

Work with your neighbor. (This will be graded for participation only.)

---

1. There are three common ways to traverse a tree. For a binary tree, list the order of visiting the nodes and the children for each of these traversals.

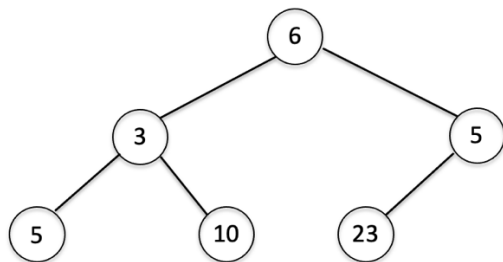
**ANS:**

pre-order:     node, left, right

in-order:      left, node, right

post-order:    left, right, node

2. Give the above traversals for the following tree:



**ANS:**

pre-order:     6, 3, 5, 10, 5, 23

in-order:      5, 3, 10, 6, 23, 5

post-order:    5, 10, 3, 23, 5, 6

For the remaining problems assume that a `BinaryTree` class has been defined with attributes `_value`, `_left`, and `_right`. The following getters have also been defined in the class:

```
def value(self):
    return self._value

def left(self):
    return self._left

def right(self):
    return self._right
```

Also, for reference, here is the code for an inorder traversal of a binary tree:

```
def inorder(tree):
    if tree == None:
        return
    else:
        inorder(tree.left())
        print(tree.value())
        inorder(tree.right())
```

3. Write a function `postorder(tree)` that prints the nodes of a tree in postorder.

**ANS:**

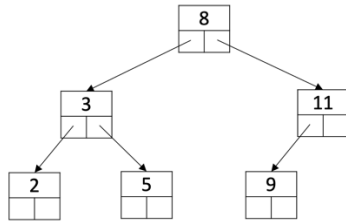
```
def postorder(tree):
    if tree == None:
        return
    else:
        postorder(tree.left())
        postorder(tree.right())
        print(tree.value())
```

4. Write a function `sum_leaves(tree)` that returns the sum of the values of the leaf nodes in the binary tree `tree`.

**ANS:**

```
def sum_leaves(tree):
    if tree == None:
        return 0
    if tree.left() == None and tree.right() == None:
        return tree.value()
    else:
        return sum_leaves(tree.left()) +
               sum_leaves(tree.right())
```

5. Write a function `inorder_str(tree)` that produces a *string* of the inorder traversal of the binary tree argument `tree`. For the tree below,



The string returned would be

"2,3,5,8,9,11"

**ANS:**

```
def inorder_str(t):
    if t == None:
        return ""

    instr = ""
    if t._left != None:
        instr += inorder_str(t._left) + "," + str(t._value)
    else:
        instr += str(t._value)
    if t._right != None:
        instr += "," + inorder_str(t._right)

    return instr
```