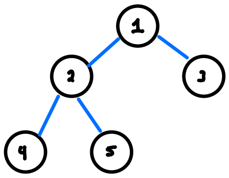
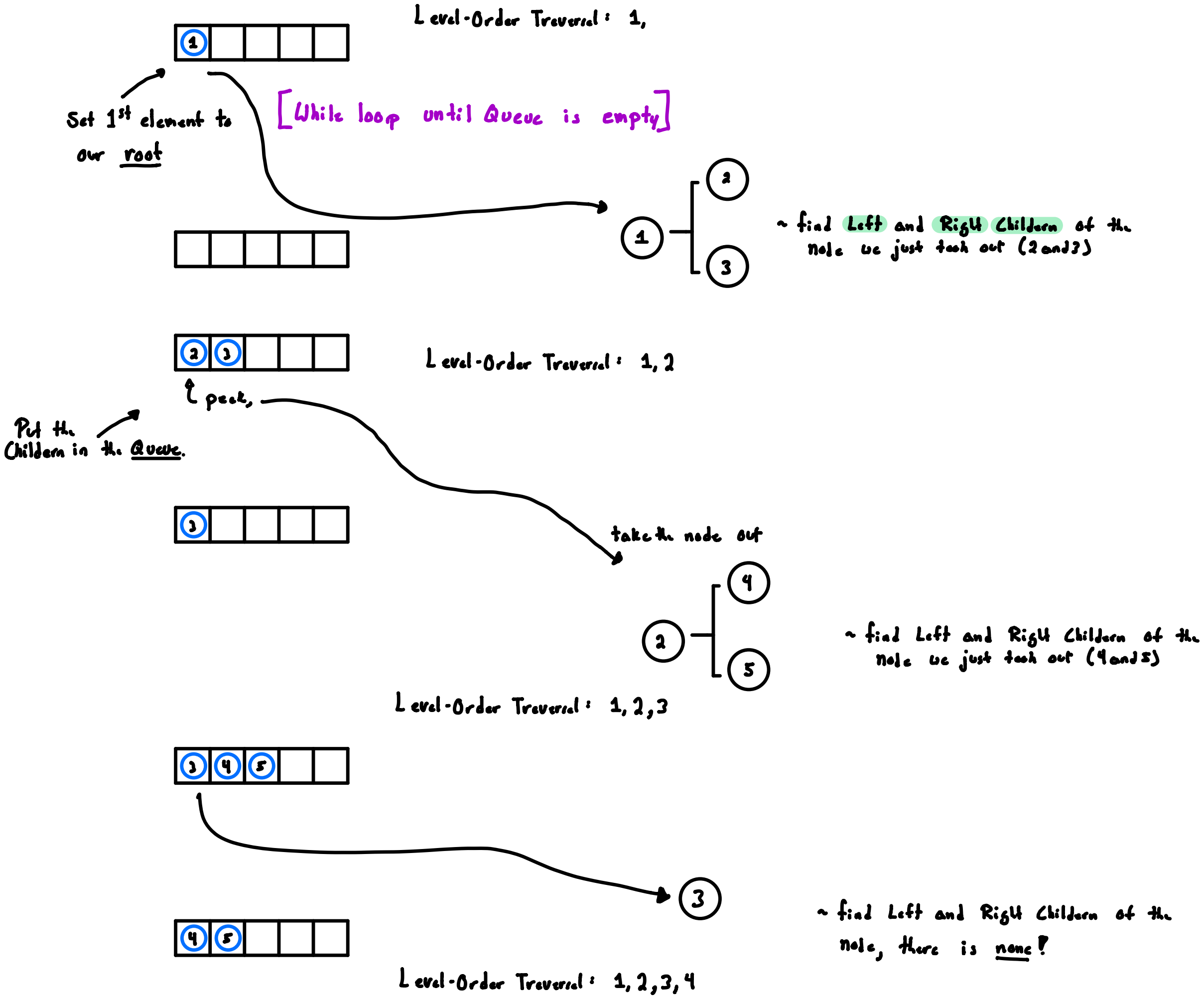
Binary Trees in Python: Level-order Traversal

