ASSIGNMENT - 1

NAME: KOTLA LAKSHMI PRIYANKA

COURSE: DATA SCIENCE AND GEN AI LLMS

HALL TICKET NO – 2406DGAL126

DATE: 29-10-2024

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 that is greater than or equal to the given
number.
    """
    if num >= 4:
        return num + (4 - (num % 4))  # Calculate the next multiple of 4
    else:
        return 4  # If the number is less than 4, return 4

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

def check_consecutive(xyz):
    """
    Checks if the numbers entered by the player are consecutive.
    Returns True if they are, otherwise False.
    """
    for i in range(1, len(xyz)):
        if xyz[i] - xyz[i - 1] != 1:  # Check if each number differs by 1
            return False
    return True  # All numbers are consecutive

def play_game():
    """
    Main function to play the number game between the user and the computer.
    """
    print("Number game between user and computer")  # Introduction to the game
    numbers_played = []  # List to keep track of all numbers played
    last_number = 0  # Variable to store the last number played

    while True:
        print("Current numbers: ", numbers_played)  # Display current numbers in
the game

        # 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:  # Validate the user's input
```

```
            print("Now enter the values:")
            player_numbers = []  # List to store the player's numbers
            for _ in range(user_input):
                number = int(input('> '))
                player_numbers.append(number)  # Add the number to the player's
list
                numbers_played.append(number)  # Add the number to the overall
game list
                last_number = number  # Update the last number played

            # Check if the player entered consecutive numbers
            if not check_consecutive(player_numbers):
                print("\nYou did not input consecutive integers.")
                lose()  # Player loses if numbers are not consecutive

            # Check if the player has reached or exceeded 20
            if last_number >= 20:
                print("\n\nCONGRATULATIONS!!! You've won!")
                break  # Player wins and exits the game

            # Computer's turn
            computer_numbers = []  # List to store the computer's numbers
            computer_pick = min(3, 20 - last_number)  # Calculate how many
numbers the computer can play
            for j in range(1, computer_pick + 1):
                computer_numbers.append(last_number + j)  # Add computer's
numbers to the list
                numbers_played.append(last_number + j)  # Update the overall
game list
            print("Computer played:", computer_numbers)  # Show the numbers the
computer played
            last_number = numbers_played[-1]  # Update the last number played

            # Check if the computer reached or exceeded 20
            if last_number >= 20:
                print("\nComputer Wins!!!")
                break  # Computer wins and exits the game

        else:
            print("Invalid input. You can only enter 1, 2, or 3 numbers.")
            lose()  # Player loses for invalid input

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

Code Breakdown:

1. nearest_multiple Function

- This function calculates the nearest multiple of 4 that is equal to or greater than the input number.

- If ‘num’ is already a multiple of 4, it returns ‘num’; otherwise, it finds the next multiple of 4 by calculating ‘num + (4 - (num % 4))’.

- This function is not used in the current game logic but might be useful if we wanted specific turn rules based on multiples of 4.

2. lose Function

- This function displays a losing message to the player and then ends the game using ‘exit(0)’.

- It's called if the player enters non-consecutive numbers or an invalid input for the number of entries (anything other than 1, 2, or 3).

3. check_consecutive Function

- Takes a list 'xyz' (the player's chosen numbers) and verifies if they are consecutive.

- Returns 'True' if all numbers in the list differ by exactly 1; otherwise, returns 'False'.

4. play_game Function

- This is the main function that orchestrates the game between the user and the computer.

- The game proceeds in a loop until one player reaches or exceeds 20.

1. Game Initialization:

- 'numbers_played' keeps a record of all numbers entered by both the player and computer.

- 'last_number' stores the last number entered, allowing the computer to continue from the player's last move.

2. Player's Turn:

- The player is prompted to enter how many numbers they'd like to play (between 1 and 3).

- If the player's input is valid, they enter that many consecutive numbers.

- 'check_consecutive' validates that the entered numbers are indeed consecutive. If they are not, the 'lose' function is called, ending the game with a loss message.

- If the player's last entered number reaches or exceeds 20, they win, and the game exits with a victory message.

3. Computer's Turn:

- The computer plays a calculated number of consecutive numbers, limited to a maximum of 3 to give the player a fair chance.

- The game checks if the computer's last entered number reaches or exceeds 20; if so, the computer wins, and the game exits with a win message for the computer.

