Code Snippets

```
157 # !!!!!!!!!! MY CODE STARTS !!!!!!!!!!
158     def findNode(self, root, val):
159         if (not root): # If there is no root
160             return None
161         elif val == root.data: # Found it!
162             return root
163         elif val < root.data: # Check the left
164             return self.findNode(root.left, val)
165         elif val > root.data: # Check the right
166             return self.findNode(root.right, val)
167 # !!!!!!!!!! MY CODE ENDS !!!!!!!!!!
```

```
184 def main():
185     myTree = AVLTree()
186     root = None
187     u_input = None
188 
189     print("Let's build an AVL Tree!")
190     while u_input == None or u_input >= 0:
191         try:
192             u_input = int(input("Provide integers 0 or greater.\nRepeats
193             data = u_input
194 
195             # Our breakout statement
196             if u_input < 0:
197                 break
198 
199             # Look for if the node exists
200             nodeFound = myTree.findNode(root, data)
201 
202             if nodeFound: # Delete if Found!
203                 root = myTree.delete_node(root, data)
204             else: # Insert if not!
205                 root = myTree.insert_node(root, data)
206 
207             # Print out our current tree
208             myTree.printHelper(root, "", True)
209         except: # Protect against bad input
210             print("^^^^^^^^^^^^^^^^^^^^^^")
211             print("Invalid Input Provided")
212             print("vvvvvvvvvvvvvvvvvvvvvv")
213 
214     print("Goodbye!")
215 
216 main()
```

NOTE: L192 is simply description of use for the user

Findings and Experience

Given the problem of inserting a node that does not exist in the tree already and deleting the ones that do when provided a user input meant that I needed to implement a binary search for the tree. Since we know that AVL Trees are balanced at all times, implementing a binary search means that we will always be hitting the **O(logn)** time complexity. Binary search in a tree only approaches O(n) when they are harshly unbalanced, but the nature of AVLs take care of that for us!

Initially, I was stuck for a moment just due to the fact that I could only get two nodes included and could not delete the second node. I quickly found out that there was an error in my implementation of the binary search where I forgot to return the left and right searches and instead was setting them equal to root.left and root.right, which did me no good. Once I realized that, the rest of the implementation was easy.

My main function consists of getting the following flow:

1. Instantiate the AVL Tree (empty)
2. Prompt the user for input
3. Convert to an int
    a. Protect against bad input using try/except
4. Check if user input is negative
    a. If so, exit the while loop
5. Search for the user input in the tree
    a. If a node with that data exists, perform the delete operation
    b. If a node with that data does not exist, perform the insert operation
6. Print out the current state of our tree

The program loops steps 2 through 6 until step 3a is true.