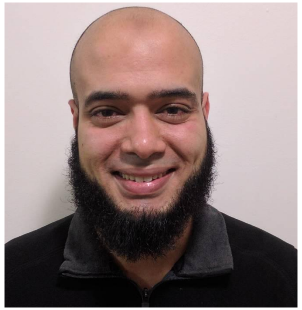
Data Structures

Binary Tree Traversal 3

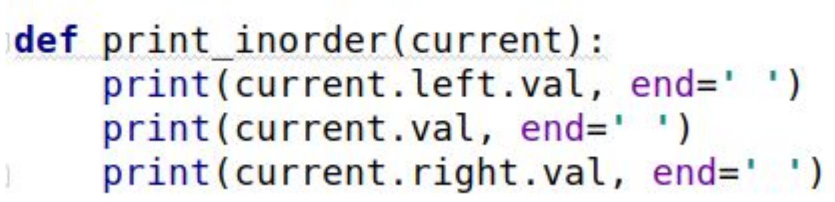
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

Artificial Intelligence & Computer Vision Researcher
PhD from Simon Fraser University - Canada
Bachelor / Msc from Cairo University - Egypt
Ex-(Software Engineer / ICPC World Finalist)

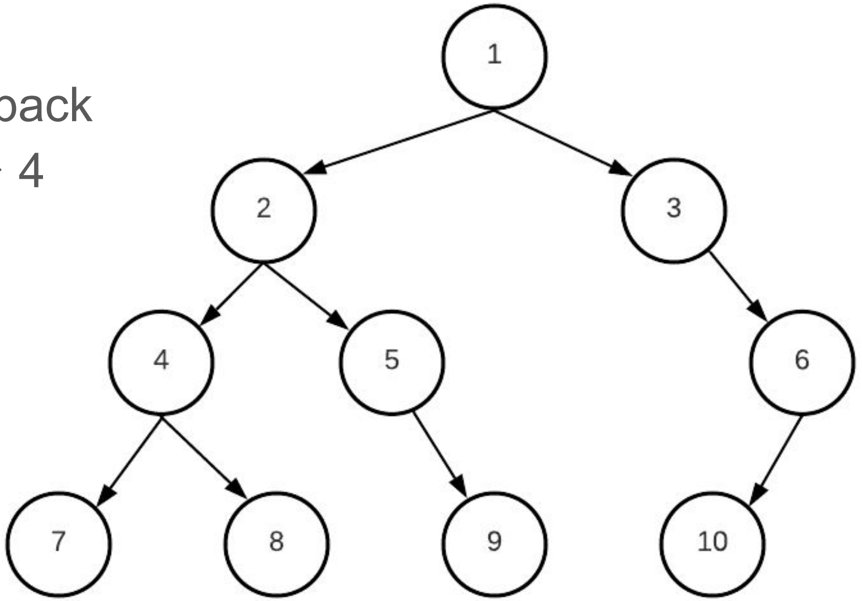


What is the in-order traversal?

- Observe the code keeps going to the **left**
- 1->2->4->7: No further left.
- At 7: **Print** data ⇒ 7. No right return. Go back
- At 4: left calls are complete. **Print** data ⇒ 4
- Go right to 8

