**Summarizing the content:**

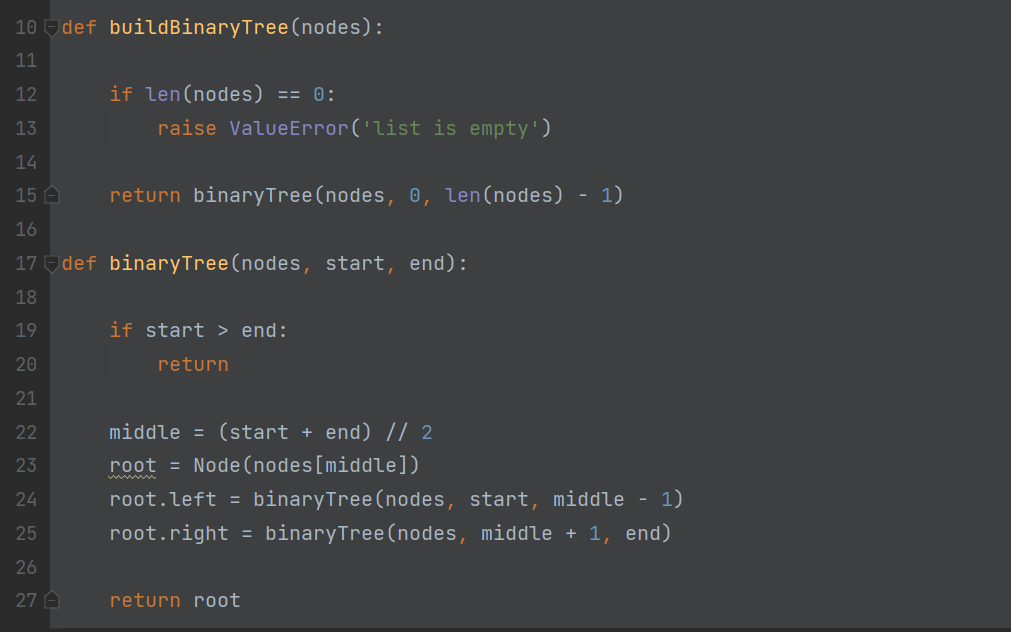
1. Summarise the main points in this module. You may include references to the learning objectives.
2. trees are form of graph data structure.
3. Binary Search Trees (BST).
4. BST are designed to achieve a goal of **O(log(n))** for IDS operations.
5. Self-Balancing Binary Search Tree
6. AVL Trees.
7. Rotation (Yoinking).
8. Right Rotation. Left Rotation. Left-Right Rotation. Right-Left Rotation.
9. Red-Black Trees.
10. **How is this useful?**
11. One reason to use trees might be because you want to store information that naturally forms a hierarchy. For example, the file system on a computer.
12. Trees (with some ordering e.g., BST) provide moderate access/search (quicker than Linked List and slower than arrays).
13. Trees provide moderate insertion/deletion (quicker than Arrays and slower than Unordered Linked Lists).
14. Like Linked Lists and unlike Arrays, Trees don’t have an upper limit on the number of nodes as nodes are linked using pointers.
15. **How do you plan to use this information?**
16. In compilers, Expression Trees are used which is an application of binary tree.
17. Huffman coding trees are used in data compression algorithms.
18. Priority Queue is another application of binary tree that is used for searching maximum or minimum in O(logN) time complexity.
19. **Provide summary of your reading list — external resources, websites, book chapters, code libraries, etc.**
20. <https://www.geeksforgeeks.org/binary-tree-set-1-introduction/>
21. <http://www.findstat.org/BinaryTrees>
22. <http://www.brpreiss.com/books/opus4/html/page355.html>
23. <https://piergiu.wordpress.com/2010/02/21/balanced-binary-search-tree-on-array/>
24. <https://lufemas.github.io/binary-tree-jr/>

