CSC148 - Binary Search Tree Deletion

On this worksheet, you'll investigate one of the two major mutating methods on binary search trees: deleting an element from the tree. (The other major mutating method, insertion, will be covered in this week's lab.) As we may expect from a worksheet from last week, the core of a deletion algorithm on binary search trees is a helper that deletes the root of the tree:

```
def delete_root(self) -> None:
    """Remove the root of this BST. Precondition: this BST is not empty."""
```

Our goal for this worksheet is to develop an implementation of this method using a case-by-case approach.

Case 1: `self` is a leaf

Suppose `self` is a leaf (i.e., its left and right subtrees are empty). Discuss with your group what should happen to the tree in this case. Then in the space below, (1) fill in the `if` condition to check whether `self` is a leaf, and (2) fill in the body of the `if` to implement `delete_root` for this case. (Review the BST representation invariants from the prep readings!)

```
def delete_root(self) -> None:

    if                                                      # Case 1: this BST is a leaf
```

Case 2: exactly one of `self`'s subtrees are empty

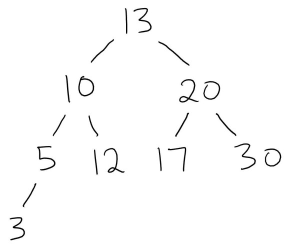
Draw two small binary search trees: one that has an empty left subtree and non-empty right subtree, and vice versa.

Now suppose we want to delete the root of each tree. The simplest approach is to use the "promote a subtree" technique from last week. Review this idea with your group, and then fill in the conditions and implementations of each `elif`.

```
    # Continued from Case 1...

    elif                                                    # Case 2a: non-empty left, empty right



    elif                                                    # Case 2b: empty left, non-empty right
```

Case 3: both subtrees are non-empty

Suppose we have the following BST, whose left and right subtrees are both non-empty.



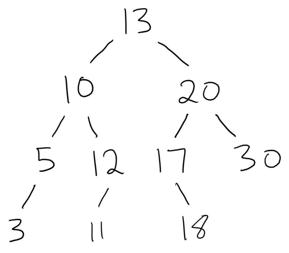
1. For this case, we'll use a different approach: *extract a value* from one of the subtrees, and use it to replace the current root value. We need to be careful to preserve the *binary search tree property*, since this is a representation invariant!

   First, look at the sample BST above, and suppose we want to replace the root 13. Circle the value(s) in the subtrees that we could use to replace the root, and make sure you understand why these values (and *only* these values) work.

2. Since there's two possible values, you have a choice about which one you want to pick. In the space below, write a helper method that you could call on `self.left` or `self.right` to extract the desired value, and then use that helper to complete the implementation of `delete_root`.

```
def delete_root(self) -> None:
    ...                                 # Parts 1 and 2 omitted
    else:                               # non-empty left, non-empty right




# Write your helper here!
```

3. Check your assumptions: did you assume that the value you were extracting is a leaf? Consider the following tree...


13
10
20
5
12
17
30
3
11
18