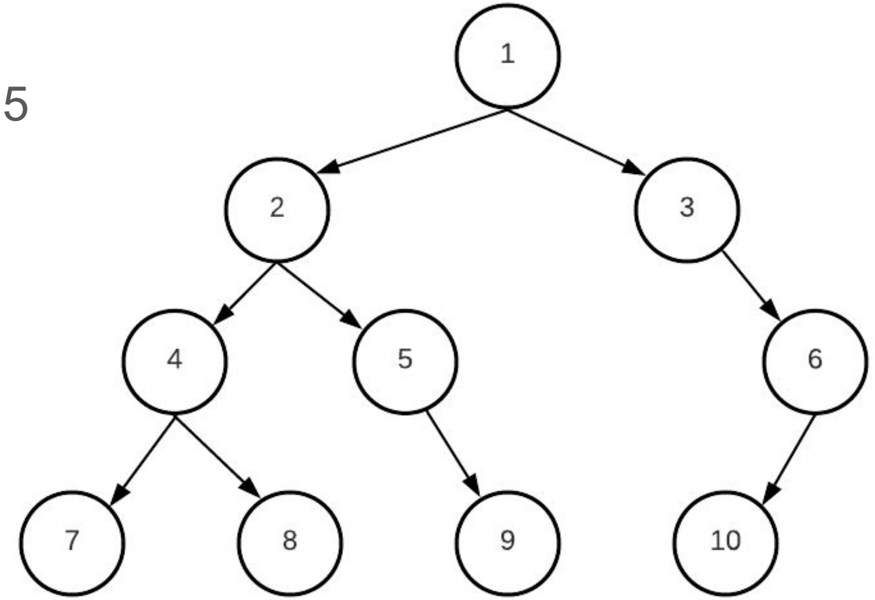


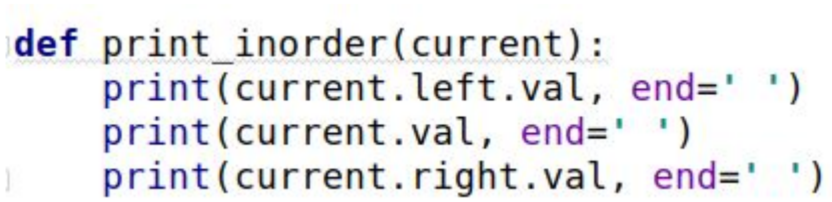
```python
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')
```



What is the in-order traversal?

- At 8: No left/right. **Print** data ⇒ 8. Go back
- At 4: left & right done. Go back.
- At 2: left done. **Print** data ⇒ 2. Go right: 5

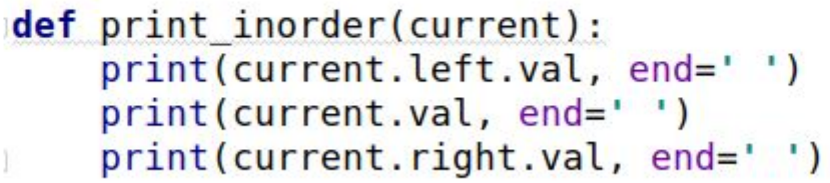




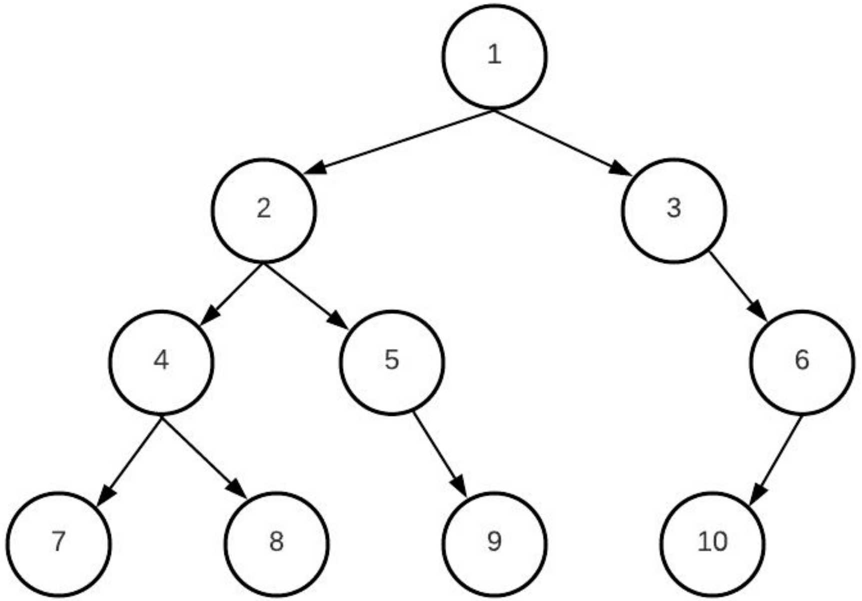
```python
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')
```

What is the in-order traversal?

- At 5: no left. **Print** data ⇒ 5 and Go right ⇒ 9
- At 9. **Print** 9 and go back
- At 5 left and right done: go back
- At 2 left and right done: go back
- At 1: left done. **Print** 1. Go right at 3

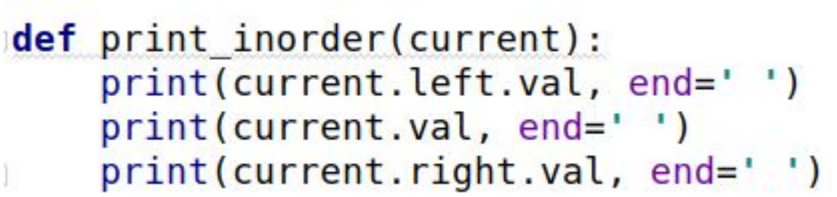


```
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')
```

