# PREORDER TREE TRAVERSAL ALGORITHM IN PYTHON

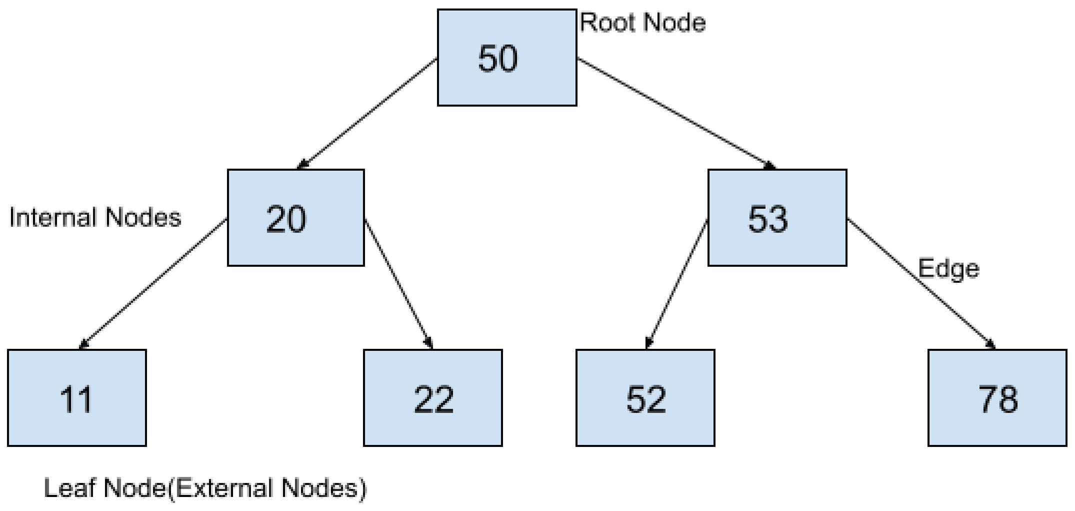
Binary trees are very useful in representing hierarchical data. In this article, we will discuss how to print all the elements in a binary tree in python. For this, we will use the preorder tree traversal algorithm. We will also implement the preorder tree traversal in python.

## What is the Preorder tree traversal ?

Preorder tree traversal is a depth first traversal algorithm. Here, we start from a root node and traverse a branch of the tree until we reach the end of the branch. After that, we move to the next branch. This process continues until all the nodes in the tree are printed.

The preorder tree traversal algorithm gets its name from the order in which the nodes of a tree are printed. In this algorithm, we first print a node. After that, we print the left child of the node. And, at last, we print the right child of the node. This process is recursive in nature. Here, the right child of a node is only printed when all the nodes in the left subtree of the current node and the current node itself have already been printed.

Let us understand the process using the binary tree given in the following image.



**Binary Tree**

Let us print all of the nodes in the above binary tree using the preorder traversal.

First, we will start from the root node and print its value i.e. 50.

After that we have to print the left child of 50. So, we will print 20.

After printing 20, we have to print the left child of 20. So, we will print 11.

11 has no children. So, we will move to node 20 and print its right child i.e. we will print 22.

22 has no children so, we will move to 20. Again all of the children of 20 have been printed. So, we will move to 50. At 50, we will print its right child as all the nodes in its left subtree have already been printed. Hence, we will print 53.

After printing 53, we have to print the left child of 53. So, we will print 52.

52 has no children. So, we will move to node 53 and print its right child i.e. we will print 78.

At this point, we have printed all of the nodes in the binary tree. So, we will terminate the process.

You can observe that we have printed the values in the order 50, 20, 11,22, 53, 52, 78. Let us now formulate an algorithm for the preorder tree traversal .

**Algorithm for preorder tree traversal**

As you have an overview of the entire process, we can formulate the algorithm for preorder tree traversal as follows.

1. Start from the root node.
2. If the root is empty, goto 6.
3. Print the root node.
4. Traverse the left subtree recursively.
5. Traverse the right subtree recursively.
6. Stop.

**Implementation of preorder tree traversal in Python**

As we have discussed the algorithm for preorder tree traversal and its working, Let us implement the algorithm and execute it for the binary tree given in the above image.

```
class BinaryTreeNode:
  def __init__(self, data):
    self.data = data
    self.leftChild = None
    self.rightChild = None
def preorder(root):
  # if root is None,return
  if root is None:
    return
  # print the current node
  print(root.data, end=" ,")
  # traverse left subtree
  preorder(root.leftChild)
  # traverse right subtree
  preorder(root.rightChild)
def insert(root, newValue):
  # if binary search tree is empty, create a new node and declare it as root
  if root is None:
    root = BinaryTreeNode(newValue)
```

```
        return root
    # if newValue is less than value of data in root, add it to left subtree and proceed recursively
    if newValue < root.data:
        root.leftChild = insert(root.leftChild, newValue)
    else:
        # if newValue is greater than value of data in root, add it to right subtree and proceed recursively
        root.rightChild = insert(root.rightChild, newValue)
    return root
root = insert(None, 50)
insert(root, 20)
insert(root, 53)
insert(root, 11)
insert(root, 22)
insert(root, 52)
insert(root, 78)
print("Preorder traversal of the binary tree is:")
preorder(root)
```

**Output:**

```
Preorder traversal of the binary tree is:
50 ,20 ,11 ,22 ,53 ,52 ,78 ,
```

You can observe that the code gives the same output that we have derived while discussing this algorithm.