**Reflecting on the content:**

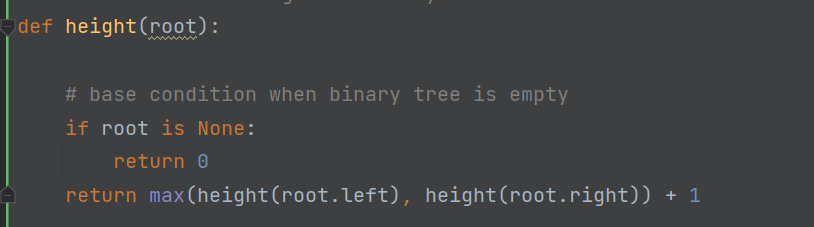
1. What is the most important thing you learnt in this module?
2. Insert, delete, query in binary tree.
3. Insert, delete, query in Binary Search Trees.
4. Insert, delete, query in Self-Balancing Binary Search Tree.
5. Insert, delete, query in Self-Balancing AVL Trees.
6. How does this relate to what you already know?

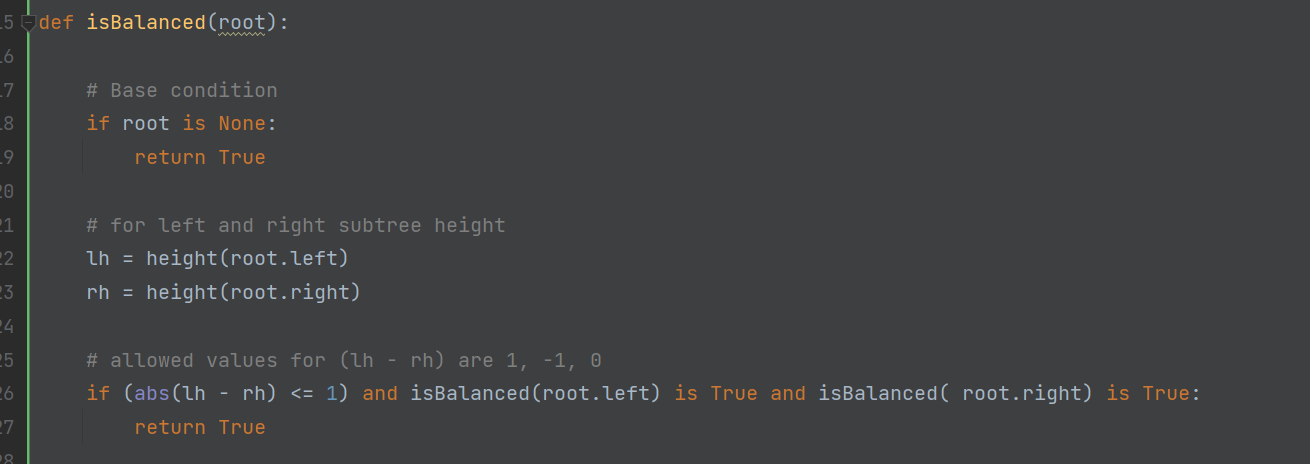
A tree is a popular data structure that is non-linear in nature. Unlike other data structures like array, stack, queue, and linked list which are linear in nature, a tree represents a hierarchical structure. The ordering information of a tree is not important. A tree contains nodes and 2 pointers. These two pointers are the left child and the right child of the parent node.

1. Reflect on the code that was given to you in the lab. You can take the screen shot of your python code and add image or just provide the code as text in your report. A good reflection includes:

