Binary Search Trees

3 Different Ways

```
Way 1
                                                                                                     Way 2
class Node:
                                                                             class Node:
    def __init__(self, d):
                                                                                 def __init__(self, d):
                                                                                     self.data = d
        self.data = d
                                                                                     self.left = None
        self.left = None
        self.right = None
                                                                                     self.right = None
                                                                                 def insert(self, d):
class BinarySearchTree:
                                                                                     if d <= self.data:</pre>
                                                                                         if self.left is None:
    def __init__(self, d):
        self.root = Node(d)
                                                                                             self.left = Node(d)
                                                                                         else:
    def insert(self, node, d):
                                                                                             self.left.insert(d)
        if d <= node.data:</pre>
                                                                                     else:
            if node.left is None:
                                                                                         if self.right is None:
                node.left = Node(d)
                                                                                              self.right = Node(d)
                                                                                         else:
                self.insert(node.left, d)
                                                                                             self.right.insert(d)
                                                                                 def search(self, d):
            if node.right is None:
                                                                                     if d == self.data:
                node.right = Node(d)
                                                                                         return True
            else:
                self.insert(node.right, d)
                                                                                     if d < self.data:</pre>
                                                                                         if self.left is None:
                                                                                             return False
    def search(self, node, d):
        if d == node.data:
                                                                                              return self.left.search(d)
            return True
        if d < node.data:</pre>
                                                                                     else:
            if node.left is None:
                                                                                         if self.right is None:
                return False
                                                                                             return False
                                                                                         else:
            else:
                                                                                             return self.right.search(d)
                return self.search(node.left, d)
        else:
                                                                                 def preorder(self):
            if node.right is None:
                                                                                     print(self.data)
                return False
                                                                                     if self.left is not None:
                                                                                         self.left.preorder()
                return self.search(node.right, d)
                                                                                     if self.right is not None:
    def preorder(self, node):
                                                                                         self.right.preorder()
        if node is not None:
            print(node.data)
                                                                                 def inorder(self):
                                                                                     if self.left is not None:
            self.preorder(node.left)
                                                                                         self.left.inorder()
            self.preorder(node.right)
                                                                                     print(self.data)
                                                                                     if self.right is not None:
    def inorder(self, node):
        if node is not None:
                                                                                         self.right.inorder()
            self.inorder(node.left)
            print(node.data)
                                                                                 def postorder(self):
            self.inorder(node.right)
                                                                                     if self.left is not None:
                                                                                         self.left.postorder()
    def postorder(self, node):
                                                                                     if self.right is not None:
                                                                                         self.right.postorder()
        if node is not None:
                                                                                     print(self.data)
            self.postorder(node.left)
            self.postorder(node.right)
                                                                                 def delete(self, d):
            print(node.data)
                                                                                     if d == self.data:
                                                                                         if self.left is None:
    def delete(self, node, d):
        if node is None:
                                                                                             return self.right
                                                                                         elif self.right is None:
            return node
        if d == node.data:
                                                                                             return self.left
            if node.left is None:
                                                                                         else:
                                                                                             left_tree = self.left
                return node.right
                                                                                              right_tree = self.right
            elif node.right is None:
                return node.left
                                                                                              cur = right_tree
            else:
                                                                                             if cur.left is not None:
                left_tree = node.left
                                                                                                  while cur.left is not None:
                                                                                                      parent = cur
                right_tree = node.right
                cur = right_tree
                                                                                                      cur = cur.left
                                                                                                  parent.left = cur.delete(cur.data)
                if cur.left is not None:
                     while cur.left is not None:
                                                                                                  cur.right = right_tree
                         parent = cur
                                                                                             cur.left = left_tree
                                                                                             return cur
                         cur = cur.left
                     parent.left = self.delete(cur, cur.data)
                                                                                     if d < self.data:</pre>
                                                                                         self.left = self.left.delete(d)
                     cur.right = right_tree
                cur.left = left_tree
                                                                                         self.right = self.right.delete(d)
                return cur
        if d < node.data:</pre>
                                                                                     return self
            node.left = self.delete(node.left, d)
                                                                                 def __display(self):...
            node.right = self.delete(node.right, d)
                                                                                 def __str__(self):...
    def __display(self, node):...
                                                                             bst = Node(10)
                                                                             bst.insert(5)
    def __str__(self):...
                                                                             bst.insert(15)
                                                                             bst.preorder()
bst = BinarySearchTree(10)
                                                                             bst.inorder()
bst.insert(bst.root, 5)
                                                                             bst.postorder()
                                                                             bst = bst.delete(5)
bst.insert(bst.root, 15)
                                                                             print(bst.search(5))
bst.preorder(bst.root)
bst.inorder(bst.root)
                                                                             print(bst)
bst.postorder(bst.root)
bst.root = bst.delete(bst.root, 5)
print(bst.search(bst.root, 5))
print(bst)
                                                                              Pros:
                                                                              Code is cleaner
                                                                              If you delete the root you will have to
Pros:
                                                                              create a new BST if you want to insert
Keeps track of root node.
                                                                              new items.
No problems with deleting the root
node and then inserting new Items.
                                                                              You can solve this by making it so you
                                                                              can't delete the root node.
Code is ugly
Usability is ugly
```

