🔥 **อธิบายโค้ด AVL Tree ใน Python**

โค้ดนี้เป็น **AVL Tree** ที่รองรับฟังก์ชันพื้นฐาน ได้แก่

1. **Insertion (avl_insert)** – เพิ่มข้อมูลและปรับสมดุล
2. **Deletion (avl_delete)** – ลบข้อมูลและปรับสมดุล
3. **Rotation (rotate_left, rotate_right)** – ใช้ปรับสมดุลเมื่อต้นไม้เสียสมดุล
4. **Printing (print_tree)** – แสดงโครงสร้างต้นไม้

🏗️ **1. โครงสร้างของโหนด (Node)**

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

🔷 **อธิบาย**:

- **Node** ใช้เก็บข้อมูล **key**
- มีตัวชี้ไปทางซ้าย (**left**) และขวา (**right**)

📏 **2. คำนวณความสูงของโหนด (avl_height)**

```
def avl_height(node):
    return -1 if node is None else 1 + max(avl_height(node.left), avl_height(node.right))
```

🔷 **อธิบาย**:

- คำนวณ **ความสูงของต้นไม้** โดยนับจากโหนดไปทางซ้ายหรือขวาที่สูงที่สุด
- ถ้าโหนดเป็น **None** ให้คืนค่า -1

🔄 **3. ฟังก์ชันหมุนต้นไม้ (Rotation)**

🔶 **Left Rotation (rotate_left)**

```
def rotate_left(r):
    u = r.right
    r.right = u.left
    u.left = r
    return u
```

🔷 **อธิบาย**:

- ใช้เมื่อโหนดขวาสูงกว่าซ้ายมากเกินไป (Right-Right Case)

🔷 **Right Rotation (rotate_right)**

```
def rotate_right(r):
    o = r.left
    r.left = o.right
    o.right = r
    return o
```

🔷 **อธิบาย**:

- ใช้เมื่อโหนดซ้ายสูงกว่าขวามากเกินไป (Left-Left Case)

🛠️ **4. ปรับสมดุลต้นไม้ (avl_adjust)**

```
def avl_adjust(r):
    balance = avl_height(r.right) - avl_height(r.left)

    if balance <= -2:  # ซ้ายหนักเกินไป
        bl = avl_height(r.left.right) - avl_height(r.left.left)
        if bl <= 0:
            return rotate_right(r)  # Left-Left Case
        else:
            r.left = rotate_left(r.left)
```

```
            return rotate_right(r)  # Left-Right Case

    if balance >= 2:  # ขวาหนักเกินไป
        br = avl_height(r.right.right) - avl_height(r.right.left)
        if br >= 0:
            return rotate_left(r)  # Right-Right Case
        else:
            r.right = rotate_right(r.right)
            return rotate_left(r)  # Right-Left Case

    return r
```

🔷 **อธิบาย**:

- ตรวจสอบ **Balance Factor**
- ถ้า **ซ้ายหนัก** → ใช้ **Right Rotation**
- ถ้า **ขวาหนัก** → ใช้ **Left Rotation**
- ถ้าเป็น **Left-Right หรือ Right-Left Case** → ใช้ **Double Rotation**

📥 **5. การเพิ่มข้อมูล (Insertion)**

```
def avl_insert(r, key):
    if r is None:
        return Node(key)

    if key < r.key:
        r.left = avl_insert(r.left, key)
    elif key > r.key:
        r.right = avl_insert(r.right, key)
```

```
    return avl_adjust(r)
```

🔷 **อธิบาย**:

- แทรกค่าลงใน **BST ตามกฎปกติ**
- หลังจากเพิ่มแล้ว **เรียก avl_adjust()** เพื่อปรับสมดุล

🗑️ **6. การลบข้อมูล (Deletion)**

```
def avl_minimum(r):
    return float('inf') if r is None else r.key if r.left is None else avl_minimum(r.left)
```

🔷 **อธิบาย**:

- หา **ค่าต่ำสุด (Minimum)** ในต้นไม้

```
def avl_delete(r, key):
    if r is None:
        return None

    if key < r.key:
        r.left = avl_delete(r.left, key)
    elif key > r.key:
        r.right = avl_delete(r.right, key)
    else:
        if r.left is None:
            return r.right
        elif r.right is None:
            return r.left

        min_key = avl_minimum(r.right)
        r.key = min_key
        r.right = avl_delete(r.right, min_key)
```

```
    return avl_adjust(r)
```

🔷 **อธิบาย**:

- ค้นหาโหนดที่ต้องการลบ
- ถ้าไม่มีลูก ⟶ **ลบได้เลย**
- ถ้ามี **ลูกเดียว → แทนที่ด้วยลูก**
- ถ้ามี **ลูกสองฝั่ง → ใช้ค่าที่น้อยสุดของฝั่งขวาแทน** แล้วลบค่านั้น

🖨️ **7. แสดงผลต้นไม้ (print_tree)**

```
def print_tree(r):
    if r is None:
        print("( )", end="")
        return
    if r.left is None and r.right is None:
        print(f"({r.key})", end="")
        return
    print("(", end="")
    print_tree(r.left)
    print(f"({r.key})", end="")
    print_tree(r.right)
    print(")", end="")
```

🔷 **อธิบาย**:

- แสดงโครงสร้าง **AVL Tree** โดยใช้วงเล็บเพื่อแสดงโครงสร้างต้นไม้

🏳️ **8. ตัวอย่างการใช้งาน**

```
if __name__ == "__main__":
```

```
root = None
for i in range(7):
    root = avl_insert(root, i)
    print_tree(root)
    print()

root = avl_delete(root, 0)
print_tree(root)
print()
root = avl_delete(root, 1)
print_tree(root)
print()
root = avl_delete(root, 2)
print_tree(root)
print()
```

🔷 **อธิบาย**:

- **เพิ่มค่า** ลงใน **AVL Tree** ทีละค่า
- **ลบค่าบางค่า** แล้วแสดงผล

🎯 **สรุป**

- **AVL Tree** คือ **Binary Search Tree** ที่ **รักษาสมดุล** เสมอ
- ใช้ **Rotation (LL, RR, LR, RL)** เพื่อปรับสมดุล
- มี **Insertion, Deletion และ Search ที่ทำงานใน O(log n)**
- เหมาะสำหรับการ **เก็บข้อมูลที่ต้องการเข้าถึงรวดเร็ว**

✅ **โค้ดนี้รองรับการเพิ่ม, ลบ และพิมพ์โครงสร้าง AVL Tree ได้อย่างสมบูรณ์!** 🚀