```
# Task-17
def print_multiplication_table(n):
    for i in range(1, 11):
        print(f"{n} x {i} = {n * i}")

# Example usage
n = int(input("Enter a number: "))
print_multiplication_table(n)
```

```
Enter a number: 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
```

```
# Task-18
def swap_numbers_add_sub(a, b):
    a = a + b
    b = a - b
    a = a - b
    return a, b

# Example
a, b = 5, 10
print("Before swap: a =", a, "b =", b)
a, b = swap_numbers_add_sub(a, b)
print("After swap: a =", a, "b =", b)
```

```
Before swap: a = 5 b = 10
After swap: a = 10 b = 5
```

```
#Task-19
def is_substring(s1, s2):
    return s2 in s1

# Example
s1 = "hello world"
s2 = "world"
print(is_substring(s1, s2))  # Output: True

s2 = "python"
print(is_substring(s1, s2))  # Output: False
```

```
True
False
```

```
#Task 20
def decimal_to_binary_bin(n):
    return bin(n)[2:]  # Remove the '0b' prefix
```

```
# Example
n = 10
print(decimal_to_binary_bin(n))  # Output: 1010
```

```
1010
```

```
#Task 21
def add_matrices(matrix1, matrix2):
    # Ensure matrices have the same dimensions
    if len(matrix1) != len(matrix2) or len(matrix1[0]) != len(matrix2[0]):
        raise ValueError("Matrices must have the same dimensions")

    # Add corresponding elements
    result = []
    for i in range(len(matrix1)):  # Iterate over rows
        row = []
        for j in range(len(matrix1[0])):  # Iterate over columns
            row.append(matrix1[i][j] + matrix2[i][j])
        result.append(row)
    return result

# Example
matrix1 = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
matrix2 = [
    [9, 8, 7],
    [6, 5, 4],
    [3, 2, 1]
]

result = add_matrices(matrix1, matrix2)
for row in result:
    print(row)
```

```
[10, 10, 10]
[10, 10, 10]
[10, 10, 10]
```

```
#Task 22
def multiply_matrices(A, B):
    # Check if matrix multiplication is possible
    if len(A[0]) != len(B):
        raise ValueError("Number of columns in A must equal the number of rows in B")

    # Initialize the result matrix with zeros
    result = [[0 for _ in range(len(B[0]))] for _ in range(len(A))]

    # Perform matrix multiplication
    for i in range(len(A)):  # Iterate over rows of A
        for j in range(len(B[0])):  # Iterate over columns of B
            for k in range(len(B)):  # Iterate over rows of B
                result[i][j] += A[i][k] * B[k][j]
    return result
```

```
# Example
A = [
    [1, 2, 3],
    [4, 5, 6]
]

B = [
    [7, 8],
    [9, 10],
    [11, 12]
]

result = multiply_matrices(A, B)
for row in result:
    print(row)
```

```
[58, 64]
[139, 154]
```

```
#Task 23
def find_second_largest_sort(nums):
    if len(nums) < 2:
        raise ValueError("List must contain at least two distinct numbers")

    nums = list(set(nums))  # Remove duplicates
    nums.sort(reverse=True)  # Sort in descending order
    return nums[1]  # Second largest is at index 1

# Example
nums = [10, 20, 4, 45, 99, 99]
print(find_second_largest_sort(nums))  # Output: 45
```

```
45
```

```
#Task 24
def are_anagrams_sorted(str1, str2):
    return sorted(str1) == sorted(str2)

# Example
str1 = "listen"
str2 = "silent"
print(are_anagrams_sorted(str1, str2))  #  True

str1 = "hello"
str2 = "world"
print(are_anagrams_sorted(str1, str2))  # False
```

```
True
False
```

```
# 3 Ai based TIC TAC TOE
import math

def print_board(board):
    for row in board:
        print(" | ".join(row))
        print("-" * 9)
```

```
def is_winner(board, player):
    # Check rows, columns, and diagonals
    for row in board:
        if all(cell == player for cell in row):
            return True

    for col in range(3):
        if all(row[col] == player for row in board):
            return True

    if all(board[i][i] == player for i in range(3)) or all(board[i][2 - i] == player for i
        return True

    return False


def is_full(board):
    return all(cell != " " for row in board for cell in row)


def get_empty_cells(board):
    return [(i, j) for i in range(3) for j in range(3) if board[i][j] == " "]


def minimax(board, depth, is_maximizing):
    if is_winner(board, "O"):  # AI wins
        return 10 - depth
    if is_winner(board, "X"):  # User wins
        return depth - 10
    if is_full(board):  # Draw
        return 0

    if is_maximizing:
        best_score = -math.inf
        for (i, j) in get_empty_cells(board):
            board[i][j] = "O"
            score = minimax(board, depth + 1, False)
            board[i][j] = " "
            best_score = max(best_score, score)
        return best_score
    else:
        best_score = math.inf
        for (i, j) in get_empty_cells(board):
            board[i][j] = "X"
            score = minimax(board, depth + 1, True)
            board[i][j] = " "
            best_score = min(best_score, score)
        return best_score


def best_move(board):
    best_score = -math.inf
    move = None
    for (i, j) in get_empty_cells(board):
        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 tic_tac_toe():
    board = [[" " for _ in range(3)] for _ in range(3)]
    print("Welcome to Tic-Tac-Toe!")
    print_board(board)

    while True:
        # User move
        user_move = None
        while user_move not in get_empty_cells(board):
            try:
                row, col = map(int, input("Enter your move (row and column: 0 1): ").split(
                user_move = (row, col)
                if user_move not in get_empty_cells(board):
                    print("Invalid move. Try again.")
            except ValueError:
                print("Invalid input. Enter row and column as two integers separated by spa

        board[user_move[0]][user_move[1]] = "X"
        print("\nYour move:")
        print_board(board)

        if is_winner(board, "X"):
            print("You win!")
            break
        if is_full(board):
            print("It's a draw!")
            break

        # AI move
        ai_move = best_move(board)
        board[ai_move[0]][ai_move[1]] = "O"
        print("\nAI's move:")
        print_board(board)

        if is_winner(board, "O"):
            print("AI wins!")
            break
        if is_full(board):
            print("It's a draw!")
            break


# Run the game
tic_tac_toe()
```

```
Welcome to Tic-Tac-Toe!
  |   |
---------
  |   |
---------
  |   |
---------
Enter your move (row and column: 0 1): 1,0
Invalid input. Enter row and column as two integers separated by space.
Enter your move (row and column: 0 1): 0 1
```

```
Your move:
  | X |
---------
  |   |
---------
  |   |
---------

AI's move:
O | X |
---------
  |   |
---------
  |   |
---------
Enter your move (row and column: 0 1): 1 1

Your move:
O | X |
---------
  | X |
---------
  |   |
---------

AI's move:
O | X |
---------
  | X |
---------
  | O |
---------
Enter your move (row and column: 0 1): 2 0

Your move:
O | X |
---------
  | X |
---------
X | O |
---------

AI's move:
O | X | O
---------
  | X |
---------
```