**TASK 8 Explain Code**

**Explanation in PGL Points**

**P – Purpose (What the code does)**

* This code implements the **Min-Max Algorithm**.
* The goal is to find the **best possible move** for a player in a two-player game (like Chess or Tic-Tac-Toe).
* One player tries to **maximize** the score (MAX player), and the other tries to **minimize** it (MIN player).

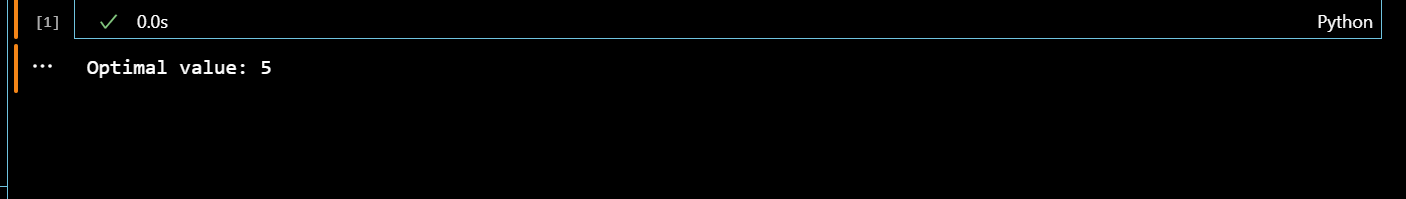
**G – General Working (How it works)**

1. The **minimax()** function is called **recursively** for every node (move).
2. **Base case:** When we reach the leaf node (depth = h), it **returns the score** of that node.
3. If it’s the **MAX player’s turn**, the algorithm chooses the **maximum value** of the next two nodes.
4. If it’s the **MIN player’s turn**, it chooses the **minimum value** of the next two nodes.
5. This process continues until we reach back to the root node, giving the **best possible score** for the MAX player.

**L – Logic / Line-by-Line Explanation**

| **Code Line** | **Explanation** |
| --- | --- |
| if depth == h: | Checks if we have reached the bottom (leaf level) of the tree. |
| return scores[nodeIndex] | Returns the leaf node’s score value. |
| if isMax: | Checks if it’s the MAX player’s turn. |
| return max(...) | Chooses the maximum of the two possible moves (best for MAX). |
| else: | Otherwise, it’s the MIN player’s turn. |
| return min(...) | Chooses the minimum of the two possible moves (best for MIN). |
| nodeIndex \* 2 / nodeIndex \* 2 + 1 | Finds left and right child node indexes. |
| scores = [...] | This list holds all the leaf node scores. |
| h = 3 | Height of the tree (number of levels). |
| print(...) | Displays the final best value for the MAX player. |

**Output:**

****