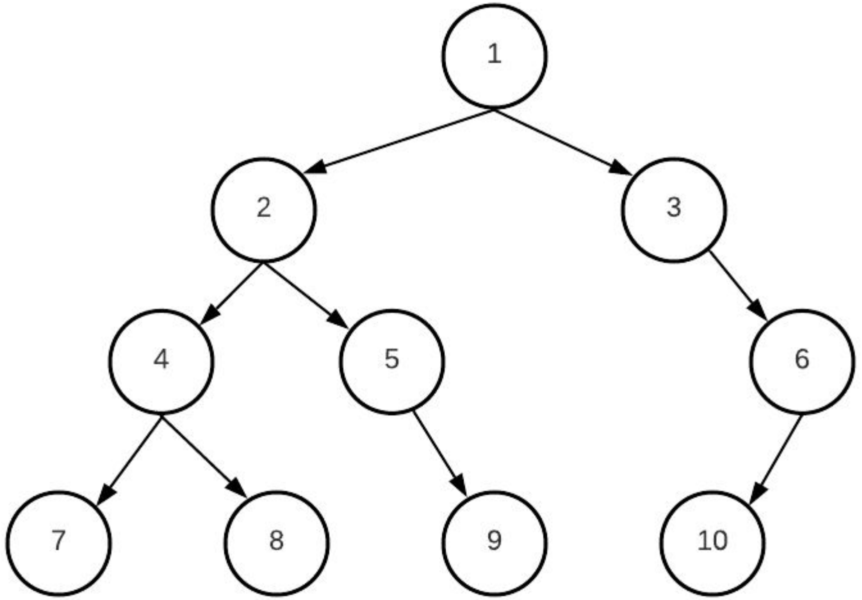


What is the in-order traversal?

- At 3: no left. **Print** 3. Go right at 6
- At 6: Go left at 10
- At 10: no left/right. **Print** 10. Go back to parent (6)
- At 6: left done. **Print** 6. No right.
- Go parent. Go parent. Done.
- 7 4 8 2 5 9 1 3 10 6

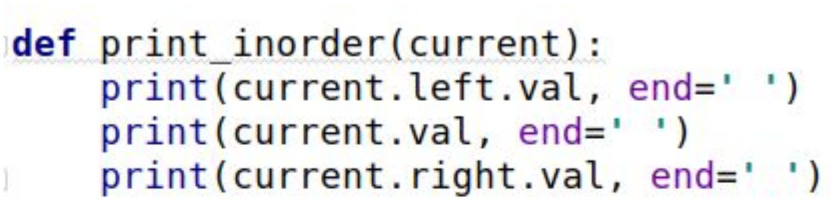


```python
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')
```

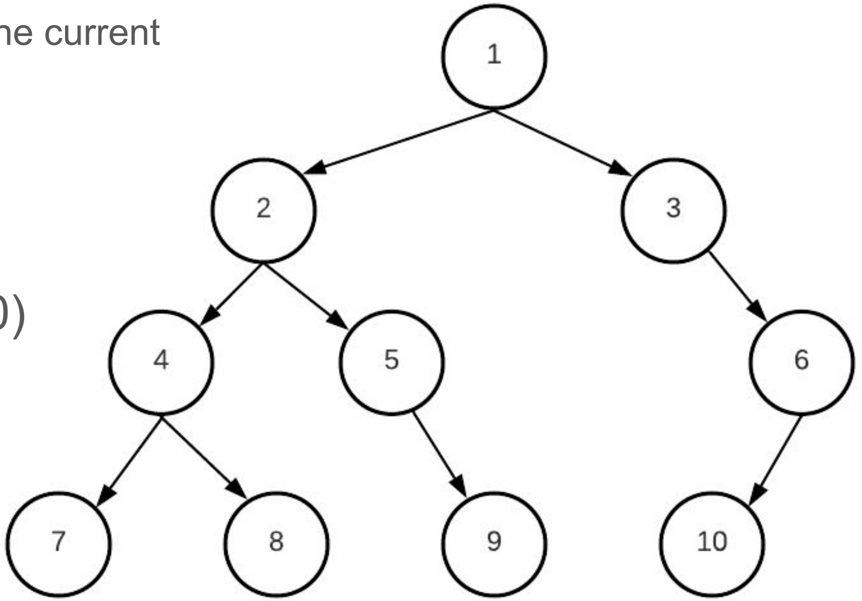


What is the in-order traversal?

