ASSIGNMENT - 1

NAME: KOTLA LAKSHMI PRIYANKA

COURSE: DATA SCIENCE AND GEN AI LLMS

HALL TICKET NO – 2406DGAL126

DATE: 03-11-2024

TABLE OF CONTENTS:

Question 1:

Number game between user and computer. The user starts by entering either 1 or 2 or 3 digits starting from 1 sequentially. The computer can return either 1 or 2 or 3 next digits in sequence, starting from the max number played by the user. User enters the next 1 or 2 or 3 next digits in sequence, starting from the max number played by the computer. Whoever reaches 20 first wins the game.

Note:

- the numbers should be in sequence starting from 1.

- minimum number user or computer should pick is at least 1 digit in sequence

- maximum number user or computer can pick only 3 digits in sequence

Code:

```
def nearest_multiple(num):
    """Returns the nearest multiple of 4 greater than or equal to given
number."""
    if num >= 4:
        return num + (4 - (num % 4))  # Next multiple of 4
    else:
        return 4  # If num < 4, return 4

def lose():
    """Displays losing message and exits."""
    print("\n\nYOU LOSE!")
    print("Better luck next time!")
    exit(0)

def check_consecutive(xyz):
    """Checks if numbers are consecutive."""
    for i in range(1, len(xyz)):
        if xyz[i] - xyz[i - 1] != 1:  # Not consecutive
            return False
    return True  # All consecutive

def play_game():
    """Main function for number game."""
    print("Number game between user and computer")
    numbers_played = []  # Track played numbers
    last_number = 0

    while True:
        print("Current numbers:", numbers_played)

        # Player's turn
        print("\nYour Turn.")
        user_input = int(input("How many numbers do you wish to enter (1-3)? "))

        if 1 <= user_input <= 3:
            print("Enter the values:")
            player_numbers = []  # Player's numbers
            for _ in range(user_input):
                number = int(input('> '))
                player_numbers.append(number)
                numbers_played.append(number)
                last_number = number

            if not check_consecutive(player_numbers):  # Check if consecutive
                print("\nYou did not input consecutive integers.")
```

```
                lose()

            if last_number >= 20:  # Check if player won
                print("\n\nCONGRATULATIONS!!! You've won!")
                break

            # Computer's turn
            computer_numbers = []
            computer_pick = min(3, 20 - last_number)  # Numbers computer can
play
            for j in range(1, computer_pick + 1):
                computer_numbers.append(last_number + j)
                numbers_played.append(last_number + j)
            print("Computer played:", computer_numbers)
            last_number = numbers_played[-1]

            if last_number >= 20:  # Check if computer won
                print("\nComputer Wins!!!")
                break

        else:
            print("Invalid input. Enter 1, 2, or 3 numbers.")
            lose()

# Start the game
if __name__ == "__main__":
    play_game()
```

Code Explanation:

1. ‘nearest_multiple’ Function

- Finds the next multiple of 4 greater than or equal to ‘num’.

2. ‘lose’ Function

- Shows a loss message and exits the game.

- Called if the player enters non-consecutive numbers or chooses an invalid number of entries (anything other than 1, 2, or 3).

3. ‘check_consecutive’ Function

- Checks if the numbers in ‘xyz’ are consecutive.

- Returns ‘True’ if consecutive, ‘False’ otherwise.

4. ‘play_game’ Function

- Main game function that alternates turns between the player and computer until one reaches 20 or more.

- Game Setup:

- ‘numbers_played’ keeps track of all numbers entered.

- ‘last_number’ stores the last number played.

- Player's Turn:

- Player selects how many numbers to enter (1-3).

- If the entries aren't consecutive, 'lose' is called.

- If the player reaches or exceeds 20, they win.

- Computer's Turn:

- Computer plays up to 3 consecutive numbers, starting from the last number.

- If it reaches 20 or more, the computer wins.

