Jainam Jagani
Intern ID: 17696

TASK-3

1. Table of a Number

Description of Problem: This problem prints the multiplication table for a given number from 1 to 10.

Approach: • Understand the core requirement of the problem • Use a loop to multiply the number by values 1 to 10 • Print each result in a formatted way

Key Challenges: • None significant (basic loop logic)

Solutions: • Simple for loop for range 1 to 10

Logic Used:

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

What I Learned Through This Problem: • Loop control and formatted output

Edge Cases Handled: • Works for negative and zero as well

Output Example:

```
Enter a number: 5
Multiplication Table of 5:
5 x 1 = 5
...
5 x 10 = 50
```

---

2. Swap Two Numbers

Description of Problem: Swap two integers without using a third variable.

Approach: • Understand arithmetic or tuple unpacking method • Perform in-place swap

Key Challenges: • Avoid data loss during swap

Solutions: • Use addition/subtraction or tuple unpacking

Logic Used:

```
a = int(input("Enter first number: "))
b = int(input("Enter second number: "))
```

```
a = a + b
b = a - b
a = a - b
print("After swapping:")
print("a =", a)
print("b =", b)
```

What I Learned Through This Problem: • Variable manipulation without extra space

Edge Cases Handled: • Works for negative values

Output Example:

```
Enter first number: 5
Enter second number: 10
After swapping:
a = 10
b = 5
```

---

3. Check Substring

Description of Problem: Determine if one string is a substring of another.

Approach: • Use Python in operator • Alternatively, loop and check manually

Key Challenges: • Handle case sensitivity and spaces if needed

Solutions: • in operator for simplicity

Logic Used:

```
s1 = input("Enter main string: ")
s2 = input("Enter substring: ")
print(s2 in s1)
```

What I Learned Through This Problem: • String membership checking

Edge Cases Handled: • Empty substring returns True

Output Example:

```
Enter main string: Hello World
Enter substring: World
True
```

4. Decimal to Binary

Description of Problem: Convert a decimal number to binary.

Approach: • Use `bin()` or manual division by 2

Key Challenges: • Correct formatting of binary string

Solutions: • Use `bin().replace('0b', '')` or manual loop

Logic Used:

```
n = int(input("Enter a decimal number: "))
print("Binary:", bin(n).replace("0b", ""))
```

What I Learned Through This Problem: • Binary representation logic

Edge Cases Handled: • Works for 0 and negative numbers

Output Example:

```
Enter a decimal number: 10
Binary: 1010
```

5. Matrix Addition

Description of Problem: Add two matrices of the same dimensions.

Approach: • Use nested loops to add corresponding elements

Key Challenges: • Validate same dimensions for both matrices

Solutions: • Iterate rows and columns to sum elements

Logic Used:

```
rows = int(input("Enter rows: "))
cols = int(input("Enter cols: "))
A = [[int(input()) for _ in range(cols)] for _ in range(rows)]
B = [[int(input()) for _ in range(cols)] for _ in range(rows)]
C = [[A[i][j] + B[i][j] for j in range(cols)] for i in range(rows)]
for row in C:
    print(row)
```

What I Learned Through This Problem: • 2D list manipulation

Edge Cases Handled: • Non-square matrices work if sizes match

Output Example:

```
Resultant Matrix:
[6, 8]
[10, 12]
```

6. Matrix Multiplication

Description of Problem: Multiply two matrices where cols of A = rows of B.

Approach: • Nested loops with triple iteration

Key Challenges: • Ensure valid dimensions

Solutions: • Classic triple loop multiplication

Logic Used:

```
rows_A = int(input("Rows A: "))
cols_A = int(input("Cols A: "))
rows_B = int(input("Rows B: "))
cols_B = int(input("Cols B: "))
if cols_A != rows_B:
    print("Not possible")
else:
    A = [[int(input()) for _ in range(cols_A)] for _ in range(rows_A)]
    B = [[int(input()) for _ in range(cols_B)] for _ in range(rows_B)]
    C = [[0]*cols_B for _ in range(rows_A)]
    for i in range(rows_A):
        for j in range(cols_B):
            for k in range(cols_A):
                C[i][j] += A[i][k] * B[k][j]
    for row in C:
        print(row)
```

What I Learned Through This Problem: • Matrix multiplication algorithm

Edge Cases Handled: • Dimension mismatch handled

7. Find Second Largest

Description of Problem: Find the second largest number in a list.

Approach: • Remove duplicates, sort, pick second largest

Key Challenges: • Handle duplicates and list with fewer than 2 elements

Solutions: • Use set to remove duplicates then sort

Logic Used:

```
nums = [int(x) for x in input("Enter numbers: ").split()]
unique_nums = sorted(set(nums), reverse=True)
if len(unique_nums) < 2:
    print("No second largest")
else:
    print("Second Largest:", unique_nums[1])
```

What I Learned Through This Problem: • Sorting and unique extraction

Edge Cases Handled: • All elements equal

---

8. Check Anagram

Description of Problem: Check if two strings are anagrams.

Approach: • Sort both strings and compare

Key Challenges: • Ignore case and spaces

Solutions: • Normalize strings, sort, compare

Logic Used:

```
s1 = input("Enter first string: ").replace(" ", "").lower()
s2 = input("Enter second string: ").replace(" ", "").lower()
print(sorted(s1) == sorted(s2))
```

What I Learned Through This Problem: • String normalization and sorting

Edge Cases Handled: • Different casing, spaces

---

9. AI-Based Tic Tac Toe

Description of Problem: Create a Tic-Tac-Toe game with AI using minimax algorithm.

Approach: • Represent board as list • Implement minimax for optimal move selection

Key Challenges: • Recursive game tree evaluation • Handling tie/win states

Solutions: • Minimax algorithm for AI decision

Logic Used:

```
import math
board = [" " for _ in range(9)]

def print_board():
    for i in range(3):
        print("|" + "|".join(board[i*3:(i+1)*3]) + "|")
```

```
def check_winner(p):
    wins = [(0,1,2),(3,4,5),(6,7,8),(0,3,6),(1,4,7),(2,5,8),(0,4,8),(2,4,6)]
    return any(board[a]==board[b]==board[c]==p for a,b,c in wins)

def is_full():
    return " " not in board

def minimax(is_max):
    if check_winner("O"): return 1
    if check_winner("X"): return -1
    if is_full(): return 0
    if is_max:
        best = -math.inf
        for i in range(9):
            if board[i]==" ":
                board[i]="O"
                best=max(best,minimax(False))
                board[i]=" "
        return best
    else:
        best = math.inf
        for i in range(9):
            if board[i]==" ":
                board[i]="X"
                best=min(best,minimax(True))
                board[i]=" "
        return best

def ai_move():
    best=-math.inf;move=0
    for i in range(9):
        if board[i]==" ":
            board[i]="O"
            score=minimax(False)
            board[i]=" "
            if score>best:
                best=score;move=i
    board[move]="O"

def play_game():
    while True:
        print_board()
        try:
            m=int(input("Move (1-9):"))-1
            if board[m]!=" ":continue
        except:continue
        board[m]="X"
        if check_winner("X"):print_board();print("You win!");break
        if is_full():print_board();print("Tie!");break
        ai_move()
        if check_winner("O"):print_board();print("AI wins!");break
        if is_full():print_board();print("Tie!");break
play_game()
```

What I Learned Through This Problem: • Game tree search • Optimal AI decision making

Edge Cases Handled: • Full board tie, invalid inputs

Output Example

```
Welcome to AI Tic-Tac-Toe!

| | | |

| | | |

| | | |


Move (1-9): 1

|X| | |

| | | |

| | | |


AI has made its move:

|X| | |

| |O| |

| | | |


...

AI wins!
```