Ujjwal Yadav
ujjwal.uy24@gmail.com

Post Order Traversal of Tree

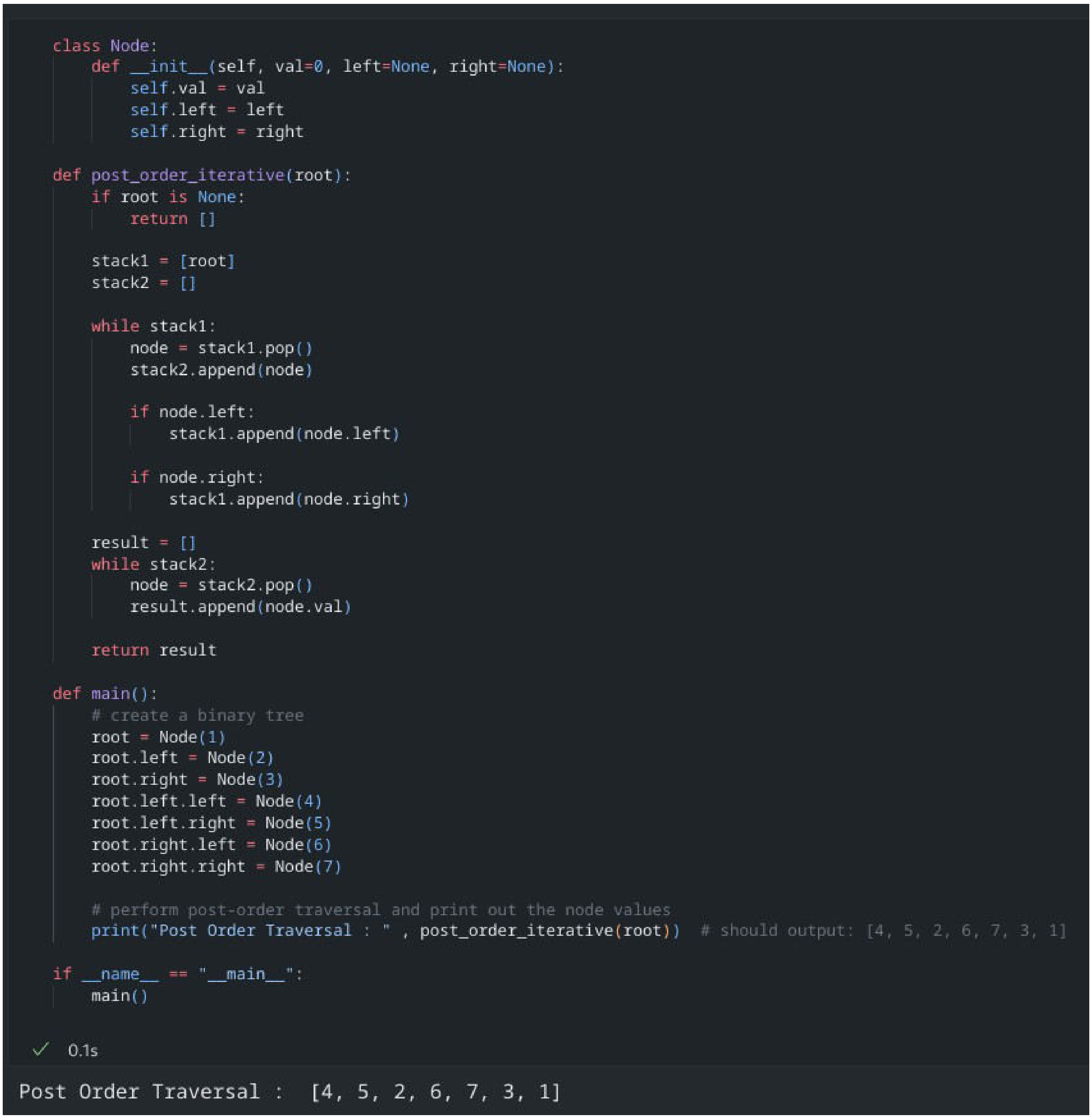
In post-order traversal (without using recursion) is carried out iteratively using two stacks. The the first stack is used to keep track of the nodes that we still need to analyse while second stack is used to store the nodes in the order we want them to appear in the output.

The root node is first pushed onto the top stack. Following that, we go into a loop where we pop nodes off of the top stack, move them to the bottom stack, and then move their child (if any) back to the top stack. Once there are no more nodes remaining to process, this procedure continues.

After processing each node, we remove them one at a time from the second stack and add their values to the final result list that we return.

The time and space complexity of this implementation, where n is the number of nodes in the tree, are O(n) & O(n) respectively .

Below is the Code of the above approach.



```python
class Node:
    def __init__(self, val=0, left=None, right=None):
        self.val = val
        self.left = left
        self.right = right

def post_order_iterative(root):
    if root is None:
        return []

    stack1 = [root]
    stack2 = []

    while stack1:
        node = stack1.pop()
        stack2.append(node)

        if node.left:
            stack1.append(node.left)

        if node.right:
            stack1.append(node.right)

    result = []
    while stack2:
        node = stack2.pop()
        result.append(node.val)

    return result

def main():
    # create a binary tree
    root = Node(1)
    root.left = Node(2)
    root.right = Node(3)
    root.left.left = Node(4)
    root.left.right = Node(5)
    root.right.left = Node(6)
    root.right.right = Node(7)

    # perform post-order traversal and print out the node values
    print("Post Order Traversal : " , post_order_iterative(root))  # should output: [4, 5, 2, 6, 7, 3, 1]

if __name__ == "__main__":
    main()
```

✓ 0.1s

```
Post Order Traversal :  [4, 5, 2, 6, 7, 3, 1]
```

This code creates a binary tree with seven nodes, where the root node
has a value of 1, its left child has a value of 2, and its right child has a value of 3.
The left child has two children of its own with values 4 and 5, and the right child
also has two children with values 6 and 7.

```
        1
      /   \
     2     3
    / \   / \
   4   5 6   7
```

We then call the post_order_iterative function with the root node of the tree as an argument. This function performs post-order traversal of the tree, which means that it visits the left child, then the right child, and then the root node of each sub-tree in the tree.

The expected output for this example is [4, 5, 2, 6, 7, 3, 1] , which corresponds to the post-order traversal of the tree