- Invalid Input:

- If the player inputs a number of entries outside the range of 1-3, the game calls 'lose', displaying a loss message and exiting.

Output:

```
Number game between user and computer
Current numbers:  []

Your Turn.
How many numbers do you wish to enter (1-3)? 2
Now enter the values:
> 1
> 2
Computer played: [3, 4]

Current numbers:  [1, 2, 3, 4]

Your Turn.
How many numbers do you wish to enter (1-3)? 3
Now enter the values:
> 5
> 6
> 7
Computer played: [8, 9, 10]

Current numbers:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

Your Turn.
How many numbers do you wish to enter (1-3)? 2
Now enter the values:
> 11
> 12
Computer played: [13, 14, 15]

Current numbers:  [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

Your Turn.
How many numbers do you wish to enter (1-3)? 3
Now enter the values:
> 16
> 17
> 18
Computer played: [19, 20]

Computer Wins!!!
```

Output Images:

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS                                                    Python

PS E:\jntuh_python> & C:/Users/VG/AppData/Local/Programs/Python/Python312/python.exe e:/jntuh_python/python/assignment/1new.py
Number game between user and computer
Current numbers: []

Your Turn.
How many numbers do you wish to enter (1-3)? 2
Enter the values:
> 1
> 2
Computer played: [3, 4, 5]
Current numbers: [1, 2, 3, 4, 5]

Your Turn.
How many numbers do you wish to enter (1-3)? 3
Enter the values:
> 6
> 7
> 8
Computer played: [9, 10, 11]
Current numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]

Your Turn.
How many numbers do you wish to enter (1-3)? 1
Enter the values:
> 12
Computer played: [13, 14, 15]
Current numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]

Your Turn.
How many numbers do you wish to enter (1-3)? 1
Enter the values:
> 16
Computer played: [17, 18, 19]
Ln 71, Col 1   Spaces: 4   UTF-8   CRLF   {} Python   3.12.4 64-bit   Prettier
```

```
Computer played: [17, 18, 19]
Current numbers: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Your Turn.
How many numbers do you wish to enter (1-3)? 1
Enter the values:
> 20


CONGRATULATIONS!!! You've won!
PS E:\jntuh_python>
Ln 71, Col 1   Spaces: 4   UTF-8   CRLF   {} Python   3.12.4 64-bit   Prettier
```

Question 2:

Develop a function called ncr(n,r) which computes r-combinations of n-distinct object . use this function to print pascal triangle, where number of rows is the input.

Code:

```
def calculate_factorial(number):
    """Calculate factorial of a number."""
    if number == 0 or number == 1:
        return 1
    result = 1
    for i in range(2, number + 1):
        result = i  # Multiply result by each number up to 'number'
    return result

def calculate_combinations(n, r):
    """Calculate combinations nCr."""
    if r < 0 or r > n:
        return 0
    return calculate_factorial(n) // (calculate_factorial(r)
calculate_factorial(n - r))

def display_pascal_triangle(number_of_rows):
    """Print Pascal's Triangle."""
    if number_of_rows <= 0:
        return  # Exit if input is zero or negative

    for row in range(number_of_rows):
        print(" "  (number_of_rows - row), end="")  # Leading spaces
        for column in range(row + 1):
            print(calculate_combinations(row, column), end=" ")
        print()  # New line after each row