- So from any node: keep going left
  - Once we have no left, or the left is done, print the current node
  - Go right and repeat
  - No right? Go parent
- **Most left node** is first printed: 7
- **Most right node** is last printed: 6 (not 10)

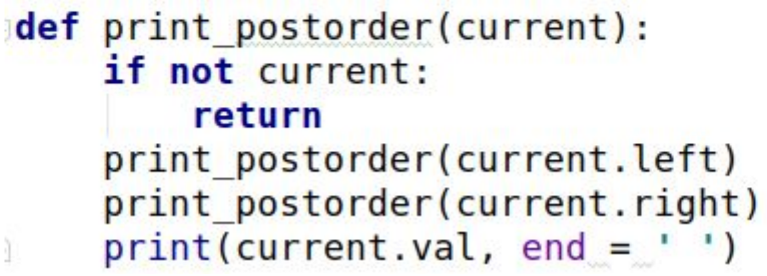


```
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')
```

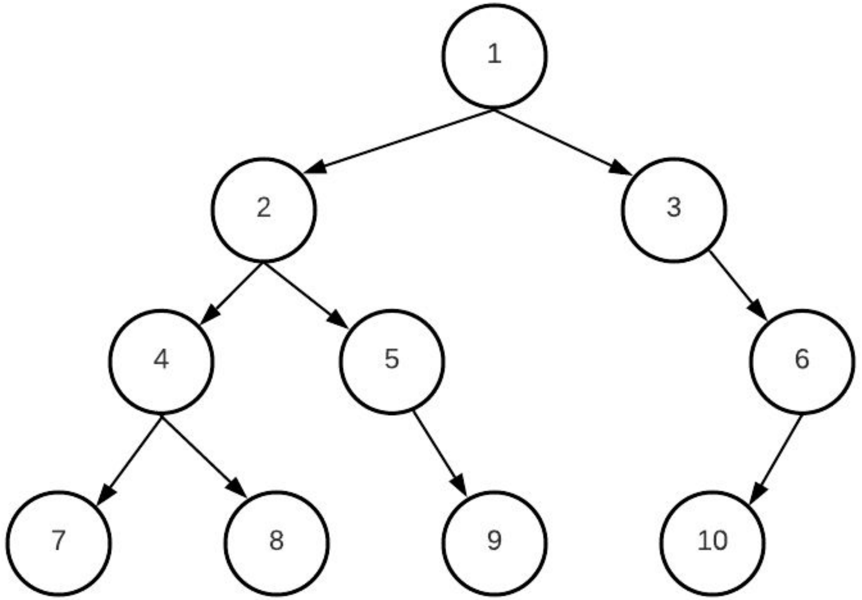


What is the post-order traversal?

- Observe the code keeps going to the **most left**
- 1->2->4->7: No further left.
- Then move to the right node
- Then, once again, find the left-most node
- Once no right or right done, print node
- Go back to the parent node

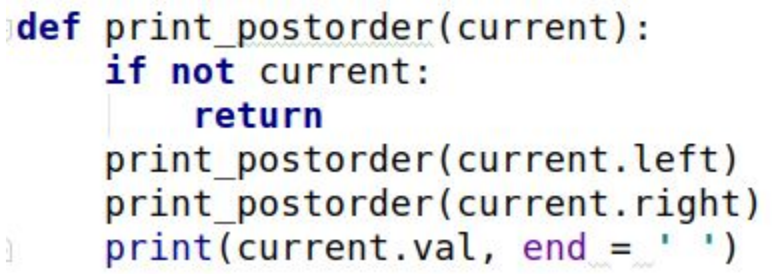


```
def print_postorder(current):
    if not current:
        return
    print_postorder(current.left)
    print_postorder(current.right)
    print(current.val, end = ' ')
```