You can add extra code to make the usability of the code look nicer by creating methods that pass the root to

other methods for you.

Way 1 Way 2 Way 3

```
class Node:
    def __init__(self, d):
        self.data = d
        self.left = None
        self.right = None
class BinarySearchTree:
    def __init__(self, d):
        self.root = Node(d)
    def insert(self, node, d):
        if d <= node.data:</pre>
            if node.left is None:
                node.left = Node(d)
            else:
                self.insert(node.left, d)
        else:
            if node.right is None:
                node.right = Node(d)
                self.insert(node.right, d)
    def search(self, node, d):
        if d == node.data:
            return True
        if d < node.data:</pre>
            if node.left is None:
                return False
            else:
                return self.search(node.left, d)
        else:
            if node.right is None:
                return False
            else:
                return self.search(node.right, d)
    def preorder(self, node):
        if node is not None:
            print(node.data)
            self.preorder(node.left)
            self.preorder(node.right)
    def inorder(self, node):
        if node is not None:
            self.inorder(node.left)
            print(node.data)
            self.inorder(node.right)
    def postorder(self, node):
        if node is not None:
            self.postorder(node.left)
            self.postorder(node.right)
            print(node.data)
    def delete(self, node, d):
        if node is None:
            return node
        if d == node.data:
            if node.left is None:
                return node.right
            elif node.right is None:
                return node.left
            else:
                left_tree = node.left
                right_tree = node.right
                cur = right_tree
                if cur.left is not None:
                    while cur.left is not None:
                        parent = cur
                        cur = cur.left
                    parent.left = self.delete(cur, cur.data)
                    cur.right = right_tree
                cur.left = left_tree
                return cur
        if d < node.data:</pre>
            node.left = self.delete(node.left, d)
        else:
            node.right = self.delete(node.right, d)
        return node
    def __display(self, node):...
    def __str__(self):...
bst = BinarySearchTree(10)
bst.insert(bst.root, 5)
bst.insert(bst.root, 15)
bst.preorder(bst.root)
bst.inorder(bst.root)
bst.postorder(bst.root)
bst.root = bst.delete(bst.root, 5)
print(bst.search(bst.root, 5))
```