- Level-by-Level



Level-Order Traversal: 1,2,3,4,5

Using Queue Data Structure



Level-Order Traversal: 1,

Set 1st element to our root

[While loop until Queue is empty]

~ find Left and Right Childern of the node we just took out (2 and 3)

Level-Order Traversal: 1, 2

↑ peek,

Put the Childern in the Queue.

take the node out

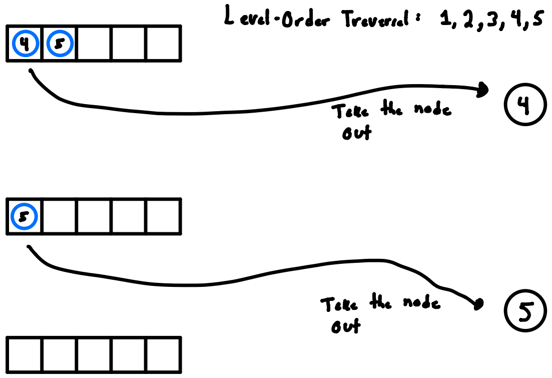
~ find Left and Right Childern of the node we just took out (4 and 5)

Level-Order Traversal: 1, 2, 3

~ find Left and Right Childern of the node, there is none!

Level-Order Traversal: 1, 2, 3, 4

Level-Order Traversal: 1, 2, 3, 4, 5

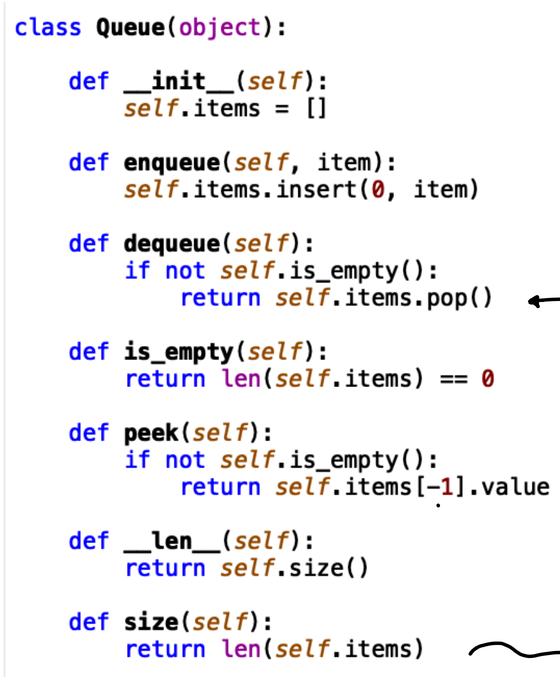


~ find Left and Right Children of the node, there is none!

~ find Left and Right Children of the node, there is none!

- loop until the queue is empty.

Creating a Queue to Use



```python
class Queue(object):

    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop()

    def is_empty(self):
        return len(self.items) == 0

    def peek(self):
        if not self.is_empty():
            return self.items[-1].value

    def __len__(self):
        return self.size()

    def size(self):
        return len(self.items)
```

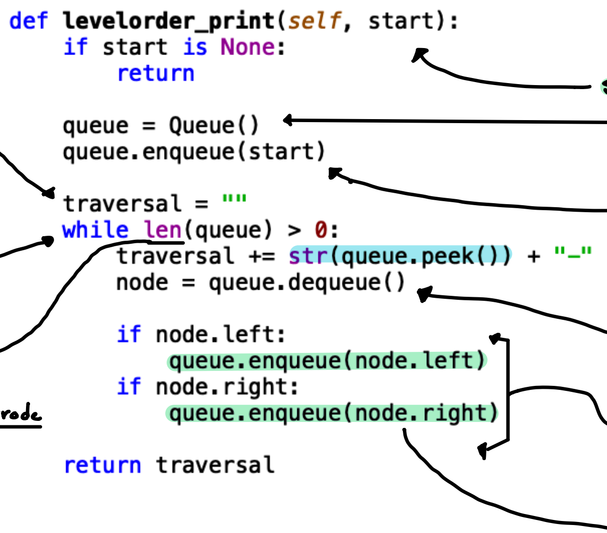
← So we can see what that node is

items in Queue

"Overriding functions"

Might not need these

extra functionality of the Class



```python
def levelorder_print(self, start):
    if start is None:
        return

    queue = Queue()
    queue.enqueue(start)

    traversal = ""
    while len(queue) > 0:
        traversal += str(queue.peek()) + "-"
        node = queue.dequeue()

        if node.left:
            queue.enqueue(node.left)
        if node.right:
            queue.enqueue(node.right)

    return traversal
```

Start from 1st Node in Tree

Defining a Queue object of the Class we just made

Added root element as the 1st element in the Queue

level Order

While loop Starts

Using len function that we overrode

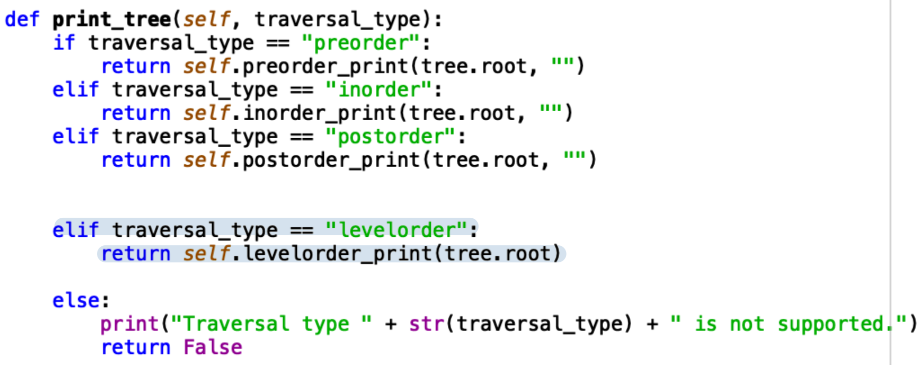
Taking Node out of Queue

Check Childern of the node we just dequeued from the Queue.

If the node has Childern, then we want to add them to our Queue

So we can perform length on an object of type Queue that we just defined, because we defined how the length function should operate on a type object that is of type Queue.

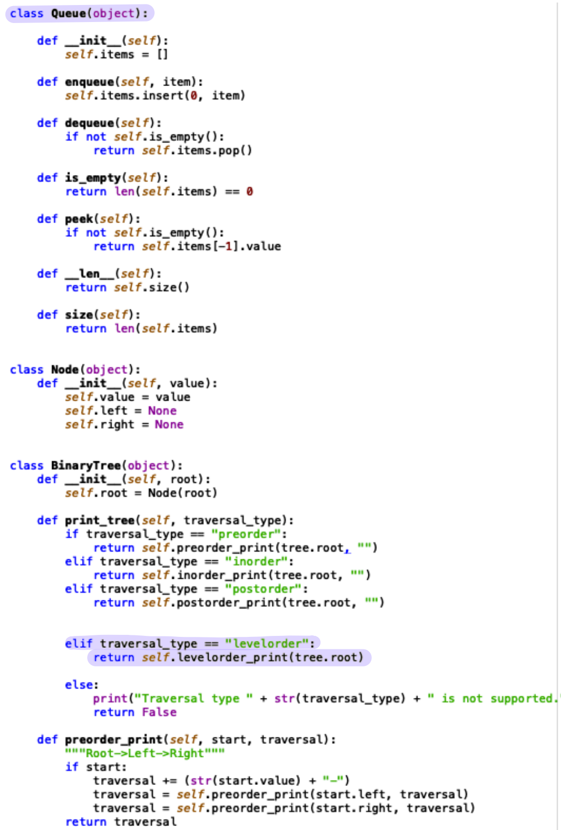
Added level Order traversal to Our Print function



