

```
In [1]: #Task-3
In [2]: ''' 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.'''
Out[2]: ' 17. Table of a Number\n Objective: Print the multiplication table for a giv
        en number nnn.\n Input: An integer nnn.\n Output: Multiplication table from 1
        11 to 101010.\n Hint: Use a loop to iterate through values 1 to 10 and multip
        ly by nnn.'
In [3]: # Solution
        n = int(input("Enter a number: "))
        for i in range(1, 11):
            print(f"{n} x {i} = {n * i}")
       326 \times 1 = 326
       326 \times 2 = 652
       326 \times 3 = 978
       326 \times 4 = 1304
       326 \times 5 = 1630
       326 \times 6 = 1956
       326 \times 7 = 2282
       326 \times 8 = 2608
       326 \times 9 = 2934
       326 \times 10 = 3260
In [4]: '''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 =
Out[4]: '18. Swap Two Numbers\n Objective: Swap two numbers without using a third var
        iable.\n Input: Two integers aaa and bbb.\n Output: Swapped values of aaa and
        bbb.\n Hint: Use arithmetic operations like addition and subtraction or XOR
        (a, b = b, a).'
In [5]: # Solution
        a = int(input("Enter first number: "))
        b = int(input("Enter second number: "))
        # Swapping without a third variable
        a = a + b
        b = a - b
        a = a - b
        print("After swapping:")
        print("First number:", a)
        print("Second number:", b)
```

```
After swapping:
       First number: 56
       Second number: 12
 In [6]: '''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.'''
Out[6]: "19. Check Substring\n Objective: Determine if one string is a substring of a
         nother.\n Input: Two strings slsls1 (main string) and s2s2s2 (substring).\n 0
         utput: True if s2s2s2 is a substring of s1s1s1, otherwise False.\n Hint: Use
         Python's in operator or string slicing to search for substrings."
 In [7]: # Solution
         s1 = input("Enter the main string: ")
         s2 = input("Enter the substring: ")
         print("Is substring:", s2 in s1)
       Is substring: False
 In [9]: '''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 remainder
Out[9]: '20. Decimal to Binary\n Objective: Convert a decimal number to its binary re
         presentation.\n Input: An integer nnn.\n Output: A string representing the bi
         nary equivalent.\n Hint: Use the bin() function or repeatedly divide nnn by
         2, storing remainders.'
In [13]: # Solution
         n = int(input("Enter a decimal number: "))
         binary = ""
         if n == 0:
             binary = "0"
         else:
             while n > 0:
                 binary = str(n % 2) + binary
                 n //= 2
         print("Binary representation:", binary)
       Binary representation: 1.00.01.00.0
In [14]: '''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 correspond

elements.'''

Out[14]: '21. Matrix Addition\nObjective: Add two matrices of the same dimensions.\nIn put: Two 2D lists (matrices) of integers.\nOutput: A 2D list containing the s um of corresponding elements.\nHint: Use nested loops to iterate through rows and columns, adding corresponding\nelements.'

```
In [20]: # Solution
         rows = int(input("Enter number of rows: "))
         cols = int(input("Enter number of columns: "))
         print("Enter elements of first matrix:")
         matrix1 = [[int(input(f"Element [{i}][{j}]: ")) for j in range(cols)] for i in
         print("Enter elements of second matrix:")
         matrix2 = [[int(input(f"Element [{i}][{j}]: ")) for j in range(cols)] for i in
         # Add matrices
         result = [[matrix1[i][j] + matrix2[i][j] for j in range(cols)] for i in range(
         print("Sum of matrices:")
         for row in result:
             print(row)
       Enter elements of first matrix:
       Enter elements of second matrix:
       Sum of matrices:
        [73, 68]
        [73, 93]
In [16]: '''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 r
          BBB.
          Output: A 2D list representing the product matrix.
          Hint: Multiply elements row-by-column and sum for each position in the result
Out[16]: '22. Matrix Multiplication\n Objective: Multiply two matrices AAA and BBB.\n
         Input: Two 2D lists where the number of columns in AAA equals the number of r
         ows in\n BBB.\n Output: A 2D list representing the product matrix.\n Hint: Mu
         ltiply elements row-by-column and sum for each position in the result matri
         х.'
In [21]: # Solution
         # Input dimensions
         rows A = int(input("Enter number of rows in Matrix A: "))
         cols A = int(input("Enter number of columns in Matrix A: "))
         rows B = int(input("Enter number of rows in Matrix B: "))
         cols B = int(input("Enter number of columns in Matrix B: "))
         # Check multiplication validity
         if cols A != rows B:
             print("Matrix multiplication not possible. Columns of A must equal rows of
         else:
             print("Enter elements of Matrix A:")
```

 $A = [[int(input(f"A[\{i\}][\{j\}]: ")) for j in range(cols_A)] for i in range($

```
print("Enter elements of Matrix B:")
B = [[int(input(f"B[{i}][{j}]: ")) for j in range(cols_B)] for i in range(
# Initialize result matrix with 0s
result = [[0 for _ in range(cols_B)] for _ in range(rows_A)]

# Multiply A and B
for i in range(rows_A):
    for j in range(cols_B):
        for k in range(cols_A):
            result[i][j] += A[i][k] * B[k][j]

print("Product of matrices:")
for row in result:
    print(row)
```

Matrix multiplication not possible. Columns of A must equal rows of B.

