TASK – 3

17. Table of a Number: Print the multiplication table for a given number *n*.

Input: An integer *n*.

Output: Multiplication table from 1 to 10.

Code:

```
n = int(input("Enter a number: "))
for i in range(1, 11):
  print(f"{n} x {i} = {n * i}")
```

Explanation:

- The loop runs from 1 to 10.
- Each iteration multiplies *n* by the loop index *i*.
- The formatted string prints the result.

18. Swap Two Numbers: Swap two numbers without using a third variable.

Input: Two integers *a* and *b*.

Output: Swapped values of *a* and *b*.

Code:

```
a, b = map(int, input("Enter two numbers: ").split())
temp = a
a = b
b = temp
print(f"After swapping: a = {a}, b = {b}")
```

Explanation:

- A temporary variable temp stores the value of *a*.
- *a* is assigned the value of *b*.
- *b* is assigned the value of temp.

19. Check Substring: Determine if one string is a substring of another.

Input: Two strings *s1* (main string) and *s2* (substring).

Output: True if *s2* is a substring of *s1*, otherwise False.

Code:

```
def is_substring(s1, s2):
  for i in range(len(s1) - len(s2) + 1):
    if s1[i:i+len(s2)] == s2:
      return True
  return False


s1 = input("Enter the main string: ")
s2 = input("Enter the substring: ")
print(is_substring(s1, s2))
```

Explanation:

Manually iterates through *s1* checking if *s2* matches any substring

20. Decimal to Binary: Convert a decimal number to binary.

Input: An integer *n*.

Output: A string representing the binary equivalent.

Code:

```
def decimal_to_binary(n):
```

```
    binary = ""
    while n > 0:
        binary = str(n % 2) + binary
        n //= 2
    return binary if binary else "0"

n = int(input("Enter a decimal number: "))
print(decimal_to_binary(n))
```

Explanation:

Uses manual division by 2 to construct the binary string.

21. Matrix Addition: Add two matrices of the same dimensions.

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

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

Code:

```
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
result = []
for i in range(len(A)):
    row = []
    for j in range(len(A[0])):
        row.append(A[i][j] + B[i][j])
    result.append(row)
print(result)
```

Explanation:

Uses nested loops to add corresponding elements manually.

22. Matrix Multiplication: Multiply two matrices.

Input: Two 2D lists where the number of columns in *A* equals the number of rows in *B*.

Output: A 2D list representing the product matrix.

Code:

```
A = [[1, 2], [3, 4]]
B = [[5, 6], [7, 8]]
result = [[0, 0], [0, 0]]
for i in range(len(A)):
    for j in range(len(B[0])):
        for k in range(len(B)):
            result[i][j] += A[i][k] * B[k][j]
print(result)
```

Explanation:

Manually performs matrix multiplication using loops.

23. Find Second Largest: Find the second largest number in a list.

Input: A list of integers.

Output: The second largest integer.

Code:

```
def second_largest(numbers):
    largest = second = float('-inf')
    for num in numbers:
        if num > largest:
            second, largest = largest, num
```

```
    elif num > second and num != largest:
        second = num
  return second

numbers = list(map(int, input("Enter numbers: ").split()))
print(second_largest(numbers))
```

Explanation:

Iterates through list tracking largest and second largest values manually.

24. Check Anagram: Check if two strings are anagrams.

Input: Two strings.

Output: True if anagrams, otherwise False.

Code:

```
def is_anagram(s1, s2):
  if len(s1) != len(s2):
    return False
  count1, count2 = {}, {}
  for char in s1:
    count1[char] = count1.get(char, 0) + 1
  for char in s2:
    count2[char] = count2.get(char, 0) + 1
  return count1 == count2

s1 = input("Enter first string: ")
s2 = input("Enter second string: ")
print(is_anagram(s1, s2))
```

Explanation:

Uses dictionaries to count character occurrences manually.

25. AI-Based Tic-Tac-Toe: Create a Tic-Tac-Toe game with AI using the minimax algorithm.

Challenges:

- Implement AI logic with decision trees.
- Handle edge cases like a full board or winning moves.
- Provide a user-friendly interface.

Code:

```
def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 5)


def check_winner(board):
    for row in board:
        if row[0] == row[1] == row[2] != ' ':
            return row[0]
    for col in range(3):
        if board[0][col] == board[1][col] == board[2][col] != ' ':
            return board[0][col]
    if board[0][0] == board[1][1] == board[2][2] != ' ':
        return board[0][0]
    if board[0][2] == board[1][1] == board[2][0] != ' ':
        return board[0][2]
```

```
    return None

def minimax(board, depth, is_maximizing):
    winner = check_winner(board)
    if winner == 'X':
        return -10 + depth
    if winner == 'O':
        return 10 - depth
    if all(board[i][j] != ' ' for i in range(3) for j in range(3)):
        return 0
    if is_maximizing:
        best_score = float('-inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'O'
                    score = minimax(board, depth + 1, False)
                    board[i][j] = ' '
                    best_score = max(score, best_score)
        return best_score
    else:
        best_score = float('inf')
        for i in range(3):
            for j in range(3):
                if board[i][j] == ' ':
                    board[i][j] = 'X'
                    score = minimax(board, depth + 1, True)
                    board[i][j] = ' '
```

```
                best_score = min(score, best_score)
        return best_score

def best_move(board):
    best_score = float('-inf')
    move = None
    for i in range(3):
        for j in range(3):
            if board[i][j] == ' ':
                board[i][j] = 'O'
                score = minimax(board, 0, False)
                board[i][j] = ' '
                if score > best_score:
                    best_score = score
                    move = (i, j)
    return move

def play_game():
    board = [[' ' for _ in range(3)] for _ in range(3)]
    while True:
        print_board(board)
        row, col = map(int, input("Enter row and column (0-2): ").split())
        if board[row][col] != ' ':
            print("Invalid move! Try again.")
            continue
        board[row][col] = 'X'
        if check_winner(board):
            print_board(board)
```

```
            print("You win!")
            break
        if all(board[i][j] != ' ' for i in range(3) for j in range(3)):
            print_board(board)
            print("It's a tie!")
            break
        move = best_move(board)
        if move:
            board[move[0]][move[1]] = 'O'
        if check_winner(board):
            print_board(board)
            print("AI wins!")
            break

play_game()
```

Explanation:

- Implements a playable Tic-Tac-Toe game with user input and AI.
- The AI uses the minimax algorithm to make optimal moves.
- The game prints the board and manages turns automatically.