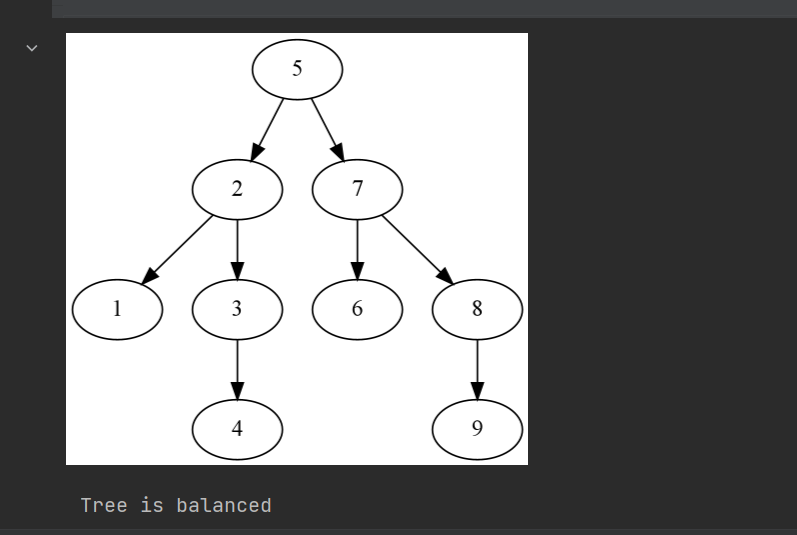
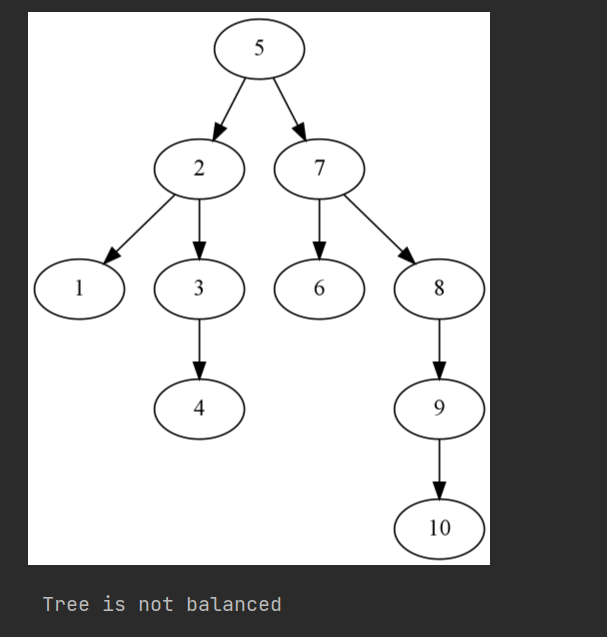


Use the recursive method to build a binary tree, the middle node is the root node, the left is the left child node, and the right is the right child node.