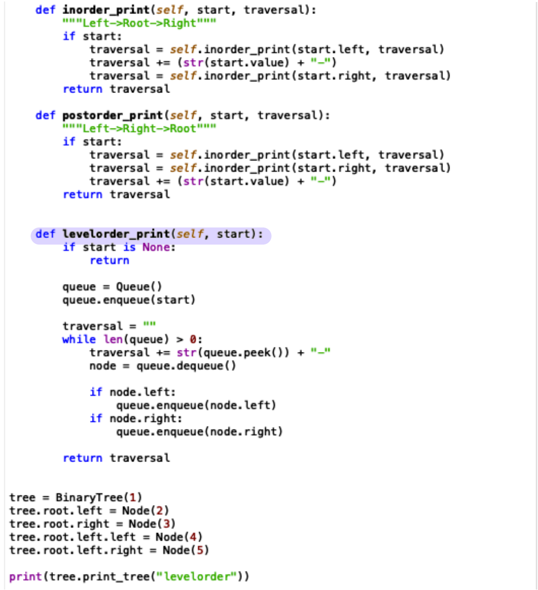
```python
def print_tree(self, traversal_type):
    if traversal_type == "preorder":
        return self.preorder_print(tree.root, "")
    elif traversal_type == "inorder":
        return self.inorder_print(tree.root, "")
    elif traversal_type == "postorder":
        return self.postorder_print(tree.root, "")


    elif traversal_type == "levelorder":
        return self.levelorder_print(tree.root)

    else:
        print("Traversal type " + str(traversal_type) + " is not supported.")
        return False
```

Full Code:

NEW



```python
class Queue(object):

    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.insert(0, item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop()

    def is_empty(self):
        return len(self.items) == 0

    def peek(self):
        if not self.is_empty():
            return self.items[-1].value

    def __len__(self):
        return self.size()

    def size(self):
        return len(self.items)


class Node(object):
    def __init__(self, value):
        self.value = value
        self.left = None
        self.right = None


class BinaryTree(object):
    def __init__(self, root):
        self.root = Node(root)

    def print_tree(self, traversal_type):
        if traversal_type == "preorder":
            return self.preorder_print(tree.root, "")
        elif traversal_type == "inorder":
            return self.inorder_print(tree.root, "")
        elif traversal_type == "postorder":
            return self.postorder_print(tree.root, "")


        elif traversal_type == "levelorder":
            return self.levelorder_print(tree.root)

        else:
            print("Traversal type " + str(traversal_type) + " is not supported.")
            return False

    def preorder_print(self, start, traversal):
        """Root->Left->Right"""
        if start:
            traversal += (str(start.value) + "-")
            traversal = self.preorder_print(start.left, traversal)
            traversal = self.preorder_print(start.right, traversal)
        return traversal
```



```python
    def inorder_print(self, start, traversal):
        """Left->Root->Right"""
        if start:
            traversal = self.inorder_print(start.left, traversal)
            traversal += (str(start.value) + "-")
            traversal = self.inorder_print(start.right, traversal)
        return traversal

    def postorder_print(self, start, traversal):
        """Left->Right->Root"""
        if start:
            traversal = self.inorder_print(start.left, traversal)
            traversal = self.inorder_print(start.right, traversal)
            traversal += (str(start.value) + "-")
        return traversal


    def levelorder_print(self, start):
        if start is None:
            return

        queue = Queue()
        queue.enqueue(start)

        traversal = ""
        while len(queue) > 0:
            traversal += str(queue.peek()) + "-"
            node = queue.dequeue()

            if node.left:
                queue.enqueue(node.left)
            if node.right:
                queue.enqueue(node.right)

        return traversal


tree = BinaryTree(1)
tree.root.left = Node(2)
tree.root.right = Node(3)
tree.root.left.left = Node(4)
tree.root.left.right = Node(5)

print(tree.print_tree("levelorder"))
```

1-2-3-4-5-