```
In [22]:
    '''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.'''
```

Out[22]: '23. Find Second Largest\n Objective: Find the second largest number in a lis t.\n Input: A list of integers.\n Output: The second largest integer.\n Hint: Use sorting or iterate to find the largest, then the second largest.'

```
In [24]: # Solution
         numbers = list(map(int, input("Enter list of numbers separated by space: ").sp
         if len(numbers) < 2:</pre>
             print("List must contain at least two numbers.")
         else:
             first = second = float('-inf')
             for num in numbers:
                  if num > first:
                      second = first
                      first = num
                  elif first > num > second:
                      second = num
              if second == float('-inf'):
                 print("No second largest number found.")
             else:
                  print("Second largest number:", second)
```

Second largest number: 54

```
In [25]: ''' 24. Check Anagram
   Objective: Check if two strings are anagrams (contain the same characters in order).
   Input: Two strings.
   Output: True if anagrams, otherwise False.
```

Hint: Use sorted() on both strings or count character occurrences using a did

Out[25]: ' 24. Check Anagram\n Objective: Check if two strings are anagrams (contain the same characters in any\n order).\n Input: Two strings.\n Output: True if a nagrams, otherwise False.\n Hint: Use sorted() on both strings or count character occurrences using a dictionary.'

```
In [26]: # Solution
    strl = input("Enter first string: ").replace(" ", "").lower()
    str2 = input("Enter second string: ").replace(" ", "").lower()
    print("Are anagrams:", sorted(str1) == sorted(str2))
```

Are anagrams: False

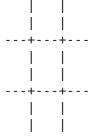
Out[27]: ' 3. AI-Based Tic-Tac-Toe\n • Description: Create a Tic-Tac-Toe game where the computer plays against the user and\n uses a minimax algorithm to make decisions.\n • Challenges:\n ○ Implement AI logic with decision trees.\n ○ Handle edge cases like a full board or winning moves.\n ○ Provide a user-friendly in terface.\n • Skills: Game theory, recursion, and strategic thinking.'

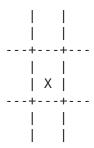
```
In [5]: import math
        def print board(board):
            print()
            for i in range(3):
                row = " | "
                print(row)
                print(f" {board[3*i]} | {board[3*i+1]} | {board[3*i+2]} ")
                if i < 2:
                    print("---+--")
            print()
        # Check for a winner or draw
        def check winner(board):
            win combinations = [
                [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows
                [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns
                [0, 4, 8], [2, 4, 6]
                                                # diagonals
            1
            for combo in win combinations:
                a, b, c = combo
                if board[a] == board[b] == board[c] != ' ':
```