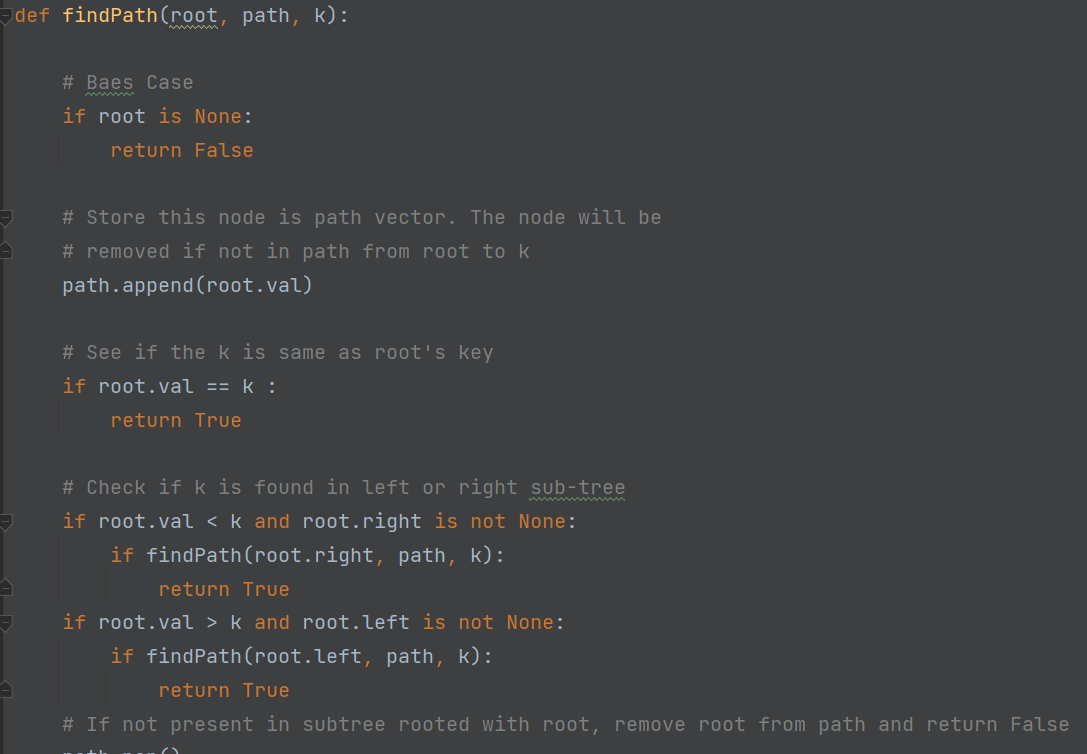




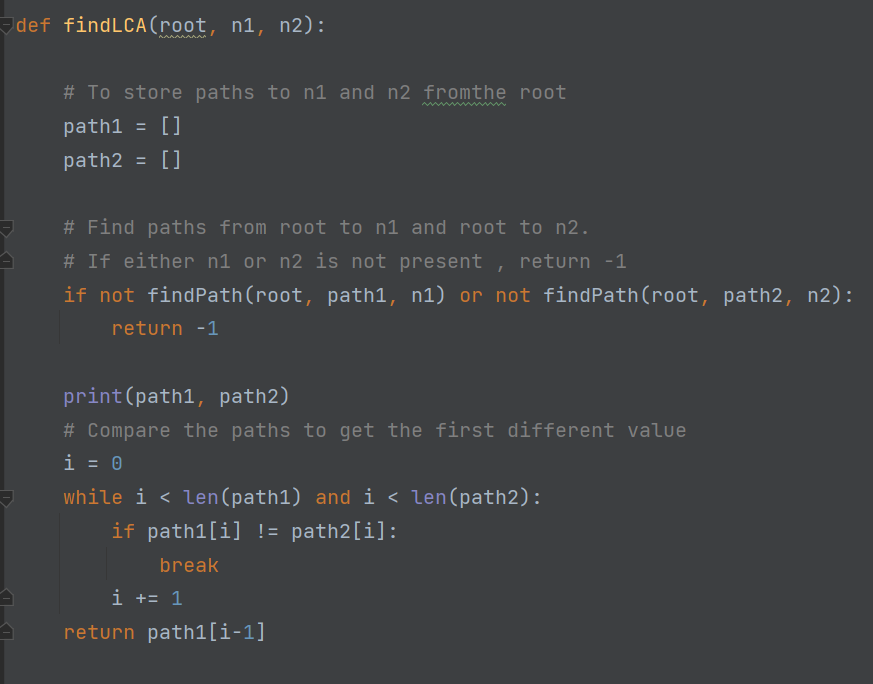
The above figure is used to recursively calculate the height difference between the left child node and the right child node of each node in the binary tree.

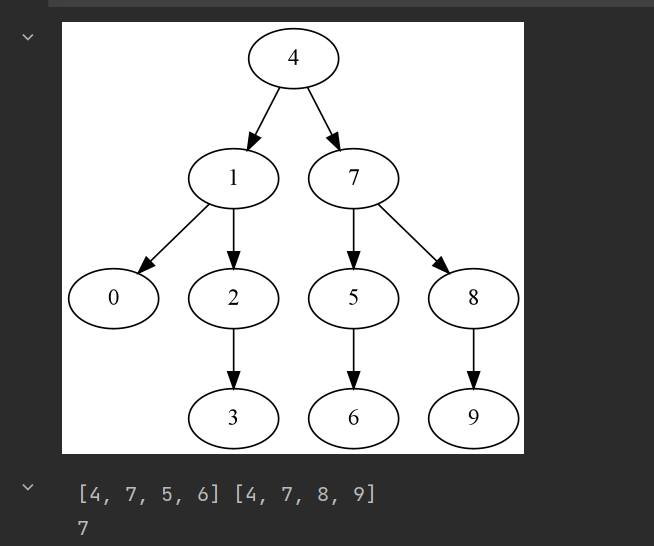
The picture above is the result of running the program.



