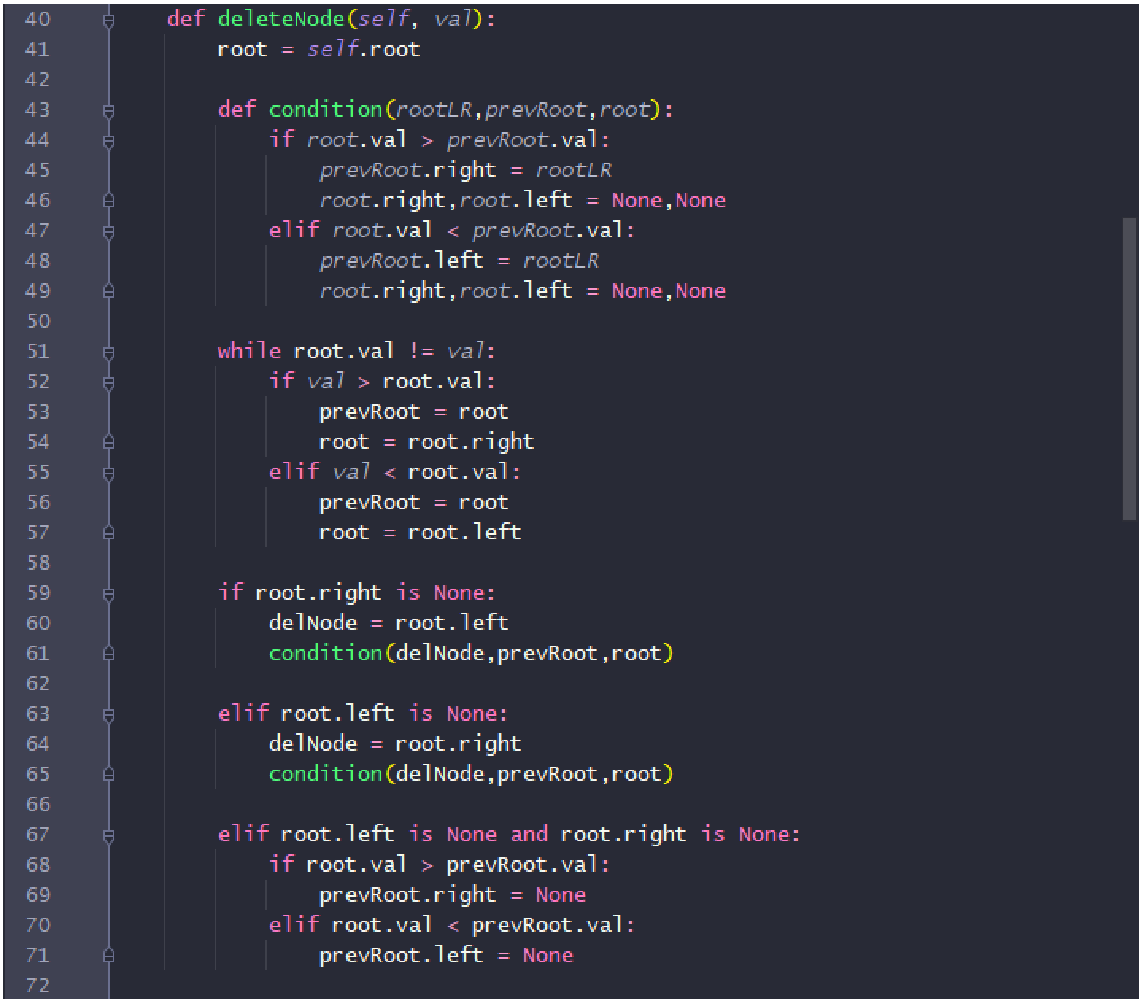
Code :