```
return board[a]
    if ' ' not in board:
        return 'Draw'
    return None
# Minimax algorithm for the AI to choose best move
def minimax(board, is maximizing):
   winner = check winner(board)
    if winner == '0':
        return 1
    elif winner == 'X':
        return -1
    elif winner == 'Draw':
        return 0
    if is_maximizing:
        best score = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = '0'
                score = minimax(board, False)
                board[i] = ' '
                best score = max(score, best score)
        return best score
    else:
        best score = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = minimax(board, True)
                board[i] = ' '
                best score = min(score, best score)
        return best score
# AI picks best move
def ai move(board):
    best score = -math.inf
   move = None
    for i in range(9):
        if board[i] == ' ':
            board[i] = '0'
            score = minimax(board, False)
            board[i] = ' '
            if score > best_score:
                best score = score
                move = i
    return move
# Main game function
def play_game():
    board = [' '] * 9
```

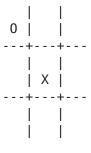
```
print("Welcome to Tic-Tac-Toe!")
     print("You are X, AI is 0.")
     print board(board)
     while True:
         # User move
         while True:
             try:
                  user move = int(input("Enter your move (1-9): ")) - 1
                  if 0 <= user move <= 8 and board[user move] == ' ':</pre>
                      board[user move] = 'X'
                     break
                  else:
                      print("Spot taken or invalid. Try again.")
             except (ValueError, IndexError):
                  print("Invalid input. Enter a number from 1 to 9.")
         print_board(board)
         if check winner(board):
             break
         # AI move
         print("AI is thinking...")
         move = ai move(board)
         board[move] = '0'
         print board(board)
         if check winner(board):
             break
     result = check winner(board)
     if result == 'Draw':
         print("It's a draw!")
     else:
         print(f"{result} wins!")
 if name == " main ":
     play game()
Welcome to Tic-Tac-Toe!
```

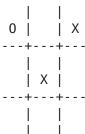
You are X, AI is 0.



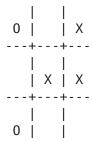


AI is thinking...





AI is thinking...



AI is thinking...



0 wins!

In [6]: '''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 det optimal moves. This restriction forces students to implement and understand t logic of decision-making algorithms, ensuring the AI plays optimally and is r random or rudimentary. This will deepen their understanding of decision trees and game theory.
- Learning Outcome: Students will learn how to create intelligent agents in d gaining insight into search algorithms, recursion, and game strategy optimize
- Out[6]: '3. AI-Based Tic-Tac-Toe\n Restriction: Only use the minimax algorithm for AI decision-making.\n ● Reason: The minimax algorithm is a classic AI strateg y used in games to determine\n optimal moves. This restriction forces student s to implement and understand the core\n logic of decision-making algorithms, ensuring the AI plays optimally and is not\n random or rudimentary. This will deepen their understanding of decision trees, recursion,\n and game theory.\n Learning Outcome: Students will learn how to create intelligent agents in g ames,\n gaining insight into search algorithms, recursion, and game strategy optimization.'

```
In [7]: import math
        # Print the board in a 3x3 grid format
        def print board(board):
           print()
           for i in range(3):
               print(" | ")
               print(f" {board[3*i]} | {board[3*i+1]} | {board[3*i+2]} ")
               if i < 2:
                   print("---+--")
           print()
```

```
# Check for a winner or draw
def check winner(board):
   win_combinations = [
        [0, 1, 2], [3, 4, 5], [6, 7, 8], # rows
        [0, 3, 6], [1, 4, 7], [2, 5, 8], # columns
        [0, 4, 8], [2, 4, 6]
                                         # diagonals
    for combo in win combinations:
        a, b, c = combo
        if board[a] == board[b] == board[c] != ' ':
            return board[a]
    if ' ' not in board:
        return 'Draw'
    return None
# Minimax algorithm
def minimax(board, is_maximizing):
   winner = check_winner(board)
    if winner == '0':
        return 1
    elif winner == 'X':
       return -1
    elif winner == 'Draw':
       return 0
    if is maximizing:
        best score = -math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = '0'
                score = minimax(board, False)
                board[i] = ' '
                best score = max(best score, score)
        return best score
    else:
        best score = math.inf
        for i in range(9):
            if board[i] == ' ':
                board[i] = 'X'
                score = minimax(board, True)
                board[i] = ' '
                best_score = min(best_score, score)
        return best_score
# AI picks the best move
def ai move(board):
    best score = -math.inf
    move = None
    for i in range(9):
        if board[i] == ' ':
            board[i] = '0'
            score = minimax(board, False)
            board[i] = ' '
```

```
if score > best score:
                best score = score
                move = i
    return move
# Main game loop
def play game():
    board = [' '] * 9
    print("Welcome to Tic-Tac-Toe!")
    print("You are X. AI is O. Try to beat the AI (you can't!)")
    print board(board)
   while True:
        # Player move
        while True:
            try:
                user move = int(input("Enter your move (1-9): ")) - 1
                if 0 <= user_move <= 8 and board[user_move] == ' ':</pre>
                    board[user move] = 'X'
                    break
                else:
                    print("Invalid or taken spot. Try again.")
            except:
                print("Please enter a number between 1 and 9.")
        print board(board)
        if check winner(board):
            break
        # AI move
        print("AI is thinking...")
        move = ai move(board)
        board[move] = '0'
        print board(board)
        if check winner(board):
            break
    # Final result
    result = check winner(board)
    if result == 'Draw':
       print("It's a draw!")
    else:
        print(f"{result} wins!")
# Run the game
if __name__ == "__main__":
    play game()
```

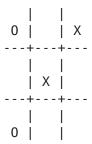
Welcome to Tic-Tac-Toe!
You are X. AI is 0. Try to beat the AI (you can't!)

 + - 	
 X	-

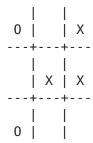
AI is thinking...

		-
0		
	 X +	

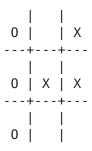
AI is thinking...



Invalid or taken spot. Try again.



AI is thinking...



0 wins!

In []: