Lab4: Binary Search Tree (BST)

Implement the following operations for a Binary Search Tree class starting from the template provided. Use the Class TreeNode that is provided. You may implement helper methods that make your code easier to write, read, and understand. Test cases will be provided later but you should write test cases of your own as you develop the methods. You may use iterative or recursive functions in your implementation.

You will likely want to add setters and getters for the tree node fields other than the key field. Changing the key of a node is equivalent to removing and inserting it and that is the safer way to do the implementation rather than trying to move the node to reflect the change in the key.

To implement BST use two classes, because you must be able to create and work with a BST that is empty. The class BinarySearchTree has a reference to the class TreeNode that is the root of the BST. The class TreeNode can provide helper functions that make implementation in class BinarySearchTree easier.

```
class TreeNode:
  def __init__(self, key):
    self.left = None
    self.right = None
    self.key = key
    self.data = None

class BinarySearchTree:
# write the __init__() method here. The tree has at least a 'root' node.

  def find(self, key):  # returns True if key is in a node of the tree, else False

  def find_min(self):  # returns min value in the tree

  def find_max(self):  # returns max value in the tree

  def insert(self, newkey): # inserts a node with key into the correct position if not a duplicate.

  def delete(self, key):  # deletes the node containing key, assumes such a node exists

  def print_tree(self):  # print inorder the entire tree

  def is_empty(self):  # returns True if tree is empty, else False

  def inorder_print_tree(self)  # print inorder the subtree of self

  def print_levels(self)  # inorder traversal prints list of pairs, [key, level of the node] where root is level 0
```