BST: for every node \( n \) with value \( v \),
- every value in \( n \)'s left subtree < \( v \), and
- \( v \) < every value in \( n \)'s right subtree

Three basic operations:
* search
* insert
* remove

SEARCH:
- start at root
- task for each node:
  1. compare node.item with value

INSERT(value):
- start at the root, and for every node:
  - if value < node.item: insert to the left
  - if value > node.item: insert to the right
  - if value == node.item: do nothing!
  - if node is None: create a new node to store value
* Project 1 test results: to be uploaded to MarkUs this week. Please wait until I have had a chance to do this, then let me know if you still have questions or concerns about the marking of Project 1.
Note: 3/4 on a category is considered "good"; 4/4 is considered "exceptional". Markers were told to be very picky before giving 4/4.

* Request for Note Takers: see course forum.

* Test 2! Next Wednesday (March 13th), 10:10am-11:00am, EX 200. Topics: linked lists and trees (and everything related, including recursion, yes...)

* E4...
Around 20 students (across both sections) being investigated for potential plagiarism... That's bad!

tree = BST()
tree.insert(20):
  _insert(tree.root, 20)
tree.insert(10):
  _insert(tree.root, 10)
trick: find largest value in the left subtree (the "in-order predecessor"), move it (value only) to replace the item to be removed, then remove it from below.