Empowering Tomorrow, Today with Main Flow

PYTHON DEVELOPER

TASK - 3

17. **Table of a Number**

Objective: Print the multiplication table for a given number nnn.

Input: An integer nnn.

Output: Multiplication table from 111 to 101010.

Hint: Use a loop to iterate through values 1 to 10 and multiply by nnn.

---

18. **Swap Two Numbers**

Objective: Swap two numbers without using a third variable.

Input: Two integers aaa and bbb.

Output: Swapped values of aaa and bbb.

Hint: Use arithmetic operations like addition and subtraction or XOR (`a, b = b, a`).

---

19. **Check Substring**

Objective: Determine if one string is a substring of another.

Input: Two strings s1s1s1 (main string) and s2s2s2 (substring).

Output: `True` if s2s2s2 is a substring of s1s1s1, otherwise `False`.

Hint: Use Python's `in` operator or string slicing to search for substrings.

---

20. **Decimal to Binary**

Objective: Convert a decimal number to its binary representation.

Input: An integer nnn.

Output: A string representing the binary equivalent.

Hint: Use the `bin()` function or repeatedly divide nnn by 2, storing remainders.

---

21. **Matrix Addition**

Objective: Add two matrices of the same dimensions.

Input: Two 2D lists (matrices) of integers.

Output: A 2D list containing the sum of corresponding elements.

Hint: Use nested loops to iterate through rows and columns, adding corresponding elements.

22. **Matrix Multiplication**
    Objective: Multiply two matrices AAA and BBB.
    Input: Two 2D lists where the number of columns in AAA equals the number of rows in BBB.
    Output: A 2D list representing the product matrix.
    Hint: Multiply elements row-by-column and sum for each position in the result matrix.

23. **Find Second Largest**
    Objective: Find the second largest number in a list.
    Input: A list of integers.
    Output: The second largest integer.
    Hint: Use sorting or iterate to find the largest, then the second largest.

24. **Check Anagram**
    Objective: Check if two strings are anagrams (contain the same characters in any order).
    Input: Two strings.
    Output: `True` if anagrams, otherwise `False`.
    Hint: Use `sorted()` on both strings or count character occurrences using a dictionary.

3. AI-Based Tic-Tac-Toe

- **Description**: Create a Tic-Tac-Toe game where the computer plays against the user and uses a minimax algorithm to make decisions.
- **Challenges**:
  - Implement AI logic with decision trees.
  - Handle edge cases like a full board or winning moves.
  - Provide a user-friendly interface.
- **Skills**: Game theory, recursion, and strategic thinking.

3. AI-Based Tic-Tac-Toe

- **Restriction**: **Only use the minimax algorithm** for AI decision-making.
- **Reason**: The **minimax algorithm** is a classic AI strategy used in games to determine optimal moves. This restriction forces students to **implement and understand the core logic of decision-making algorithms**, ensuring the AI plays optimally and is not random or rudimentary. This will deepen their understanding of decision trees, recursion, and game theory.
- **Learning Outcome**: Students will learn how to create **intelligent agents** in games, gaining insight into **search algorithms**, **recursion**, and **game strategy optimization**.

Deadline Compliance

- **Restriction**: **Submit the project within 7 days** from the start date.
- **Reason**: Meeting deadlines is crucial in the real-world software development environment. This restriction helps students practice **time management** and **task prioritization**. In professional settings, tight deadlines are often the norm, and learning to meet them without compromising quality is an essential skill.
- **Learning Outcome**: Students will learn to manage their time effectively, complete projects under pressure, and **deliver results on time**, which are all important skills in the workplace.