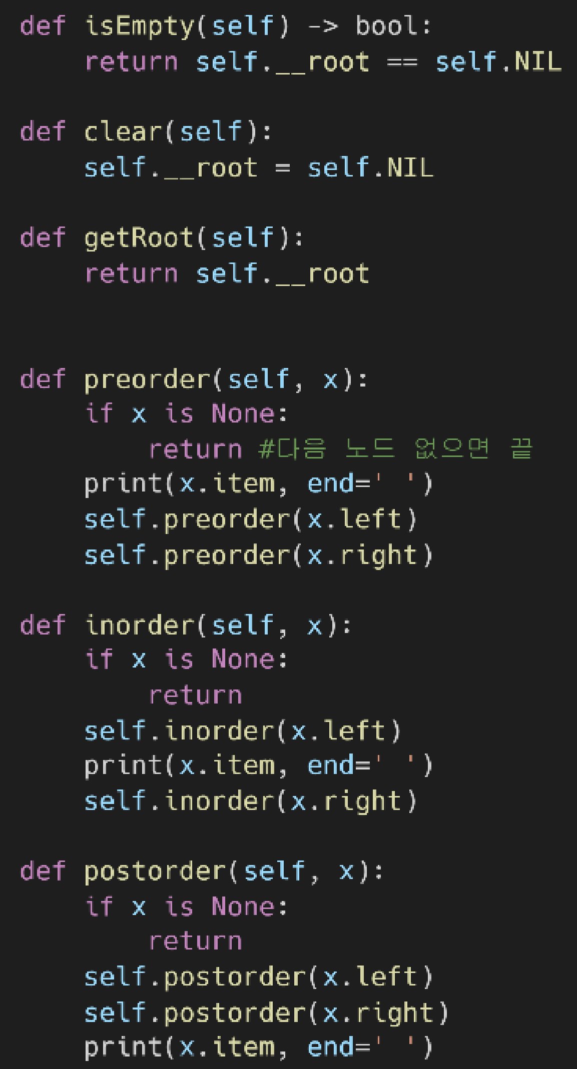
▼ binarySearchTree.py

```python
class TreeNode:
    def __init__(self, newItem, left, right):
        self.item = newItem
        self.left = left
        self.right = right

class BinarySearchTree:
    def __init__(self):
        self.__root = None

    def search(self, x) -> TreeNode:
        return self.__searchItem(self.__root, x)

    def __searchItem(self, tNode:TreeNode, x) -> TreeNode:
        if (tNode == None):
            return None
        elif (x == tNode.item):
            return tNode
        elif (x < tNode.item):
            return self.__searchItem(tNode.left, x)
        else:
            return self.__searchItem(tNode.right, x)


    def insert(self, newItem):
        self.__root = self.__insertItem(self.__root, newItem)

    def __insertItem(self, tNode:TreeNode, newItem) -> TreeNode:
        if (tNode == None):
            tNode = TreeNode(newItem, None, None)
        elif (newItem < tNode.item):  # left
            tNode.left = self.__insertItem(tNode.left, newItem)
        else: # right
            tNode.right = self.__insertItem(tNode.right, newItem)
        return tNode
```

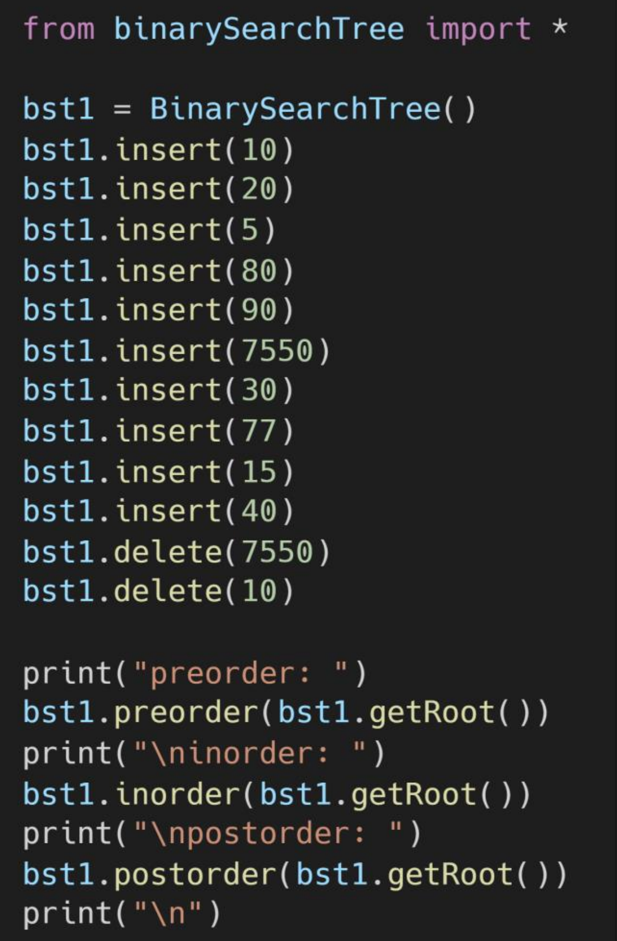
```python
    def delete(self, x):
        self.__root = self.__deleteItem(self.__root, x)

    def __deleteItem(self, tNode:TreeNode, x) -> TreeNode:
        if (tNode == None): # 못찾음
            return None
        elif (x == tNode.item): # 찾음
                tNode = self.__deleteNode(tNode)
        elif (x < tNode.item):
                tNode.left = self.__deleteItem(tNode.left, x)
        else:
                tNode.right = self.__deleteItem(tNode.right, x)
        return tNode # tNode: parent에 매달리는 노드

    def __deleteNode(self, tNode:TreeNode) -> TreeNode:
        # case1. tNode이 리프 노드
        # case2. tNode이 자식이 하나만 있음
        # case3. tNode이 자식이 둘 있음

        if tNode.left == None and tNode.right == None: # case 1(자식이 없음)
            return None
        elif tNode.left == None:  # case 2(rigt만 있음)
            return tNode.right
        elif tNode.right == None: # case 2(left만 있음)
            return tNode.left
        else: # case 3(right, left 둘 다 있음)
            (rtnItem, rtnNode) = self.__deleteMinItem(tNode.right)
            tNode.item = rtnItem
            tNode.right = rtnNode
            return tNode  # tNode survived

    def __deleteMinItem(self, tNode:TreeNode) -> tuple:
        if tNode.left == None:
            # found min at tNode
            return (tNode.item, tNode.right)
        else: # 작은 쪽(left)으로 이동
            (rtnItem, rtnNode) = self.__deleteMinItem(tNode.left)
            tNode.left = rtnNode
            return (rtnItem, tNode)
```



```python
def isEmpty(self) -> bool:
    return self.__root == self.NIL

def clear(self):
    self.__root = self.NIL

def getRoot(self):
    return self.__root


def preorder(self, x):
    if x is None:
        return #다음 노드 없으면 끝
    print(x.item, end=' ')
    self.preorder(x.left)
    self.preorder(x.right)

def inorder(self, x):
    if x is None:
        return
    self.inorder(x.left)
    print(x.item, end=' ')
    self.inorder(x.right)

def postorder(self, x):
    if x is None:
        return
    self.postorder(x.left)
    self.postorder(x.right)
    print(x.item, end=' ')
```

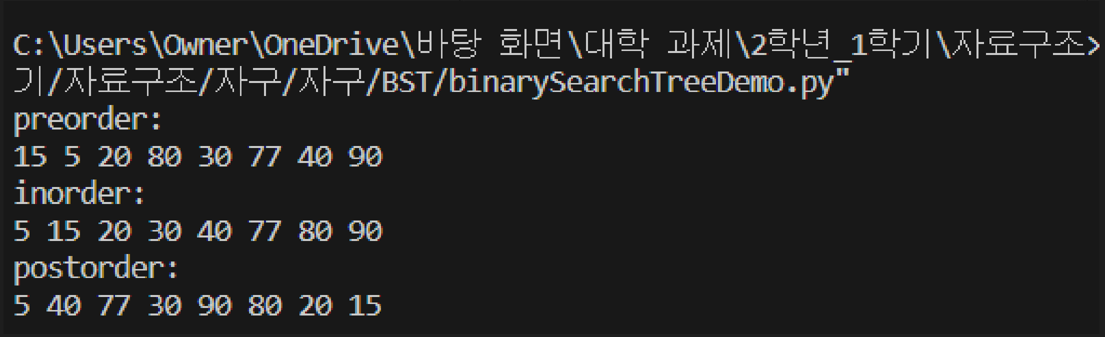
▲ binarySearchTree.py

binarySearchTreeDemo.py ▶



```python
from binarySearchTree import *

bst1 = BinarySearchTree()
bst1.insert(10)
bst1.insert(20)
bst1.insert(5)
bst1.insert(80)
bst1.insert(90)
bst1.insert(7550)
bst1.insert(30)
bst1.insert(77)
bst1.insert(15)
bst1.insert(40)
bst1.delete(7550)
bst1.delete(10)

print("preorder: ")
bst1.preorder(bst1.getRoot())
print("\ninorder: ")
bst1.inorder(bst1.getRoot())
print("\npostorder: ")
bst1.postorder(bst1.getRoot())
print("\n")
```

binarySearchTreeDemo.py 실행 결과 ▶



```
C:\Users\Owner\OneDrive\바탕 화면\대학 과제\2학년_1학기\자료구조>
기/자료구조/자구/자구/BST/binarySearchTreeDemo.py"
preorder:
15 5 20 80 30 77 40 90
inorder:
5 15 20 30 40 77 80 90
postorder:
5 40 77 30 90 80 20 15
```