Class and insert node function



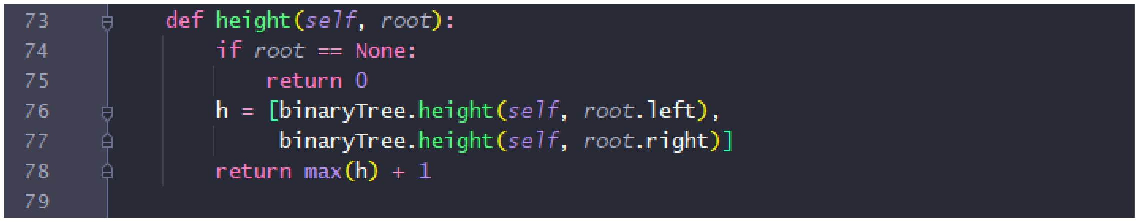
```
class node:
    def __init__(self, val):
        self.val = val
        self.left = None
        self.right = None


class binaryTree:
    def __init__(self):
        self.root = None
        self.parent = []
        self.child = []
        self.leaves = []
        self.sibling = []

    def addNode(self, val):
        newNode = node(int(val))
        if self.root == None:
            self.root = newNode
        else:
            root = self.root
            while True:
                if newNode.val > root.val:
                    if root.right == None:
                        root.right = newNode
                        break
                    else:
                        root = root.right

                elif newNode.val < root.val:
                    if root.left == None:
                        root.left = newNode
                        break
                    else:
                        root = root.left
                else:
                    print("เลขซ้ำ")
                    break