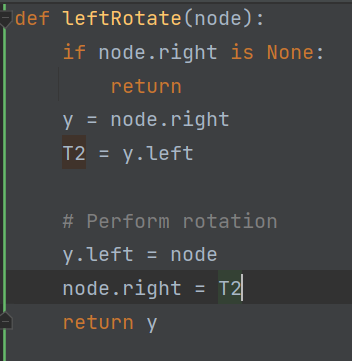
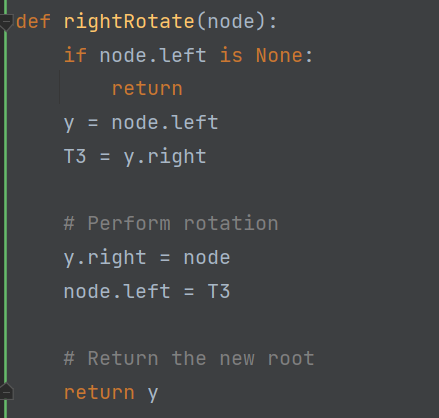
Finds the path from root node to given root of the tree.



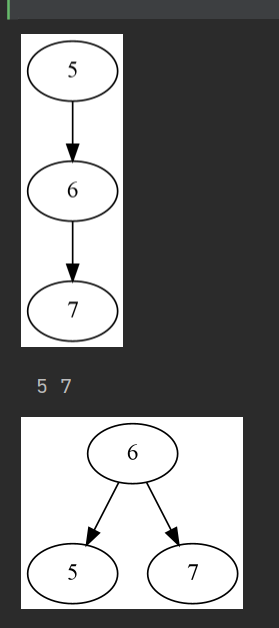
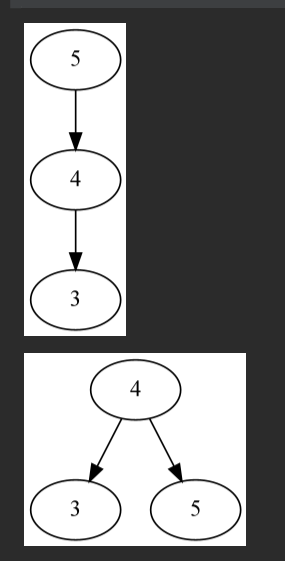
Returns LCA if node n1 , n2 are present in the given.



The result of running the program is shown in the figure.

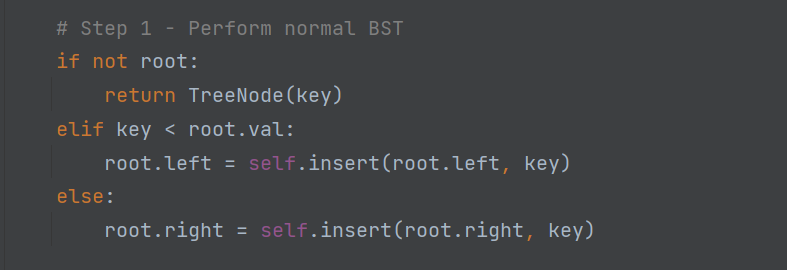
The above picture is the code for left-hand and right-hand rotation.

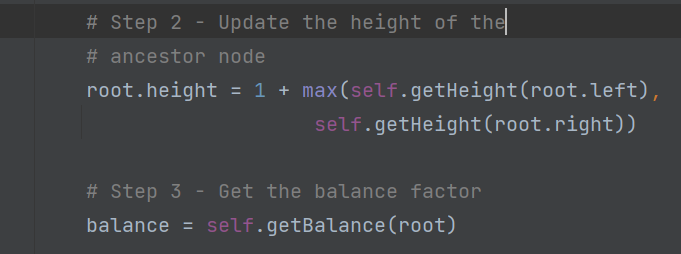
The results for left-hand and right-hand rotation are shown in the figure.