print(bst)

```
class Node:
    def __init__(self, d):
        self.data = d
        self.left = None
        self.right = None
    def insert(self, d):
        if d <= self.data:</pre>
            if self.left is None:
                self.left = Node(d)
                self.left.insert(d)
        else:
            if self.right is None:
                self.right = Node(d)
            else:
                self.right.insert(d)
    def search(self, d):
        if d == self.data:
             return True
        if d < self.data:</pre>
            if self.left is None:
                return False
            else:
                return self.left.search(d)
        else:
            if self.right is None:
                return False
                return self.right.search(d)
    def preorder(self):
        print(self.data)
        if self.left is not None:
            self.left.preorder()
        if self.right is not None:
            self.right.preorder()
    def inorder(self):
        if self.left is not None:
            self.left.inorder()
        print(self.data)
        if self.right is not None:
            self.right.inorder()
    def postorder(self):
        if self.left is not None:
            self.left.postorder()
        if self.right is not None:
            self.right.postorder()
        print(self.data)
    def delete(self, d):
        if d == self.data:
            if self.left is None:
                return self.right
            elif self.right is None:
                return self.left
            else:
                 left_tree = self.left
                right_tree = self.right
                cur = right_tree
                if cur.left is not None:
                    while cur.left is not None:
                        parent = cur
                         cur = cur.left
                     parent.left = cur.delete(cur.data)
                     cur.right = right_tree
                cur.left = left_tree
                return cur
        if d < self.data:</pre>
            self.left = self.left.delete(d)
        else:
            self.right = self.right.delete(d)
    def __display(self):...
    def __str__(self):...
bst = Node(10)
bst.insert(5)
bst.insert(15)
bst.preorder()
bst.inorder()
bst.postorder()
bst = bst.delete(5)
print(bst.search(5))
print(bst)
```

```
class Node:
     def __init__(self, d):
         self.data = d
         self.left = None
         self.right = None
     def insert(self, d):
          if d <= self.data:</pre>
              if self.left is None:
                  self.left = Node(d)
              else:
                  self.left.insert(d)
         else:
              if self.right is None:
                  self.right = Node(d)
              else:
                 self.right.insert(d)
     def search(self, d):
         if d == self.data:
              return True
         if d < self.data:</pre>
              if self.left is None:
                  return False
              else:
                  return self.left.search(d)
         else:
              if self.right is None:
                  return False
              else:
                  return self.right.search(d)
     def preorder(self):
          print(self.data)
          if self.left is not None:
              self.left.preorder()
         if self.right is not None:
             self.right.preorder()
     def inorder(self):
         if self.left is not None:
              self.left.inorder()
          print(self.data)
         if self.right is not None:
             self.right.inorder()
     def postorder(self):
          if self.left is not None:
              self.left.postorder()
          if self.right is not None:
              self.right.postorder()
         print(self.data)
     def delete(self, d):
         if d == self.data:
             if self.left is None:
                  return self.right
              elif self.right is None:
                  return self.left
              else:
                  left_tree = self.left
                  right_tree = self.right
                  cur = right_tree
                  if cur.left is not None:
                      while cur.left is not None:
                          parent = cur
                          cur = cur.left
                      parent.left = cur.delete(cur.data)
                      cur.right = right_tree
                  cur.left = left_tree
                  return cur
         if d < self.data:</pre>
              self.left = self.left.delete(d)
             self.right = self.right.delete(d)
          return self
     def display(self):...
 class BinarySearchTree:
     def __init__(self):
         self.root = None
     def insert(self, d):
          if self.root is None:
              self.root = Node(d)
          else:
             self.root.insert(d)
     def search(self, d):
         if self.root is not None:
              return self.root.search(d)
          else:
              return False
     def preorder(self):
          if self.root is not None:
             self.root.preorder()
     def inorder(self):
         if self.root is not None:
             self.root.inorder()
     def postorder(self):
          if self.root is not None:
             self.root.postorder()
     def delete(self, d):
         if self.root is not None:
             self.root = self.root.delete(d)
     def __str__(self):...
bst = BinarySearchTree()
bst.insert(10)
bst.insert(5)
bst.insert(15)
bst.preorder()
bst.inorder()
bst.postorder()
bst.delete(5)
```

print(bst.search(5))

print(bst)