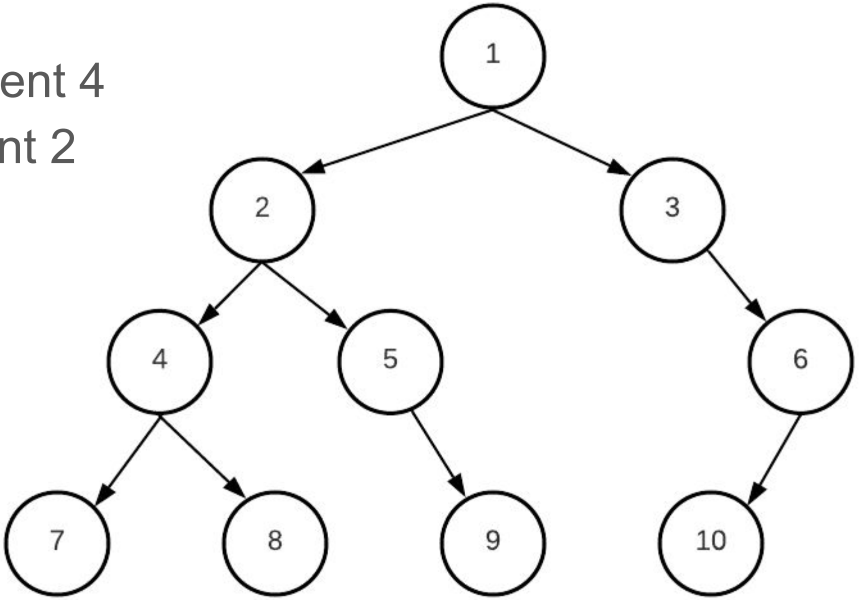


What is the post-order traversal?

- From 1 goes to 7. No right. **Print** 7.
- Go back to the parent at 4. Go right at 8
- At 8: no left/right. **Print** 8. Go back to parent 4
- At 4: right is done. **Print** 4. Go back parent 2
- At 2. Left done. Go right at 5.

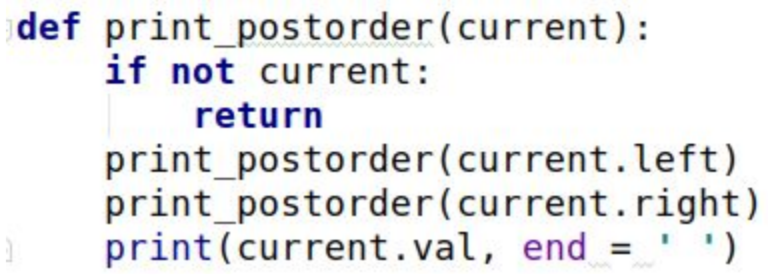


```
def print_postorder(current):
    if not current:
        return
    print_postorder(current.left)
    print_postorder(current.right)
    print(current.val, end = ' ')
```

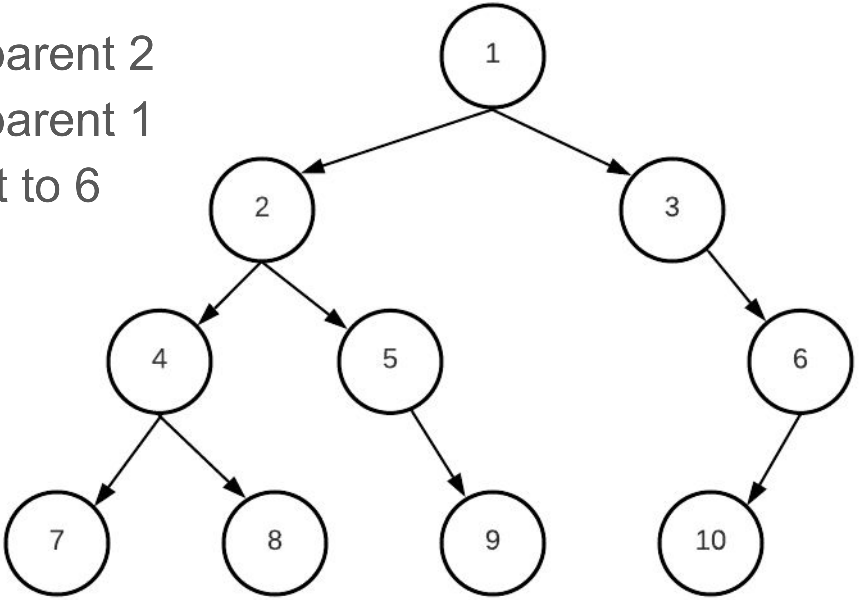


What is the post-order traversal?

- At 5: No left. Go right to 9
- At 9: **print** 9 and go back
- Back to 5: right done. **Print 5**. Go back parent 2
- Back to 2: right done. **Print 2**. Go back parent 1
- Back to 1: Go right at 3, which goes right to 6

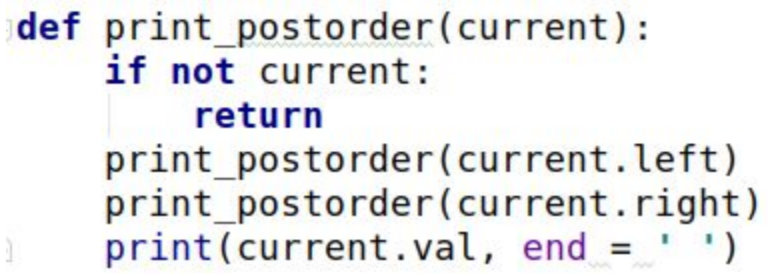


```
def print_postorder(current):
    if not current:
        return
    print_postorder(current.left)
    print_postorder(current.right)
    print(current.val, end = ' ')
```

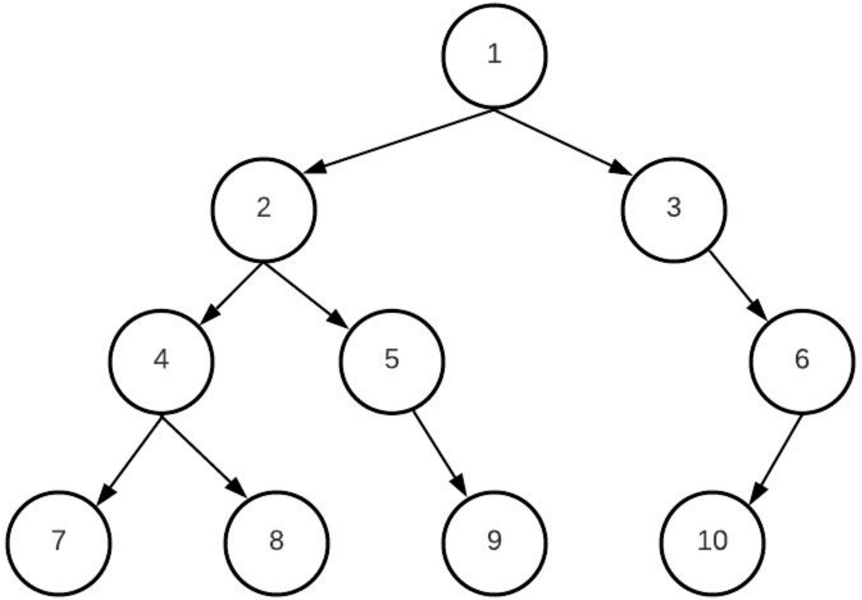


What is the post-order traversal?

- At 6: go left to 10. **Print 10** and go back
- **Print 6, print 3, print 1**
- In total: 7 8 4 9 5 2 10 6 3 1
- The root is the last printed value!

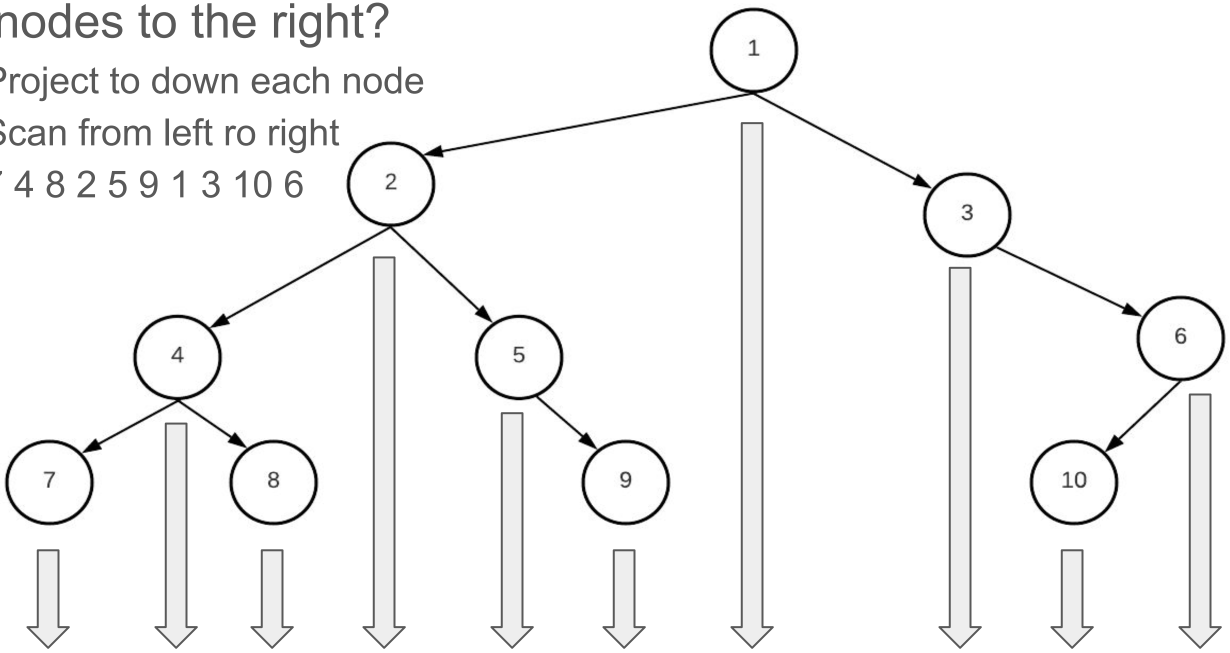


```
def print_postorder(current):
    if not current:
        return
    print_postorder(current.left)
    print_postorder(current.right)
    print(current.val, end = ' ')
```



