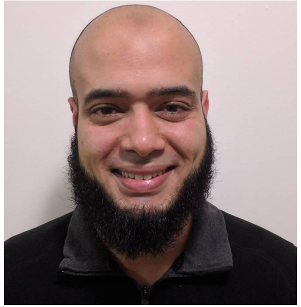
Data Structures

Binary Tree Traversal 1

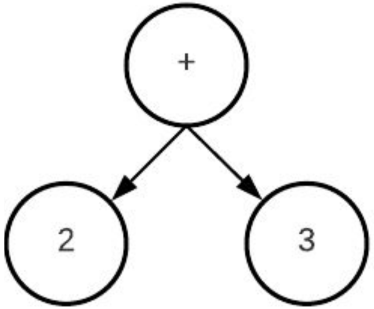
Mostafa S. Ibrahim
Teaching, Training and Coaching since more than a decade!

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



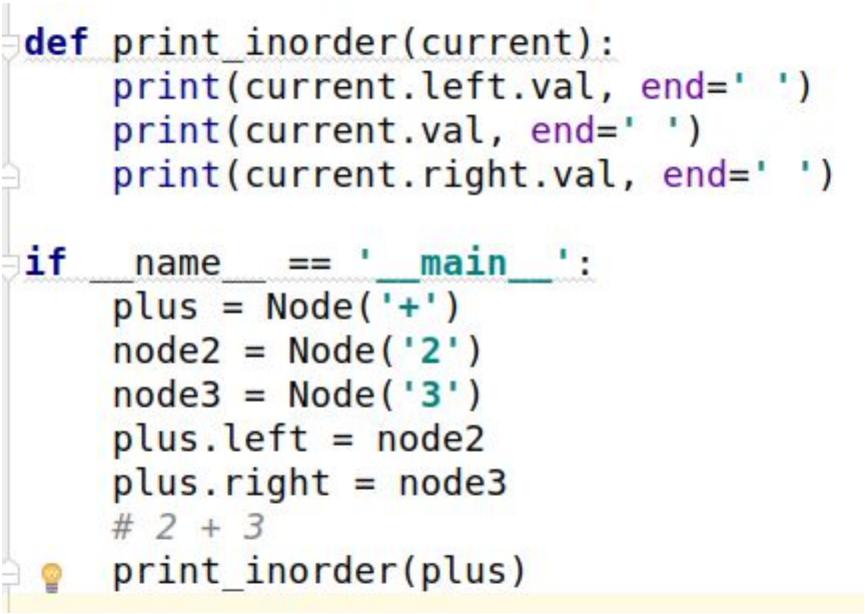
Tree Traversal

- **Traversal** Terminology: Walk through the elements of a data structure.
- We want to implement: def print(current_node)
  - Goal: print out the entire content of a tree or subtree in a systematic way, starting from the 'current' node (which is usually the root of that tree or subtree)
- Let's create an **Expression Tree** (leaves have operand values, non-leaves contain operators)
  - The diagram represents 2 + 3
  - We can draw complex expressions: e.g. (2+3)*4
  - For now, assume we have a **simple 2-level tree**:
    - Try to implement a print function
    - It should print: **2 + 3**

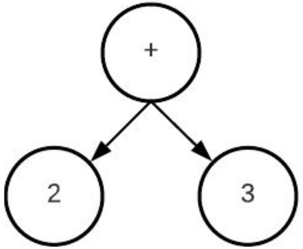


Print Expression Tree: 2 + 3

- A print function simply prints off the value of the **L**eft node, then the **V**alue of the 'current' node, then the value of the **R**ight nodede value, then myself then right node value
- Let’s call that LVR
  - L = left subtree (2)
  - V = Current node value (+)
  - R = right subtree (3)
  - This is inorder traversal
    - V = in the middle

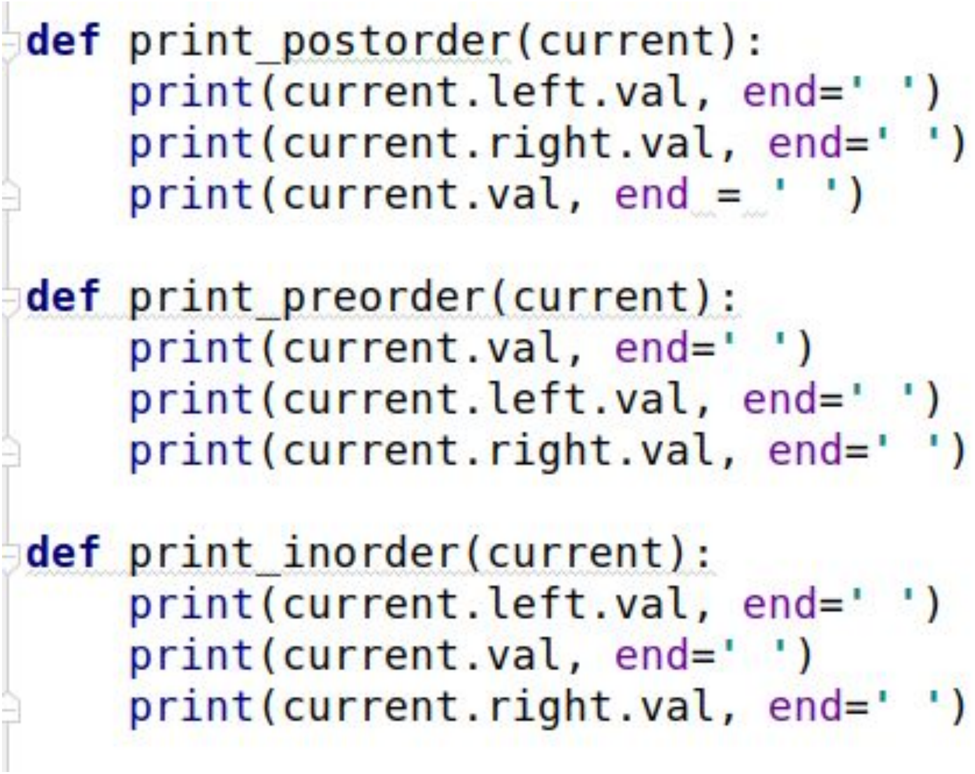


```python
def print_inorder(current):
    print(current.left.val, end=' ')
    print(current.val, end=' ')
    print(current.right.val, end=' ')

if __name__ == '__main__':
    plus = Node('+')
    node2 = Node('2')
    node3 = Node('3')
    plus.left = node2
    plus.right = node3
    # 2 + 3
    print_inorder(plus)
```