# Run the program
if __name__ == "__main__":
    rows = int(input("Enter the number of rows for Pascal's Triangle: "))
    display_pascal_triangle(rows)
```

Code Explanation:

1. calculate_factorial(number)

- This function calculates the factorial of ‘number’.

- If ‘number’ is 0 or 1, it returns 1 (since ‘0!’ and ‘1!’ are both 1).

- For other numbers, it multiplies all integers up to ‘number’.

2. calculate_combinations(n, r)

- Calculates combinations (nCr), representing the number of ways to choose ‘r’ items from ‘n’.

- Formula: nCr = n! / (r! (n - r)!)

- Uses calculate_factorial to get factorials for ‘n’, ‘r’, and ‘n - r’.

- Returns 0 if ‘r’ is out of range (e.g., less than 0 or greater than ‘n’).

3. display_pascal_triangle(number_of_rows)

- Prints Pascal's Triangle with ‘number_of_rows’ rows.
- For each row ‘n’, it calculates combinations nC0, nC1, ..., nCn.
- Adds spaces to format the triangle visually.

4. Main Execution

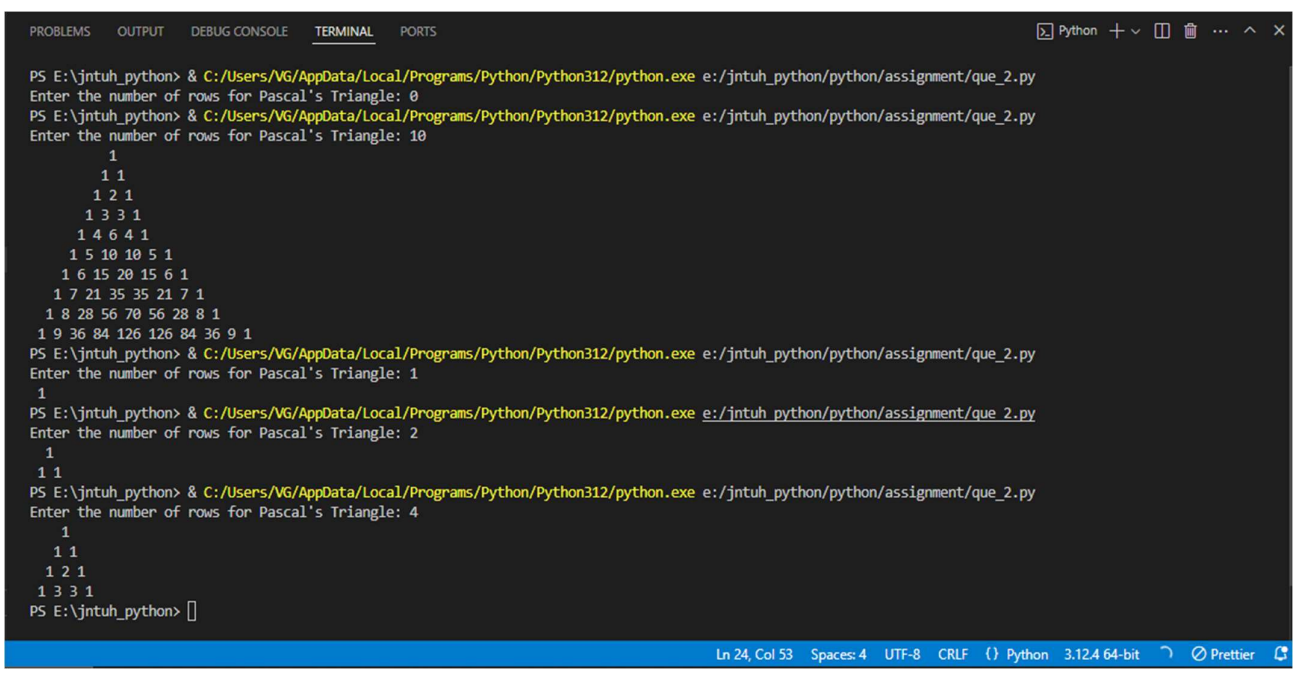
- Prompts the user for the number of rows.
- Calls display_pascal_triangle(rows) to print Pascal’s Triangle.

Output:

For an input of ‘5’ rows, the output of Pascal’s Triangle :

```
Enter the number of rows for Pascal's Triangle: 5
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
```

Outputs Image:



Question 3:

Read a list of n numbers during runtime. Write a Python program to print the repeated elements with frequency count in a list.

Code:

```
def count_frequencies(numbers):
    """Count the frequency of each element in the list."""
    frequency = {}  # Store frequencies

    # Count occurrences
    for num in numbers:
        if num in frequency:
            frequency[num] += 1  # Increment count
        else:
            frequency[num] = 1  # Initialize count

    # Print frequencies
    for element, count in frequency.items():
        print(f"Element {element} has come {count} times")

# Main execution
if __name__ == "__main__":
    user_input = input("Enter a list of numbers separated by commas: ")
    if user_input.strip():  # Check if input is not empty
        numbers = list(map(int, user_input.split(',')))  # Convert input to a
list of integers
        count_frequencies(numbers)  # Count and print frequencies
    else:
        print("No input provided.")  # Handle empty input
```

Code Explanation:

1. count_frequencies(numbers)

- This function counts how often each unique number appears in a list.
- It uses a dictionary called 'frequency' to store each number (as a key) and its count (as a value).
- Counting Logic:
  - For each number in the list:
    - If it's already in the dictionary, its count goes up by 1.
    - If it's not there, it gets added with a count of 1.
- It prints each number and its frequency in the format:

  "Element <number> has come <count> times".

2. Main Program Execution

- The program asks the user to input numbers as a comma-separated string.
- It splits the string by commas and converts it into a list of integers.

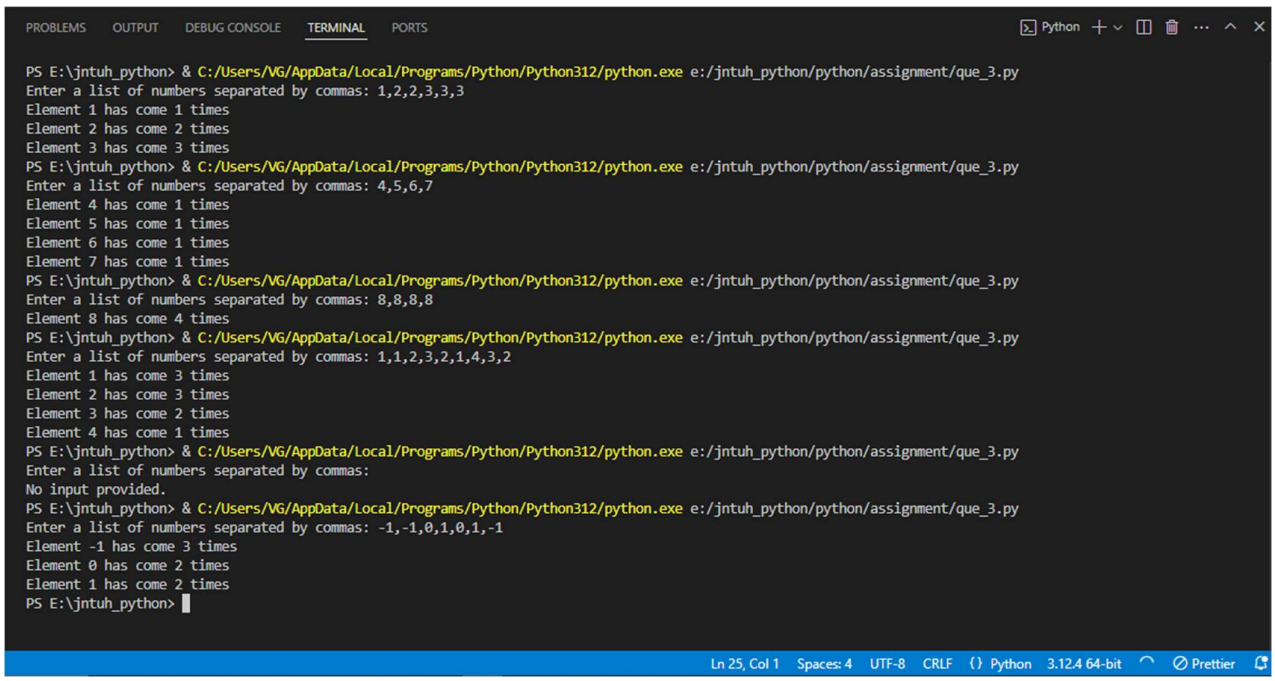
- Then, it calls 'count_frequencies(numbers)' to show the counts.

Output:

```
Enter a list of numbers separated by commas: 2,1,2,3,4,5,1,3,6,2,3,4


Element 2 has come 3 times
Element 1 has come 2 times
Element 3 has come 3 times
Element 4 has come 2 times
Element 5 has come 1 times
Element 6 has come 1 times
```

Outputs Image:



Question 4:-

Develop a python code to read matric A of order 2X2 and Matrix B of order 2X2 from a file and perform the addition of Matrices A & B and Print the results.

Code:

```
def read_matrix_from_file(filename):
    # Read two matrices from a file and return them
    matrices = {}
    current_matrix = None  # Track current matrix

    with open(filename, 'r') as file:  # Open file
        for line in file:
            line = line.strip()  # Clean line
            if line.startswith('A='):  # Matrix A
                current_matrix = 'A'
                continue
            elif line.startswith('B='):  # Matrix B
                current_matrix = 'B'
                continue

            if current_matrix:
                # Convert line to list of integers
                row = list(map(int, line.split()))
                if current_matrix not in matrices:
                    matrices[current_matrix] = []  # Initialize list
                matrices[current_matrix].append(row)  # Add row

    return matrices['A'], matrices['B']  # Return matrices

def add_matrices(matrix_a, matrix_b):
    # Add two 2x2 matrices
    result = []  # Resultant matrix
    for i in range(2):  # Two rows
        # Sum corresponding elements
        result_row = [matrix_a[i][j] + matrix_b[i][j] for j in range(2)]
        result.append(result_row)  # Add row to result
    return result

def print_matrix(matrix, name):
    # Print matrix with its name
    print(f"Matrix {name}:")
    for row in matrix:
        print(' '.join(map(str, row)))  # Join row elements
    print()  # Newline for spacing

if __name__ == "__main__":
    filename = 'matrices.txt'  # File with matrices
    print("Reading matrices from", filename)

    # Read matrices from file
    matrix_a, matrix_b = read_matrix_from_file(filename)

    # Print matrices before addition
    print_matrix(matrix_a, 'A')  # Matrix A
    print_matrix(matrix_b, 'B')  # Matrix B

    # Add matrices
    result_matrix = add_matrices(matrix_a, matrix_b)

    # Print resultant matrix
    print("Resultant Matrix after addition:")
    print_matrix(result_matrix, 'Result')
```

Code Explanation:

1. read_matrix_from_file(filename)

- Reads matrices (A) and (B) from a file. Lines starting with "A=" or "B=" indicate the matrix, followed by rows of numbers.

- It converts each row into a list of integers and assigns them to the respective matrix.

- Returns both matrices as a tuple: (matrix A, matrix B).

2. add_matrices(matrix_a, matrix_b)

- Adds two 2x2 matrices (A and B).

- Calculates the sum of corresponding elements and stores them in a new matrix called ‘result’.

- Returns the result matrix.

3. print_matrix(matrix, name)

- Takes a matrix and its name, then prints the matrix with the name above it (e.g., "Matrix A:").

4. Main Program Execution

- Reads matrices (A) and (B) from a file called ‘matrices.txt’.

- Prints both matrices.

- Adds the matrices and prints the result.

Output:

```
Reading matrices from matrices.txt
Matrix A:
1 2
3 4

Matrix B:
5 6
7 8

Resultant Matrix after addition:
Matrix Result:
6 8
10 12
```

Outputs Image:

matrices.txt

