Tree

1] Binary tree in Python :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

#driver code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.left.right=Node(40)
```

2] Inorder Traversal in Python:

```
class Node:
    def __init__(self,k):
        self.key=k
        self.left=None
        self.right=None


def inorder(root):
    if root !=None:
        inorder(root.left)
        print(root.key ,end=" ")
        inorder(root.right)

# drive code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.right.left=Node(40)
root.right.right=Node(50)

inorder(root)
```

OUTPUT :

20 10 40 30 50

3] preorder traversal in python :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

def preorder(root):
    if root!=None:
        print(root.key)
        preorder(root.left)
        preorder(root.right)


#driver Code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.right.left=Node(40)
root.right.right=Node(50)

preorder(root)
```

OUTPUT :

10

20

30

40

50

4] PostOrder Traversal in Python :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

def postorder(root):
    if root !=None:
        postorder(root.left)
        postorder(root.right)
        print(root.key)

#driver code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.right.left=Node(40)
root.right.right=Node(50)

postorder(root)
```

OUTPUT :

20

40

50

30

10

5] Size of Binary Tree In Python :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

def treeSize(root):
    if root==None:
        return 0
    else:
        ls=treeSize(root.left)      #we can also use
        rs=treeSize(root.right)     #return treeSize(root.left)+treeSize(root.right) +1
        return ls+rs+1

#driver Code

root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.right.left=Node(40)
root.right.right=Node(50)

print(treeSize(root))
```

OUTPUT :

5

6] maximum in binary tree :

```
from math import inf

class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k


def getMax(root):
    if root==None:
        return -inf
    else:
        lm=getMax(root.left)
        rm=getMax(root.right)
        return max(root.key,lm,rm)
        # return max(root.key, getMax(root.left), getMax(root.right))


#driver Code

root=Node(10)
root.left=Node(80)
root.right=Node(15)
root.right.left=Node(40)
root.right.right=Node(50)

print(getMax(root))
```

OUTPUT :

80

7] search in binary tree :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k


def searchKey(root,key):
    if root is None:
        return False

    elif root.key==key:
        return True

    elif searchKey(root.left,key):
        return True

    else:
        return searchKey(root.right,key)

# Driver Code

root=Node(10)
root.left=Node(12)
root.right=Node(40)
root.right.left=Node(15)
root.right.right=Node(30)

print(searchKey(root,15))
print(searchKey(root,25))
print(searchKey(root,30))
```

OUTPUT :

True

False

True

8] Height Of Binary Tree :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

def heightTree(root):
    if root==None:
        return 0

    else:
        lh=heightTree(root.left)
        rh=heightTree(root.right)
        return max(lh,rh)+1
        #return max(heightTree(root.left),heightTree(root.right))+1


#Driver Code

root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.right.left=Node(40)

print(heightTree(root))
```

OUTPUT :

3

9] Iterative inorder traversal :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k


def itrInorder(root):
    if root is None:
        return

    st=[]
    curr=root
    while curr is not None:
        st.append(curr)
        curr=curr.left

    while len(st)>0:
        curr=st.pop()
        print(curr.key)
        curr=curr.right

        while curr is not None:
            st.append(curr)
            curr=curr.left
#Driver Code

root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.left.left=Node(40)
root.left.right=Node(50)

itrInorder(root)
```

OUTPUT :40

20

50

10

30

10] Iterative preorder traversal :

```
class Node:
    def __init__(self,k):
        self.left=None
        self.right=None
        self.key=k

def itrPreorder(root):
    if root is None:
        return

    st=[root]

    while len(st)>0:
        curr=st.pop()
        print(curr.key)

        if curr.right is not None:
            st.append(curr.right)

        if curr.left is not None:
            st.append(curr.left)


# Driver Code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.left.left=Node(40)
root.left.right=Node(50)
root.right.right=Node(60)

itrPreorder(root)
```

OUTPUT : 10

20

40

50

30

60

11] Level Order Traversal :

```
from collections import deque

class Node:
    def __init__(self,k):
        self.left = None
        self.right = None
        self.key=k

def printLevelOrder(root):
    if root is None:
        return

    q=deque()
    q.append(root)

    while len(q)>0:
        node=q.popleft()
        print(node.key)

        if node.left is not None:
            q.append(node.left)

        if node.right is not None:
            q.append(node.right)

#Driver Code
root=Node(10)
root.left=Node(20)
root.right=Node(30)
root.left.left=Node(40)
root.right.left=Node(50)
root.right.right=Node(60)
root.right.left.left=Node(70)
root.right.left.right=Node(80)

printLevelOrder(root)
```

OUTPUT :

10 20 30 40 50 60 70 80