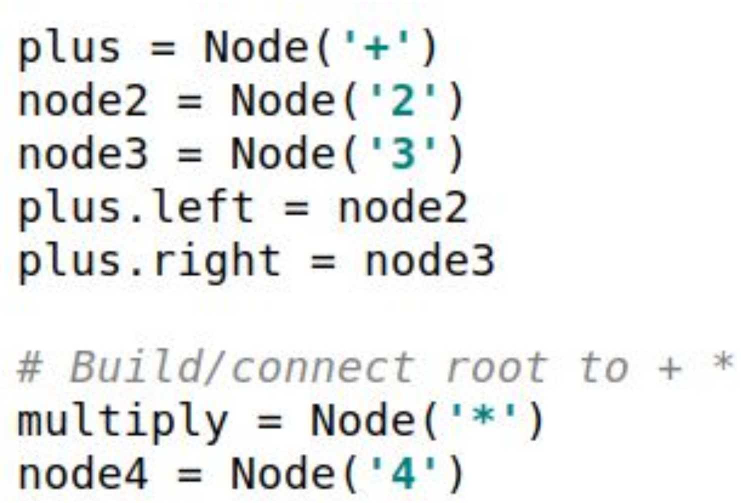
Print Expression Tree: 2 + 3

- Depending on when/where the current node value is printed, we can have:
  - 2 + 3 [in-order = infix]
  - 2 3 + [post-order = postfix]
  - + 2 3 [pre-order = prefix]
- We can summarize this as:
  - In-order = LVR
  - Post-order = LRV
  - Pre-order = VLR
- Other variants are not useful
  - LRV, RLV, VRL

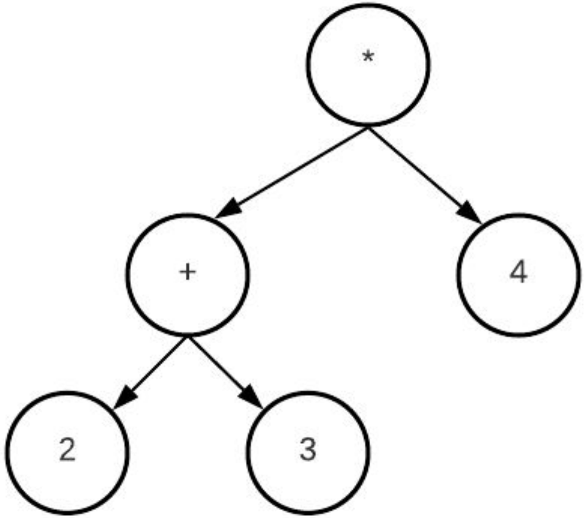


```python
def print_postorder(current):
    print(current.left.val, end=' ')
    print(current.right.val, end=' ')
    print(current.val, end = ' ')

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

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

Print Expression Tree: (2 + 3) * 4

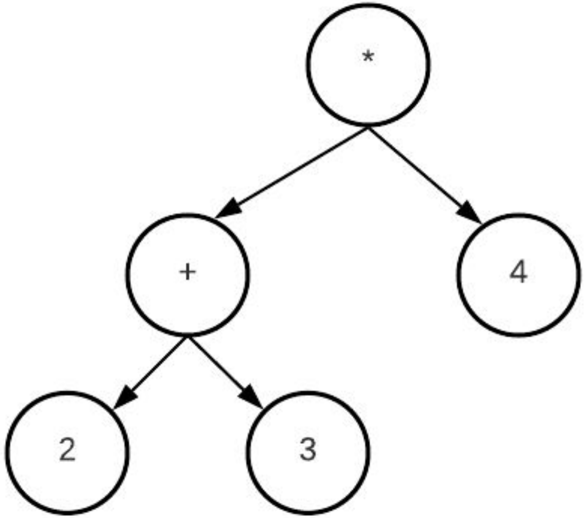


```
plus = Node('+')
node2 = Node('2')
node3 = Node('3')
plus.left = node2
plus.right = node3

# Build/connect root to + *
multiply = Node('*')
node4 = Node('4')
```



Print Expression Tree: (2 + 3) * 4

- How can we print such a complex tree in **post-order**?
- We know the right subtree is 23+
- We need recursive thinking here!
- Instead of printing out the 'left' value, we need to print out the left sub-tree



"Acquire knowledge and impart it to the people."

"Seek knowledge from the Cradle to the Grave."