Activity 4:

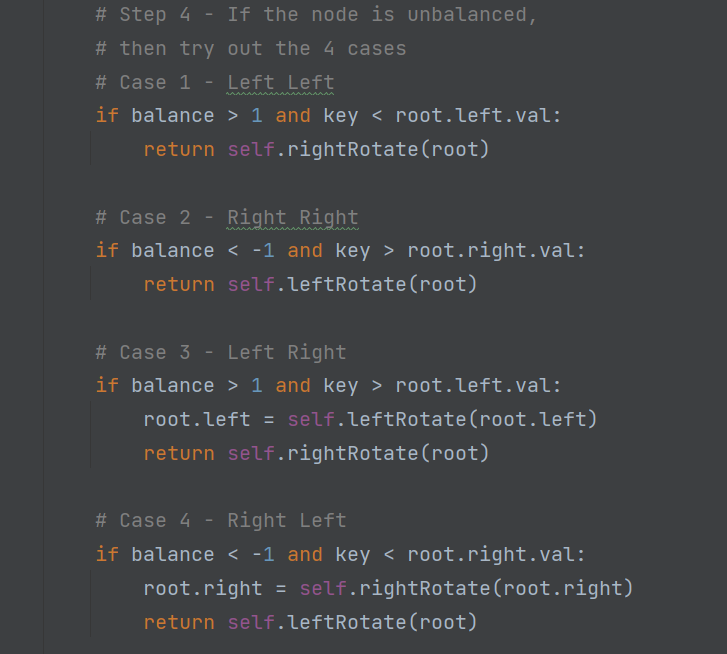
Run the AVL insertion code given in this week's lab. Demonstrate your understanding of the code by (doing some of the following activities):



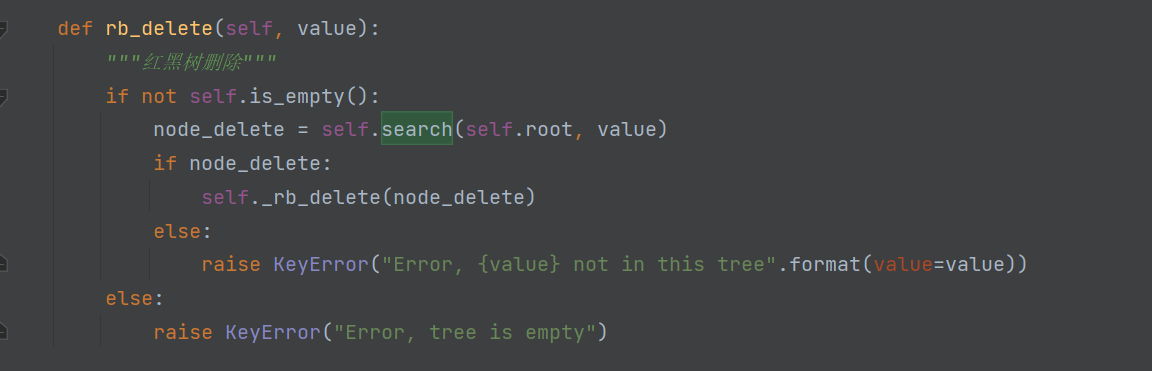
Find the node to insert.



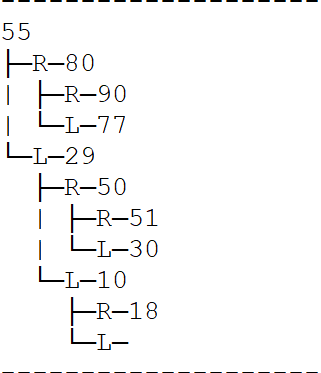
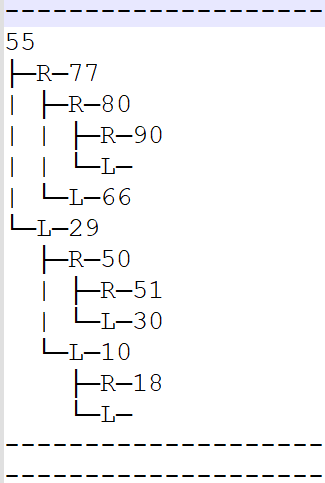
After inserting a node, determine whether the binary tree is balanced.



If the binary tree is unbalanced, rotate it in four cases.



red-black tree deletion code.



The picture on the left is before deletion, and the picture on the right is after deletion. The removed node is 66.