**Course: DSGALLM**

**Name of Student: K.Raghavendra**

**Roll.no : 2406DGAL111**

**Assignment Answers**

**1Ans.**

```
import random
def get_user_input(max_number):
    while True:
        user_input = input(f"Enter 1 to 3 digits starting from {max_number + 1}: ")
        digits = list(map(int, user_input.split()))
        if all(d == max_number + i + 1 for i, d in enumerate(digits)) and len(digits) in {1, 2, 3}:
            return digits
        else:
            print("Invalid input. Please enter a valid sequence.")
def get_computer_input(max_number):
    digits_count = random.randint(1, 3)
    return list(range(max_number + 1, max_number + digits_count + 1))
def play_game():
    user_max = 0
    computer_max = 0
    while user_max < 20 and computer_max < 20:
        # User's turn
        user_digits = get_user_input(user_max)
        user_max += len(user_digits)
        print(f"You played: {user_digits}, your total is now: {user_max}")
        if user_max >= 20:
            print("Congratulations! You win!")
```

```
            break
        # Computer's turn
        computer_digits = get_computer_input(user_max)
        computer_max += len(computer_digits)
        print(f"Computer played: {computer_digits}, computer's total is now: {computer_max}")
        if computer_max >= 20:
            print("Computer wins! Better luck next time!")
            break
if __name__ == "__main__":
    play_game()
```

**2Ans:**

```
def ncr(n, r):
    if r > n or r < 0:
        return 0
    # Calculate n! / (r! * (n - r)!)
    numerator = 1
    denominator = 1
    for i in range(1, r + 1):
        numerator *= (n - i + 1)
        denominator *= i
    return numerator // denominator

def print_pascals_triangle(rows):
    for i in range(rows):
        # Print leading spaces for alignment
        print(' ' * (rows - i), end='')
```

```
        for j in range(i + 1):
            # Calculate the value using ncr
            value = ncr(i, j)
            print(value, end=' ')

        print()  # New line after each row

if __name__ == "__main__":
    num_rows = int(input("Enter the number of rows for Pascal's Triangle: "))
    print_pascals_triangle(num_rows)
```

**3Ans:**

```
def count_frequencies(numbers):
    frequency_dict = {}
    for number in numbers:
        if number in frequency_dict:
            frequency_dict[number] += 1
        else:
            frequency_dict[number] = 1

    return frequency_dict
def print_repeated_elements(frequency_dict):
    print("Repeated elements with their frequency count:")
    for number, count in frequency_dict.items():
        if count > 1:
            print(f"Number: {number}, Frequency: {count}")
```

```
if __name__ == "__main__":
    # Read input from user
    input_numbers = input("Enter a list of numbers separated by spaces: ")
    numbers = list(map(int, input_numbers.split()))
    # Count frequencies
    frequency_dict = count_frequencies(numbers)
    # Print repeated elements
    print_repeated_elements(frequency_dict)
```

**4Ans:**

```
def read_matrices_from_file(filename):
    with open(filename, 'r') as file:
        lines = file.readlines()

    # Parse the matrices from the lines
    A = []
    B = []
    for i in range(2):  # First 2 lines for Matrix A
        A.append(list(map(int, lines[i].strip().split())))

    for i in range(2, 4):  # Next 2 lines for Matrix B
        B.append(list(map(int, lines[i].strip().split())))

    return A, B

def add_matrices(A, B):
    # Initialize result matrix
    result = [[0, 0], [0, 0]]
```

```
    for i in range(2):
        for j in range(2):
            result[i][j] = A[i][j] + B[i][j]

    return result

def print_matrix(matrix):
    for row in matrix:
        print(' '.join(map(str, row)))

if __name__ == "__main__":
    filename = 'matrices.txt'  # The name of the file containing the matrices
    A, B = read_matrices_from_file(filename)

    print("Matrix A:")
    print_matrix(A)

    print("\nMatrix B:")
    print_matrix(B)

    result = add_matrices(A, B)
    print("\nResult of A + B:")
    print_matrix(result)
```

**5Ans:**

```
from math import gcd
```

```
class Fraction:
    def __init__(self, numerator, denominator):
        if denominator == 0:
            raise ValueError("Denominator cannot be zero.")
        self.numerator = numerator
        self.denominator = denominator
        self._reduce()

    def _reduce(self):
        # Reduce the fraction by dividing both numerator and denominator by their GCD
        common_divisor = gcd(self.numerator, self.denominator)
        self.numerator //= common_divisor
        self.denominator //= common_divisor

    def __add__(self, other):
        if not isinstance(other, Fraction):
            return NotImplemented
        new_numerator = (self.numerator * other.denominator) + (other.numerator * self.denominator)
        new_denominator = self.denominator * other.denominator
        return Fraction(new_numerator, new_denominator)
    def __str__(self):
        return f"{self.numerator}/{self.denominator}"

# Example usage
if __name__ == "__main__":
    fraction1 = Fraction(1, 2)  # Represents 1/2
    fraction2 = Fraction(1, 3)  # Represents 1/3
```

```
result = fraction1 + fraction2  # Using overloaded + operator
print(f"{fraction1} + {fraction2} = {result}")
```