```
A=
1 3
4 5

B=
1 2
3 4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\jntuh_python> & C:/Users/VG/AppData/Local/Programs/Python/Python312/python.exe e:/jntuh_python/python/assignment/que_4.py
Reading matrices from matrices.txt
Matrix A:
1 3
4 5


Matrix B:
1 2
3 4

Resultant Matrix after addition:
Matrix Result:
2 5
7 9

PS E:\jntuh_python>
```

Ln 4, Col 1 Spaces: 4 UTF-8 CRLF Plain Text Prettier

matrices.txt

```
A=
-1 -2
-3 -4

B=
1 2
3 4
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS E:\jntuh_python> & C:/Users/VG/AppData/Local/Programs/Python/Python312/python.exe e:/jntuh_python/python/assignment/que_4.py
Reading matrices from matrices.txt
Matrix A:
-1 -2
-3 -4


Matrix B:
1 2
3 4

Resultant Matrix after addition:
Matrix Result:
0 0
0 0

PS E:\jntuh_python>
```

Ln 7, Col 4 Spaces: 4 UTF-8 CRLF Plain Text Prettier

Question 5:-

Write a program that overloads the + operator so that it can add two objects of the class Fraction.

Fraction can be considered of the for P/Q where P is the numerator and Q is the denominator

Code:

```
# Function to return gcd of a and b
def gcd(a, b):
    if a == 0:
        return b
    return gcd(b % a, a)

# Function to convert the obtained fraction into its simplest form
def lowest(num3, den3):
    common_factor = gcd(abs(num3), abs(den3))  # Use absolute values for
GCD
    den3 = den3 // common_factor  # Simplify denominator
    num3 = num3 // common_factor  # Simplify numerator

    # Adjust the signs: if denominator is negative, flip the signs
    if den3 < 0:
        num3 = -num3
        den3 = -den3

    return num3, den3  # Return the simplified fraction

# Function to add two fractions
def addFraction(num1, den1, num2, den2):
    den3 = (den1  den2) // gcd(den1, den2)  # Calculate common denominator
    num3 = (num1  (den3 // den1)) + (num2  (den3 // den2))  # Calculate
numerator
    return lowest(num3, den3)  # Ensure to simplify the fraction correctly

if __name__ == "__main__":
    # Input first fraction
    num1 = int(input("Enter the numerator of the first fraction: "))
    den1 = int(input("Enter the denominator of the first fraction: "))

    # Input second fraction
    num2 = int(input("Enter the numerator of the second fraction: "))
    den2 = int(input("Enter the denominator of the second fraction: "))

    print(f"{num1}/{den1} + {num2}/{den2} is equal to ", end="")
    result_num, result_den = addFraction(num1, den1, num2, den2)
    print(f"{result_num}/{result_den}")
```

Code Explanation:

1. gcd(a, b):

- This function calculates the greatest common divisor (GCD) of two numbers ‘a’ and ‘b’ using recursion.

- If ‘a’ is ‘0’, it returns ‘b’ (since the GCD of any number and 0 is the number itself).

- Otherwise, it calls itself with ‘b % a’ and ‘a’ until it finds the GCD.

2. lowest(den3, num3):

- This function simplifies a fraction.

- It calculates the GCD of the numerator (‘num3’) and denominator (‘den3’).

- It then divides both ‘num3’ and ‘den3’ by their GCD to simplify the fraction and returns the simplified numerator and denominator.

3. addFraction(num1, den1, num2, den2):

- This function adds two fractions.

- It first calculates a common denominator (‘den3’) for the two fractions using the GCD to ensure they can be combined.

- It then calculates the new numerator (‘num3’) by adjusting both fractions to the common denominator.

- Finally, it simplifies the result using the ‘lowest’ function and returns the simplified fraction.

4. Main Program Execution:

- It prompts the user to enter the numerators and denominators of two fractions.

- It calls the ‘addFraction’ function with the user-provided values to compute the sum.

- Finally, it prints the result in a readable format.

Outputs Images: