val tree

Problem 1 AVL Tree Insertion（AVL 樹插入）

實作一個 **AVL Tree**，並完成以下功能：

- `insert(key)`：將鍵值插入 AVL 樹中
- 樹在每次插入後都必須維持平衡（LL、RR、LR、RL）

Problem 2 — AVL Tree Deletion（AVL 樹刪除）

延續上一題已建立好的 AVL Tree，實作：

- `delete(key)`：刪除一個節點
- 刪除後必須保持 AVL 平衡
- 若刪除的節點有兩個子樹 → 使用右子樹中最小值替代

範例code

```
class Node:
    def __init__(self, key):
        self.key = key
        self.left = None
        self.right = None
        self.height = 1


class AVLTree:

    def get_height(self, root):
        pass

    def get_balance(self, root):
        pass

    # Right rotation
```

```
def right_rotate(self, z):
        pass

# Left rotation
def left_rotate(self, z):
    pass

# Insert key
def insert(self, root, key):
    pass



# Delete key
def delete(self, root, key):
    pass

# Traversals
def preorder(self, root):
    if not root:
        return
    print(root.key, end=" ")
    self.preorder(root.left)
    self.preorder(root.right)

def inorder(self, root):
    if not root:
        return
    self.inorder(root.left)
    print(root.key, end=" ")
    self.inorder(root.right)

def postorder(self, root):
    if not root:
        return
    self.postorder(root.left)
```

```
        self.postorder(root.right)
        print(root.key, end=" ")
```

左右旋示意圖

右旋（Right Rotate）

原本：

```
   z
  /
 y
  \
   T3
```

右旋後：

```
   y
  / \
 ?   z
    /
   T3
```

（`?` 是 y 原本的左子樹，照樣保持不變）

左旋（Left Rotate）

原本：

```
  z
   \
    y
   /
  T2
```

左旋後：

```
 y
/\
z ?
\
 T2
```