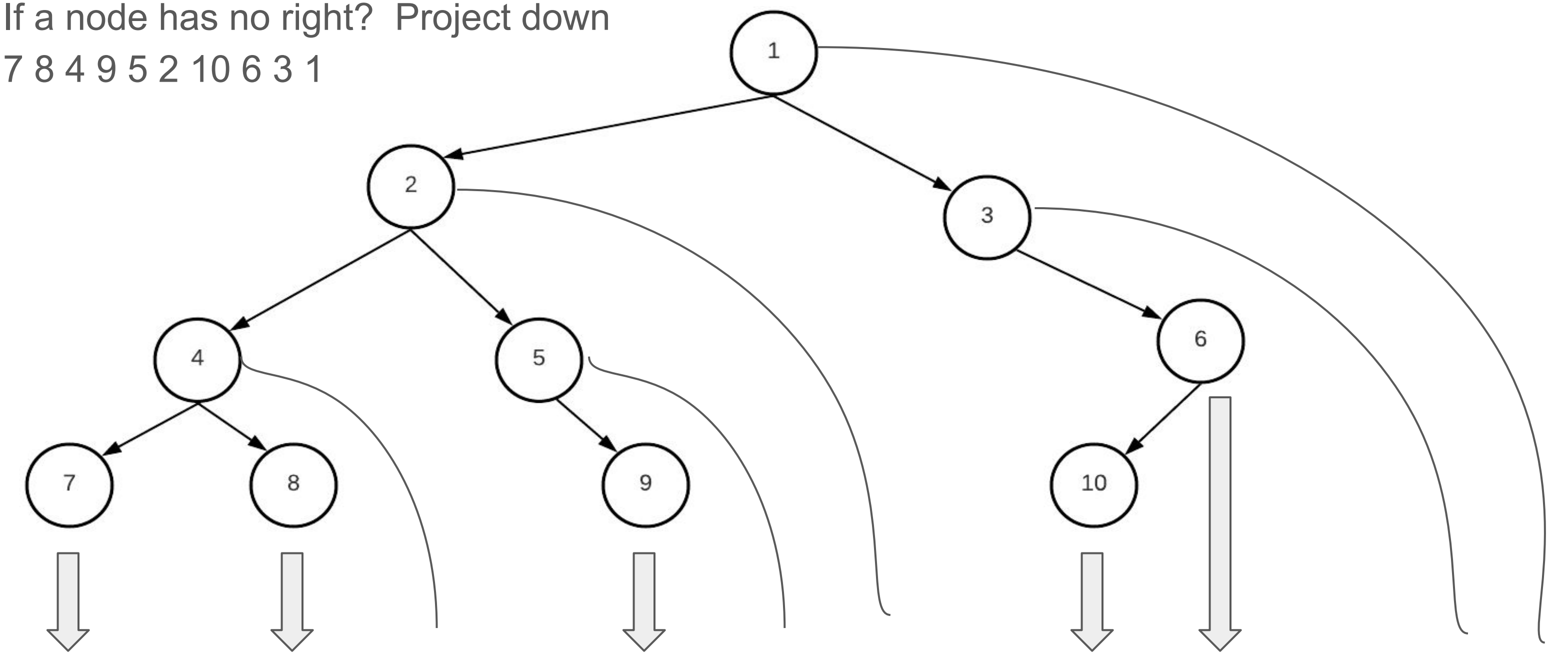
What is the in-order traversal? **Visually**