```

Delete node function



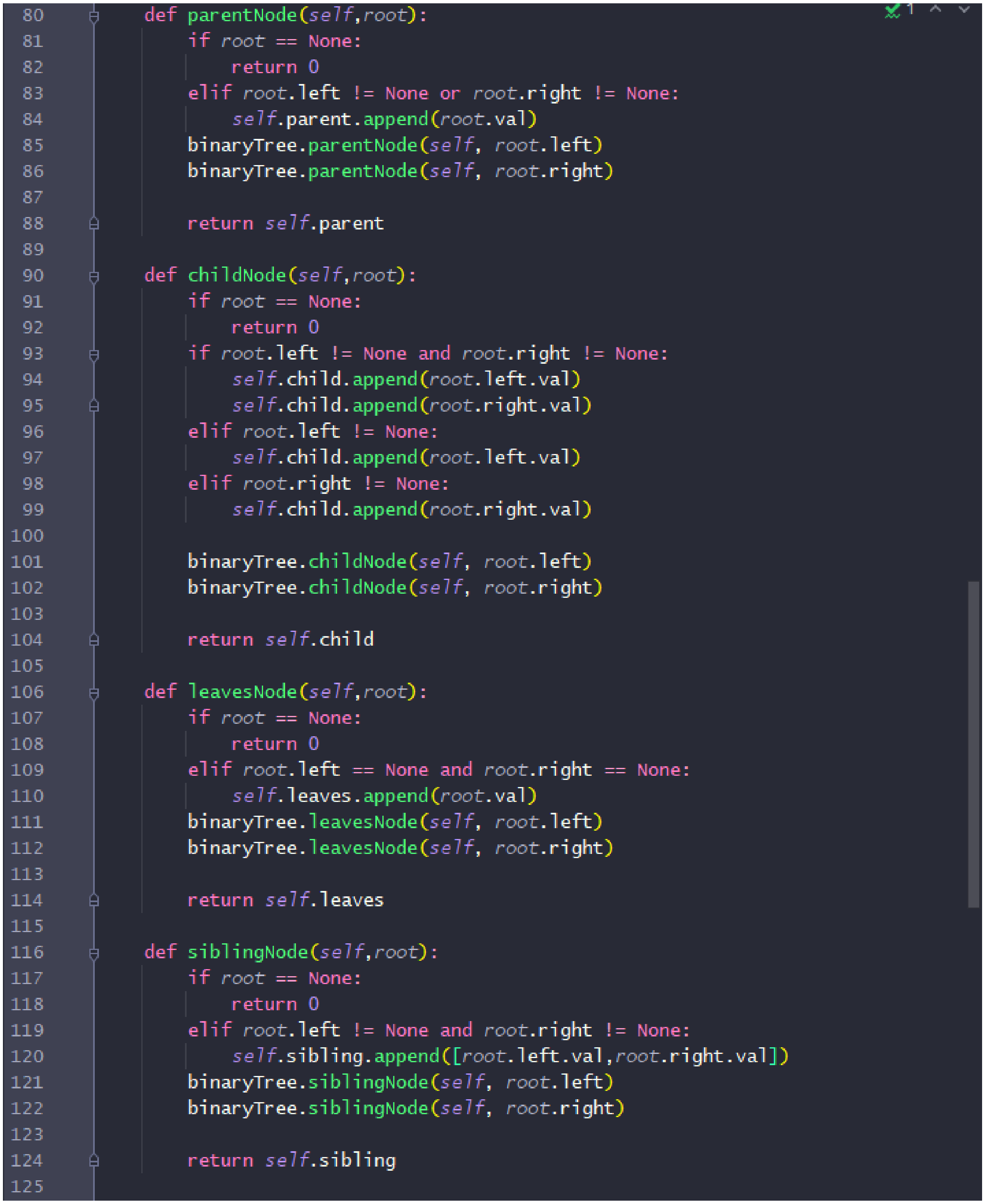
```python
    def deleteNode(self, val):
        root = self.root

        def condition(rootLR,prevRoot,root):
            if root.val > prevRoot.val:
                prevRoot.right = rootLR
                root.right,root.left = None,None
            elif root.val < prevRoot.val:
                prevRoot.left = rootLR
                root.right,root.left = None,None

        while root.val != val:
            if val > root.val:
                prevRoot = root
                root = root.right
            elif val < root.val:
                prevRoot = root
                root = root.left

        if root.right is None:
            delNode = root.left
            condition(delNode,prevRoot,root)

        elif root.left is None:
            delNode = root.right
            condition(delNode,prevRoot,root)

        elif root.left is None and root.right is None:
            if root.val > prevRoot.val:
                prevRoot.right = None
            elif root.val < prevRoot.val:
                prevRoot.left = None

```

Find height of binary tree function



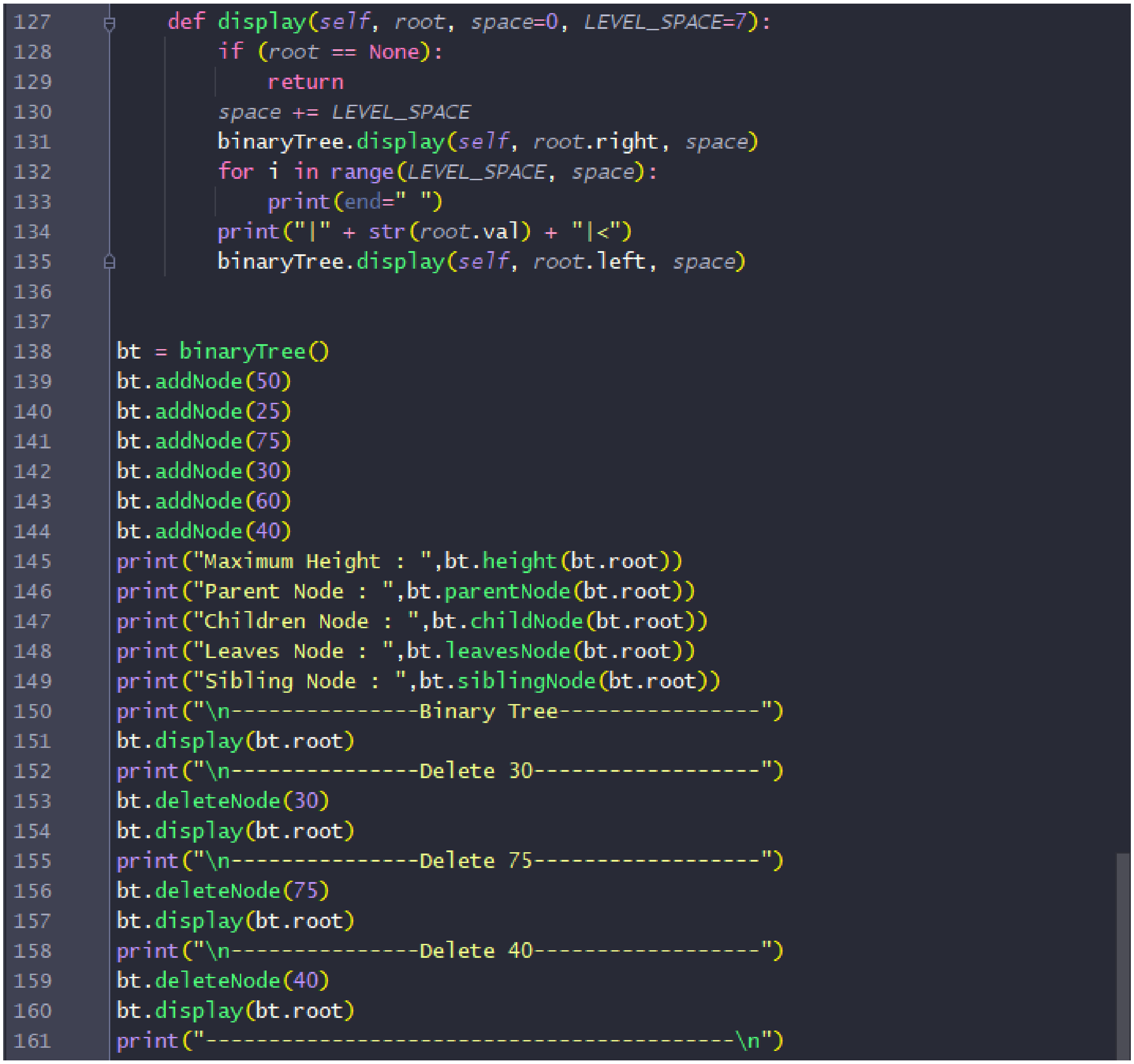
```python
    def height(self, root):
        if root == None:
            return 0
        h = [binaryTree.height(self, root.left),
             binaryTree.height(self, root.right)]
        return max(h) + 1

```

Find parent, child, leaves, sibling node of binary tree function



```python
    def parentNode(self,root):
        if root == None:
            return 0
        elif root.left != None or root.right != None:
            self.parent.append(root.val)
        binaryTree.parentNode(self, root.left)
        binaryTree.parentNode(self, root.right)

        return self.parent

    def childNode(self,root):
        if root == None:
            return 0
        if root.left != None and root.right != None:
            self.child.append(root.left.val)
            self.child.append(root.right.val)
        elif root.left != None:
            self.child.append(root.left.val)
        elif root.right != None:
            self.child.append(root.right.val)

        binaryTree.childNode(self, root.left)
        binaryTree.childNode(self, root.right)

        return self.child

    def leavesNode(self,root):
        if root == None:
            return 0
        elif root.left == None and root.right == None:
            self.leaves.append(root.val)
        binaryTree.leavesNode(self, root.left)
        binaryTree.leavesNode(self, root.right)

        return self.leaves

    def siblingNode(self,root):
        if root == None:
            return 0
        elif root.left != None and root.right != None:
            self.sibling.append([root.left.val,root.right.val])
        binaryTree.siblingNode(self, root.left)
        binaryTree.siblingNode(self, root.right)

        return self.sibling
```

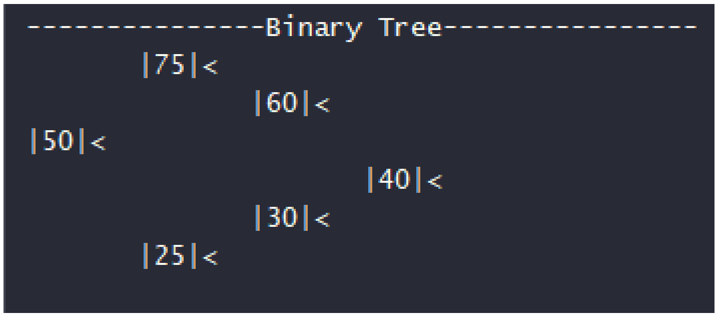
Display function



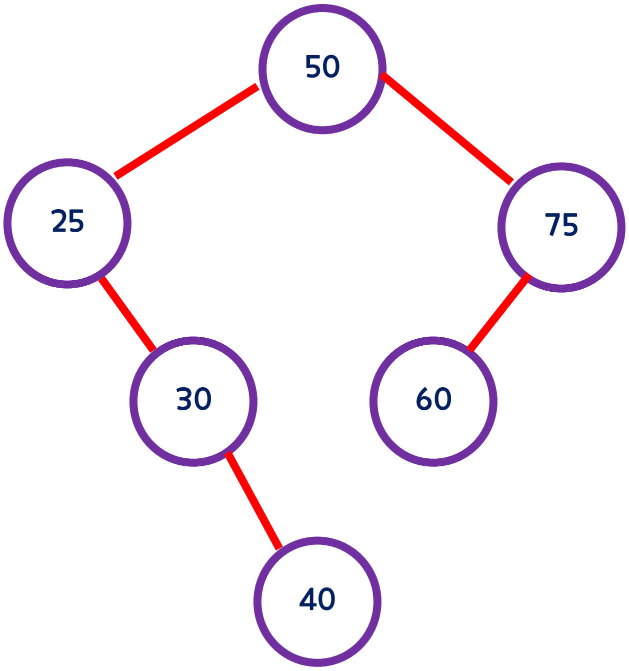
```
    def display(self, root, space=0, LEVEL_SPACE=7):
        if (root == None):
            return
        space += LEVEL_SPACE
        binaryTree.display(self, root.right, space)
        for i in range(LEVEL_SPACE, space):
            print(end=" ")
        print("|" + str(root.val) + "|<")
        binaryTree.display(self, root.left, space)


bt = binaryTree()
bt.addNode(50)
bt.addNode(25)
bt.addNode(75)
bt.addNode(30)
bt.addNode(60)
bt.addNode(40)
print("Maximum Height : ",bt.height(bt.root))
print("Parent Node : ",bt.parentNode(bt.root))
print("Children Node : ",bt.childNode(bt.root))
print("Leaves Node : ",bt.leavesNode(bt.root))
print("Sibling Node : ",bt.siblingNode(bt.root))
print("\n---------------Binary Tree----------------")
bt.display(bt.root)
print("\n---------------Delete 30------------------")
bt.deleteNode(30)
bt.display(bt.root)
print("\n---------------Delete 75------------------")
bt.deleteNode(75)
bt.display(bt.root)
print("\n---------------Delete 40------------------")
bt.deleteNode(40)
bt.display(bt.root)
print("------------------------------------------\n")
```

Result :

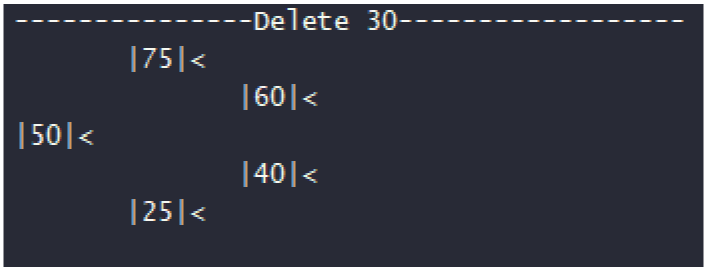
Exercise 2.1 : Create binary tree



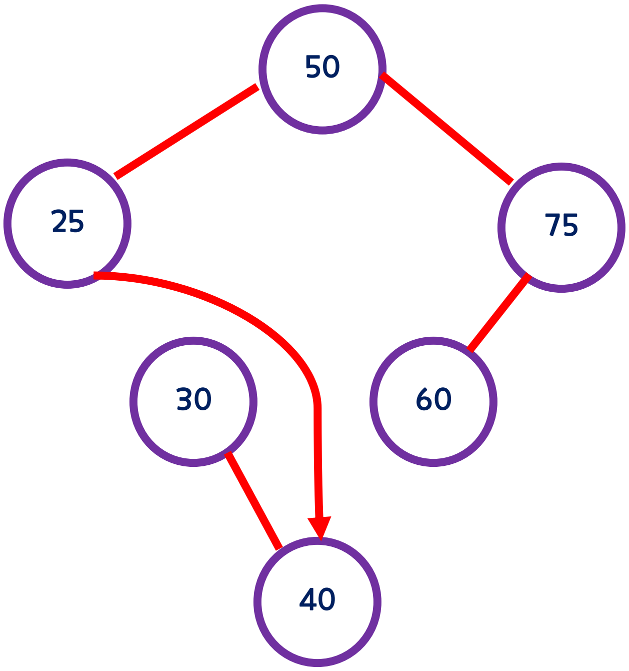
```
----------------Binary Tree----------------
        |75|<
                |60|<
|50|<
                        |40|<
                |30|<
        |25|<
```

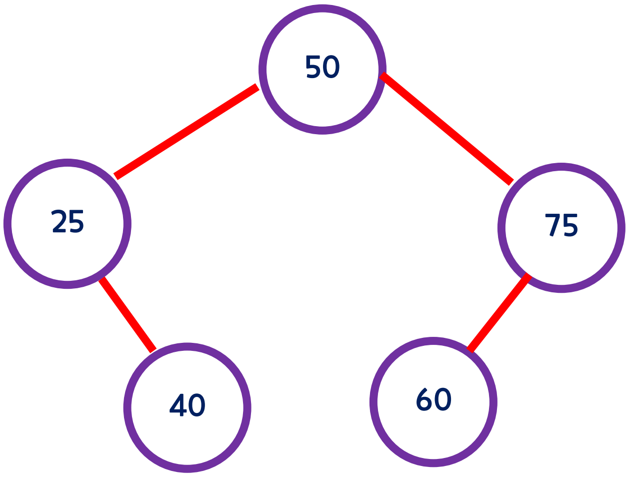


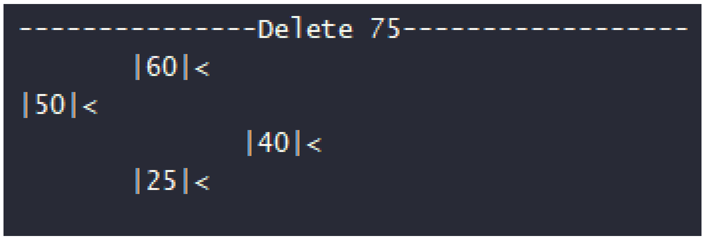
Exercise 2.1 : Remove value in binary tree



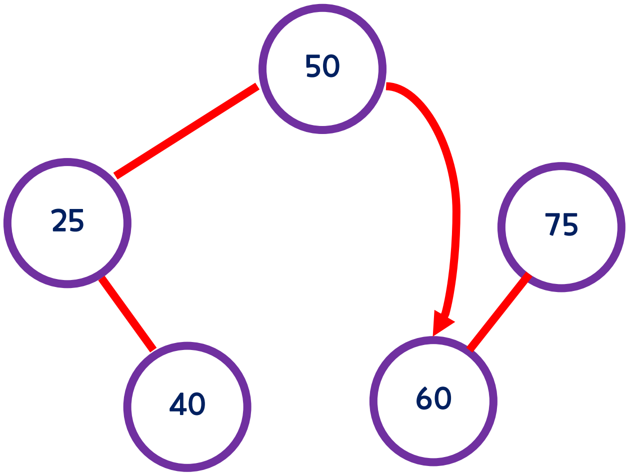
```
----------------Delete 30------------------
        |75|<
                |60|<
|50|<
                |40|<
        |25|<
```

