Implementing a Binary Search Tree in Python

In this lesson, we'll implement a very basic Binary Search Tree in Python

We'll cover the following

- Introduction
  - The Node Class
  - The BinarySearchTree class
  - Putting the two together

Introduction

The Node Class

To implement a BST, the first thing you'd need is a node. A node should have a value, a left child, a right child, and a parent. This node can be implemented as a Python class and here is the code.

Node.py

```
class Node:
    def __init__(self, val):  # Constructor to initialize the value of the n
        self.val = val
        self.leftChild = None  # Sets the left and right children to `None`
        self.rightChild = None
        self.parent = None  # Sets the parent to `None`

```

The `BinarySearchTree` class

You can then choose to create a wrapper class for the tree itself; this can sometimes make your code cleaner and easier to read, but not always. However, this is a programming convention so let's create a tree class:

BinarySearchTree.py

```
class BinarySearchTree:
    def __init__(self, val):  # Initializes a root node
        self.root = Node(val)

```

Putting the two together

When both classes are put together, you get a BST. Let's try running this.

BinarySearchTree.py

Node.py

```
from Node import Node  # use `Node` class from Node.py


class BinarySearchTree:
    def __init__(self, val):  # Initializes a root node
        self.root = Node(val)


BST = BinarySearchTree(6)  # Initializes a BST
print(BST.root.val)  # print value of root node
```

Now that we have some bare bones code for binary search tr et k a high-level algorithm and code to insert values into a BST!

Back

What is a Binary Search Tree (BST)?

Next

Binary Search Tree Insertion

Completed

Report an Issue