- What happens if you draw it out; so that all left nodes are to the left, and all right nodes to the right?
  - Project to down each node
  - Scan from left ro right
  - 7 4 8 2 5 9 1 3 10 6



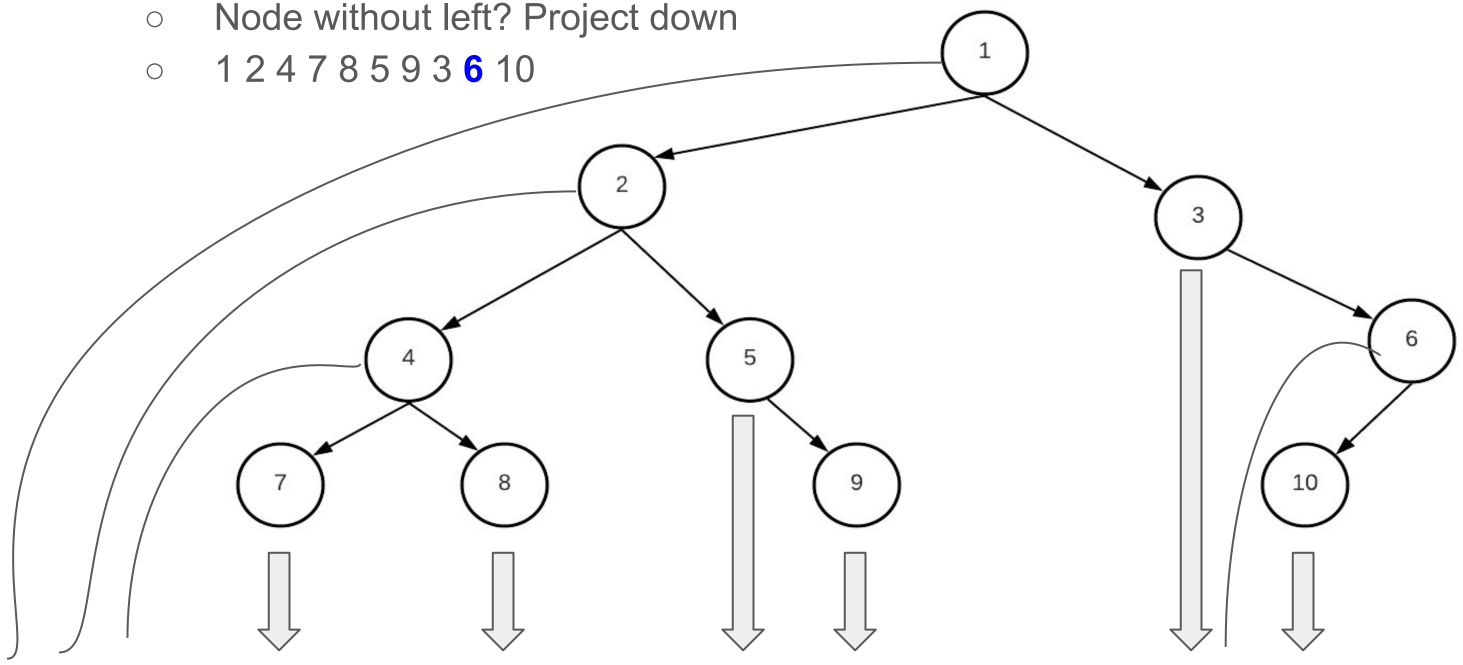
What is the post-order traversal? **Visually**

- Project each node **after its right subtree**
    - If a node has no right? Project down
    - 7 8 4 9 5 2 10 6 3 1



What is the pre-order traversal? **Visually**

- Project each node **before its left subtree**
  - Node without left? Project down
  - 1 2 4 7 8 5 9 3 **6** 10



Computations

- Most tasks follow one of these traversal strategies
  - Find minimum value of a tree
  - Find height of a tree
  - Count how many leaf or non-leaf nodes
  - Etc
- In all of them you need to follow some style. Go preorder: **VLR**
  - *Proper basecase handling*
  - *Compute something based on current->data*
  - *Compute the left subtree recursively*
  - *Compute the right subtree recursively*
  - *Compute the overall of these **3 values***
  - Examples in homework

Computations

- For some tasks, we might easily compute inorder traversal and check
    - Save the traversal in an array
    - Do the operation if applicable
        - Tree sum, min, max, if a value exists, if the tree has duplicate values
    - The major downside of this is that we must traverse the whole tree, i.e. it is an inefficient means by which to search for a specific value

"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."