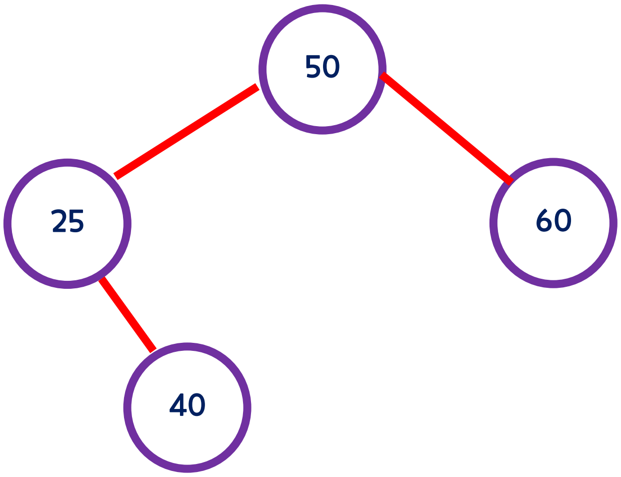


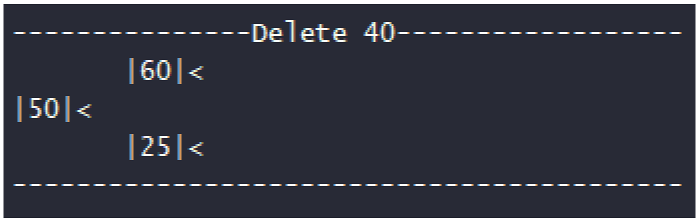




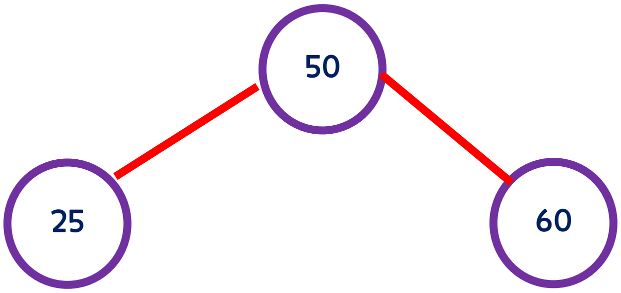
```
---------------Delete 75-----------------
       |60|<
|50|<
              |40|<
       |25|<
```



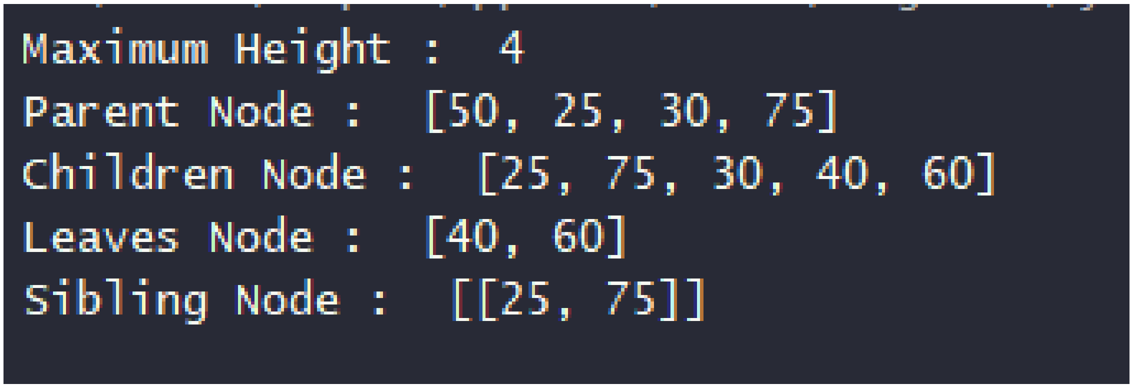




```
----------------Delete 40-------------------
        |60|<
|50|<
        |25|<
---------------------------------------------
```



Ex 2.3 : Theory part (coding is needed)



```
Maximum Height :  4
Parent Node :  [50, 25, 30, 75]
Children Node :  [25, 75, 30, 40, 60]
Leaves Node :  [40, 60]
Sibling Node :  [[25, 75]]
```