CSC148 - `delete_root` Helper for BST Deletion

As we saw earlier, a core step in the deletion algorithm on binary search trees is deletion at the root. Now we'll write a helper for that:

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

The docstring tells us what our job is.

Now we'll lead you through the cases to develop an implementation of this method.

Case 1: `self` is a leaf

a 3-column table, with row expanded.

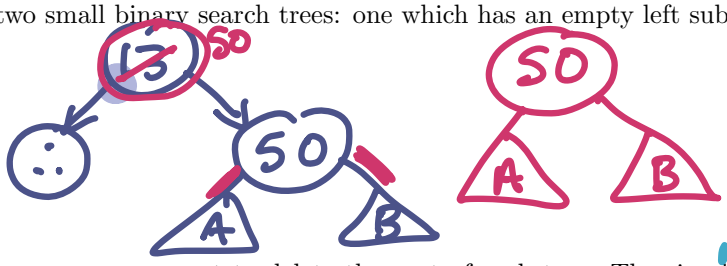
Suppose `self` is a leaf (i.e., its left and right subtrees are empty). Discuss with your group what should happen to the tree in this case. Then in the space below, (1) fill in the `if` condition to check whether `self` is a leaf, and (2) fill in the body of the `if` to implement `delete_root` for this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

```
def delete_root(self) -> None:
    if self._left.is_empty() and self._right.is_empty():          # Case 1: this BST is a leaf
        self._root = None
        self._left = None
        self._right = None
```

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

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

2a)



NB: We must re-use the BST object with the 13 in it (self) because our job is to mutate this BST, not make a new one. Remember that the caller is holding that ID somewhere!

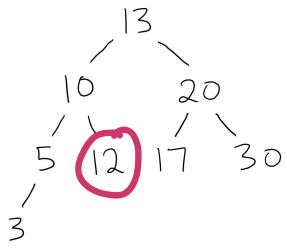
Now suppose we want to delete the root of each tree. The simplest approach is to use the "promote a subtree" technique from last week. Review this idea with your group, and then fill in the conditions and implementations of each `elif`.

```
    # Continued from Case 1...

    elif self._left.is_empty():                                   # Case 2a: non-empty right, empty left
        self._root, self._left, self._right = \
            self._right._root, self._right._left, self._right._right

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

Case 2b is an exercise (and is analogous to case 2a).

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, as we discovered earlier, we can *extract a value* from one of the subtrees and use it to replace the current root value. We need to do so carefully, to preserve the *binary search tree property*, since this is a representation invariant!

   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 are 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:
    ...                                 # Cases 1 and 2 omitted
    else:                               # Case 3: 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...