4. Invalid Input Handling:

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

Sample 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!!!
```

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:

```python
def calculate_factorial(number):
    """Calculate the factorial of a number."""
    if number == 0 or number == 1:
        return 1  # Factorial of 0 and 1 is 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 the number of ways to choose r items from n distinct items.
    This is often written as n choose r (nCr).
    """
    # If r is out of valid range, return 0
    if r < 0 or r > n:
        return 0
    # Calculate and return nCr using the factorial formula
    return calculate_factorial(n) // (calculate_factorial(r)
calculate_factorial(n - r))

def display_pascal_triangle(number_of_rows):
    """Print Pascal's Triangle with the specified number of rows."""
    for row in range(number_of_rows):
        # Print leading spaces for formatting
        print(" "  (number_of_rows - row), end="")
        for column in range(row + 1):
            # Calculate and print nCr for the current position
            print(calculate_combinations(row, column), end=" ")
        print()  # Move to the next line after each row

# Main program execution
if __name__ == "__main__":
    # Ask the user for the number of rows for Pascal's Triangle
    rows = int(input("Enter the number of rows for Pascal's Triangle: "))
    # Display the triangle
    display_pascal_triangle(rows)
```

Code Breakdown:

1. calculate_factorial(number)

- This function calculates the factorial of a given ‘number’.
- If ‘number’ is 0 or 1, the function returns 1 since ‘0!’ and ‘1!’ are both defined as 1.
- For other values, it calculates the factorial by multiplying all integers up to ‘number’.

2. calculate_combinations(n, r)

- This function calculates the combinations, often denoted as (nCr), which is the number of ways to choose ‘r’ items from ‘n’ distinct items.

- The formula used is:

  nCr = frac{n!}{r!(n-r)!}

- It uses the ‘calculate_factorial’ function to get the factorial values for ‘n’, ‘r’, and ‘n - r’.

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

3. display_pascal_triangle(number_of_rows)

- This function prints Pascal's Triangle for a specified number of rows.

- Each row represents combinations for that row's number, so for row ‘n’, it calculates the combinations for (nC0, nC1, …., nCn).

- Spaces are added to align the triangle shape visually.

4. Main Execution

- The program prompts the user to enter the number of rows to display.

- It then calls ‘display_pascal_triangle(rows)’ to print Pascal’s Triangle.

Sample Output:

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

```
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
```

Each row corresponds to the coefficients in the expansion of ((a + b)^n) and demonstrates the pattern of Pascal’s Triangle, where each number is the sum of the two directly above it.

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 = {}  # Dictionary to store element frequencies

    # Iterate through each number in the list
    for num in numbers:
        if num in frequency:
            frequency[num] += 1  # Increment the count if the number is already
in the dictionary
        else:
            frequency[num] = 1  # Initialize count for the number

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

# Main program execution
if __name__ == "__main__":
    # Read a list of numbers from the user
    user_input = input("Enter a list of numbers separated by commas: ")
    # Convert the input string into a list of integers
    numbers = list(map(int, user_input.split(',')))

    # Call the function to count frequencies and print results
    count_frequencies(numbers)
```

Code Breakdown:

1. count_frequencies(numbers)

- This function takes a list of numbers as input and counts the frequency of each unique number in the list.

- Dictionary Usage: A dictionary named ‘frequency’ is used to store each number as a key and its frequency (count) as the value.

- Counting Frequency:

  - For each number in the list:

    - If it’s already in the dictionary, its count is incremented by 1.

    - If it’s not in the dictionary, it’s added with an initial count of 1.

- After counting, it prints the frequency of each element in the format:

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

2. Main Program Execution

- It reads a list of numbers from the user as a comma-separated string.
- The input is split by commas and converted into a list of integers.
- Then, ‘count_frequencies(numbers)’ is called to display the frequencies.

Sample 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
```

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):
    # This function reads two matrices from a file and returns them as a tuple
    matrices = {}
    current_matrix = None  # Keep track of which matrix we are reading

    with open(filename, 'r') as file:  # Open the file in read mode
        for line in file:
            line = line.strip()  # Remove any leading or trailing spaces
            if line.startswith('A='):  # Check if this line is for Matrix A
                current_matrix = 'A'  # Set current matrix to A
                continue
            elif line.startswith('B='):  # Check if this line is for Matrix B
                current_matrix = 'B'  # Set current matrix to B
                continue

            if current_matrix:
                # Convert the line of numbers into a list of integers
                row = list(map(int, line.split()))
                if current_matrix not in matrices:
                    matrices[current_matrix] = []  # Create a list for the
current matrix
                matrices[current_matrix].append(row)  # Add the row to the
current matrix

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

def add_matrices(matrix_a, matrix_b):
    # This function adds two 2x2 matrices
    result = []  # Create an empty list to hold the result
    for i in range(2):  # There are 2 rows in the matrices
        # Add the corresponding elements from both matrices
        result_row = [matrix_a[i][j] + matrix_b[i][j] for j in range(2)]
        result.append(result_row)  # Add the row to the result
    return result

def print_matrix(matrix, name):
    # This function prints the matrix in a readable format with its name
    print(f"Matrix {name}:")
    for row in matrix:
        print(' '.join(map(str, row)))  # Join the numbers in the row with
spaces
    print()  # Print a newline for better spacing

if __name__ == "__main__":
    filename = 'matrices.txt'  # Specify the filename containing the matrices
    print("Reading matrices from", filename)

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

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

    # Add the two matrices together
    result_matrix = add_matrices(matrix_a, matrix_b)

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

Code Breakdown:

1. read_matrix_from_file(filename)

- This function reads matrices (A) and (B) from a file. Each matrix is identified by lines starting with "A=" or "B=" in the file, followed by rows of numbers.

- Process:

  - It reads each line, ignoring "A=" and "B=", and assigns the following rows to the corresponding matrix.

  - Each line of numbers is converted into a list of integers, which represents a row in the matrix.

- It returns both matrices as a tuple: '(matrix A, matrix B)'.

2. add_matrices(matrix_a, matrix_b)

- This function adds the two matrices (A) and (B), assuming they are both (2 X 2) matrices.

- Process:

  - For each corresponding element in both matrices, it calculates the sum and stores it in a new matrix called 'result'.

- The result matrix is returned.

3. print_matrix(matrix, name)

- This function takes a matrix and a name, then prints the matrix with each row on a new line.

- For example, "Matrix A:" is printed before showing each row of matrix (A).

4. Main Program Execution

- Reads the matrices (A) and (B) from a file ('matrices.txt').

- Prints matrices (A) and (B).

- Adds the two matrices.

- Prints the result of the addition.

Expected File Format (‘matrices.txt’): The file should look something like this:

```
A=
1 3
4 5

B=
1 2
3 4
```

Sample 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
```

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:

```python
# 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(den3, num3):
    common_factor = gcd(num3, den3)
    den3 = den3 // common_factor  # Use integer division
    num3 = num3 // common_factor
    return num3, den3  # Return the simplified fraction

# Function to add two fractions
def addFraction(num1, den1, num2, den2):
    den3 = gcd(den1, den2)
    den3 = (den1 * den2) // den3  # Use integer division
    num3 = (num1 * (den3 // den1)) + (num2 * (den3 // den2))  # Use integer
division
    return lowest(den3, num3)  # Return the simplified fraction

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 Breakdown:

Here's an explanation of the code with step-by-step details, including an example of the output you might expect.

1. gcd(a, b)

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

- If (a) is zero, it returns (b) as the GCD.

- Otherwise, it recursively calls itself with parameters 'b % a' and 'a' to perform the Euclidean algorithm.

2. lowest(den3, num3)

- This function simplifies a fraction.

- Process:

  - It calls ‘gcd(num3, den3)’ to find the common factor of the numerator and denominator.

  - Both the numerator and denominator are divided by this common factor to get the fraction in its simplest form.

- It returns the simplified numerator and denominator as a tuple.

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

- This function adds two fractions with given numerators (‘num1’ and ‘num2’) and denominators (‘den1’ and ‘den2’).

- Process:

  - It first finds the GCD of the two denominators.

  - Then, it calculates the least common multiple (LCM) of the two denominators by dividing the product of ‘den1’ and ‘den2’ by their GCD, which gives the common denominator for the addition.

  - Each fraction is converted to this common denominator, and the resulting numerators are added together.

- The sum (numerator and denominator) is then simplified by calling ‘lowest(den3, num3)’.

- It returns the simplified fraction.

4. Main Program Execution

- Input the numerators and denominators for two fractions.

- Prints the result of adding the two fractions.

Sample Output:

For example, let's say the input is:

```
Enter the numerator of the first fraction: 1
Enter the denominator of the first fraction: 2
Enter the numerator of the second fraction: 1
Enter the denominator of the second fraction: 3
```

Output is:

```
1/2 + 1/3 is equal to 5/6
```