Table of a Number

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

Input: An integer nnn.

Program :

```
def multiplication_table(n):

  for i in range(1, 11):
    print(f"{n} x {i} = {n * i}")

n = int(input("Enter an integer: "))
multiplication_table(n)
```

output :

Enter an integer: 5

5 x 1 = 5

5 x 2 = 10

5 x 3 = 15

5 x 4 = 20

5 x 5 = 25

5 x 6 = 30

5 x 7 = 35

5 x 8 = 40

5 x 9 = 45

5 x 10 = 50

Swap Two Numbers

Objective: Swap two numbers without using a third variable.

Input: Two integers aaa and bbb.

Program :

```
def swap_numbers(a, b):

  a, b = b, a
  return a, b

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

swapped_a, swapped_b = swap_numbers(a, b)

print(f"After swapping: a = {swapped_a}, b = {swapped_b}")
```

output :

Enter the first number: 6

Enter the second number: 7

After swapping: a = 7, b = 6

Check Substring

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

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

Program :

```
def is_substring(s1, s2):

  return s2 in s1

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

```
if is_substring(s1, s2):
  print(True)
else:
  print(False)
```

Output :

```
Hello World
 World
True
```

Decimal to Binary

Objective: Convert a decimal number to its binary representation.

Input: An integer nnn.

Program :

```
def decimal_to_binary(n):
  if n == 0:
    return "0"

  binary = ""
  while n > 0:
    remainder = n % 2
    binary = str(remainder) + binary
    n = n // 2
  return binary
n = int(input("Enter a decimal number: "))
binary_representation = decimal_to_binary(n)
print(f"Binary representation of {n} is: {binary_representation}")

#Example using bin()
def decimal_to_binary_bin(n):
  return bin(n)[2:] #slice the "0b" off of the beginning of the string.
```

```
print(f"Binary representation of {n} using bin() is: {decimal_to_binary_bin(n)}")
```

Output :

10

Binary representation of 10 is: 1010

Binary representation of 10 using bin() is: 1010

Matrix Addition

Objective: Add two matrices of the same dimensions.

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

Program :

```
def matrix_addition(matrix1, matrix2):  if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
    return None  # Matrices must have the same dimensions

  result = []
  for i in range(len(matrix1)):  # Iterate through rows
    row = []
    for j in range(len(matrix1[0])):  # Iterate through columns
      row.append(matrix1[i][j] + matrix2[i][j])
    result.append(row)
  return result
matrix_a = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
matrix_b = [[9, 8, 7], [6, 5, 4], [3, 2, 1]]

sum_matrix = matrix_addition(matrix_a, matrix_b)

if sum_matrix:
  print("Matrix A:")
```

```
  for row in matrix_a:
    print(row)

  print("\nMatrix B:")
  for row in matrix_b:
    print(row)

  print("\nSum Matrix:")
  for row in sum_matrix:
    print(row)
else:
  print("Matrices have different dimensions and cannot be added.")
```

Output :

```
Matrix A: [1, 2, 3]
 [4, 5, 6]
 [7, 8, 9]
Matrix B: [9, 8, 7]
 [6, 5, 4]
 [3, 2, 1]
Sum Matrix: [10, 10, 10]
 [10, 10, 10]
 [10, 10, 10]
```

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.

Program :

```
def matrix_multiplication(matrix1, matrix2):
```

```
 rows1 = len(matrix1)
 cols1 = len(matrix1[0])
 rows2 = len(matrix2)
 cols2 = len(matrix2[0])

 if cols1 != rows2:
  return None  # Multiplication is not possible

 result = [[0 for _ in range(cols2)] for _ in range(rows1)]

 for i in range(rows1):
  for j in range(cols2):
   for k in range(cols1):  # or rows2
    result[i][j] += matrix1[i][k] * matrix2[k][j]

 return result

matrix_a = [[1, 2, 3], [4, 5, 6]]
matrix_b = [[7, 8], [9, 10], [11, 12]]

product_matrix = matrix_multiplication(matrix_a, matrix_b)

if product_matrix:
 print("Matrix A:")
 for row in matrix_a:
  print(row)

 print("\nMatrix B:")
 for row in matrix_b:
  print(row)
```

```
  print("\nProduct Matrix:")
  for row in product_matrix:
    print(row)
else:
  print("Matrices cannot be multiplied due to incompatible dimensions.")
```

Output :

Matrix A: [1, 2, 3] [4, 5, 6] Matrix B: [7, 8] [9, 10] [11, 12] Product Matrix: [58, 64] [139, 154]

Find Second Largest

Objective: Find the second largest number in a list.

Input: A list of integers.

Program :

```
def find_second_largest(numbers):
  if len(numbers) < 2:
    return None

  largest = numbers[0]
  second_largest = numbers[1]

  if second_largest > largest:
    largest, second_largest = second_largest, largest

  for num in numbers[2:]:
    if num > largest:
      second_largest = largest
      largest = num
    elif num > second_largest and num != largest:
      second_largest = num
```

```
 if largest == second_largest:
   return None #all numbers are the same.
 return second_largest

numbers1 = [12, 45, 2, 41, 31, 10, 6, 4]
second_largest1 = find_second_largest(numbers1)
print(f"Second largest number in {numbers1} is: {second_largest1}")

numbers2 = [10, 5, 10, 15, 20, 15]
second_largest2 = find_second_largest(numbers2)
print(f"Second largest number in {numbers2} is: {second_largest2}")

numbers3 = [1,1,1,1,1]
second_largest3 = find_second_largest(numbers3)
print(f"Second largest number in {numbers3} is: {second_largest3}")

numbers4 = [1]
second_largest4 = find_second_largest(numbers4)
print(f"Second largest number in {numbers4} is: {second_largest4}")

numbers5 = [1,2]
second_largest5 = find_second_largest(numbers5)
print(f"Second largest number in {numbers5} is: {second_largest5}")
```

Output :

Second largest number in [12, 45, 2, 41, 31, 10, 6, 4] is: 41

Check Anagram

Objective: Check if two strings are anagrams (contain the same characters in any order).

Input: Two strings.

Program :

```
def are_anagrams(str1, str2):
  str1 = str1.replace(" ", "").lower()  # Remove spaces and lowercase
  str2 = str2.replace(" ", "").lower()  # Remove spaces and lowercase

  return sorted(str1) == sorted(str2)

string1 = "listen"
string2 = "silent"

if are_anagrams(string1, string2):
  print(True)
else:
  print(False)

string3 = "hello"
string4 = "world"

if are_anagrams(string3, string4):
    print(True)
else:
    print(False)
```

Output :

True

False

AI-Based Tic-Tac-Toe

Program :

```
import math
```

```
def print_board(board):
    """Prints the Tic-Tac-Toe board."""
    for row in board:
        print(" | ".join(row))
        print("-" * 9)

def check_winner(board, player):
    """Checks if a player has won."""
    for row in board:
        if all(cell == player for cell in row):
            return True
    for col in range(3):
        if all(board[row][col] == player for row in range(3)):
            return True
    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i in range(3)):
        return True
    return False

def is_board_full(board):
    """Checks if the board is full."""
    return all(cell != " " for row in board for cell in row)

def get_available_moves(board):
    """Returns a list of available moves."""
    moves = []
    for row in range(3):
        for col in range(3):
            if board[row][col] == " ":
                moves.append((row, col))
    return moves
```

```
def minimax(board, depth, maximizing_player):
    """Minimax algorithm for AI decision-making."""
    if check_winner(board, "X"):
        return -1
    if check_winner(board, "O"):
        return 1
    if is_board_full(board):
        return 0

    if maximizing_player:
        max_eval = -math.inf
        for move in get_available_moves(board):
            row, col = move
            board[row][col] = "O"
            eval = minimax(board, depth + 1, False)
            board[row][col] = " "
            max_eval = max(max_eval, eval)
        return max_eval
    else:
        min_eval = math.inf
        for move in get_available_moves(board):
            row, col = move
            board[row][col] = "X"
            eval = minimax(board, depth + 1, True)
            board[row][col] = " "
            min_eval = min(min_eval, eval)
        return min_eval

def get_best_move(board):
    """Returns the best move for the AI."""
```

```
    best_move = None
    best_eval = -math.inf
    for move in get_available_moves(board):
        row, col = move
        board[row][col] = "O"
        eval = minimax(board, 0, False)
        board[row][col] = " "
        if eval > best_eval:
            best_eval = eval
            best_move = move
    return best_move

def play_tic_tac_toe():
    """Plays the Tic-Tac-Toe game."""
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        # User's turn
        while True:
            try:
                row = int(input("Enter row (0, 1, 2): "))
                col = int(input("Enter column (0, 1, 2): "))
                if 0 <= row <= 2 and 0 <= col <= 2 and board[row][col] == " ":
                    board[row][col] = "X"
                    break
                else:
                    print("Invalid move. Try again.")
            except ValueError:
                print("Invalid input. Enter integers.")
```

```
        print_board(board)
        if check_winner(board, "X"):
            print("You win!")
            break
        if is_board_full(board):
            print("It's a tie!")
            break

        # AI's turn
        print("AI's turn...")
        ai_move = get_best_move(board)
        if ai_move:
            row, col = ai_move
            board[row][col] = "O"
            print_board(board)
            if check_winner(board, "O"):
                print("AI wins!")
                break
            if is_board_full(board):
                print("It's a tie!")
                break
        else:
            print("It's a tie!")
            break

play_tic_tac_toe()
```

Output :

```
Welcome to Tic-Tac-Toe!
  |   |
---------
  |   |
---------
  |   |
---------
Enter row (0, 1, 2): 0
Enter column (0, 1, 2): 0
X |   |
---------
  |   |
---------
  |   |
---------
AI's turn...
X |   | O
---------
  |   |
---------
  |   |
---------
Enter row (0, 1, 2): 1
Enter column (0, 1, 2): 1
X |   | O
---------
  | X |
---------
  |   |
---------
AI's turn...
X |   | O
---------
  | X |
---------
O |   |
---------
Enter row (0, 1, 2): 2
Enter column (0, 1, 2): 1
X |   | O
---------
  | X |
---------
O | X |
---------
AI's turn...
X | O | O
---------
  | X |
---------
O | X |
---------
Enter row (0, 1, 2): 1
Enter column (0, 1, 2): 0
X | O | O
---------
X | X |
---------
O | X |
---------
AI wins!
```