Problem 1

```
def calculate_final_grade(coursework_marks, coursework_weights, extenuating_circumstance, drop_lowest_mark):
    # Adjust coursework weights if extenuating circumstance is provided
    if extenuating_circumstance is not None:
        ec_index = extenuating_circumstance - 1
        x = coursework_weights[ec_index] / (len(coursework_weights) - 1)
        del coursework_marks[ec_index]  # Remove mark associated with extenuating circumstance
        del coursework_weights[ec_index]  # Remove weight associated with extenuating circumstance
        for i in range(len(coursework_weights)):
            coursework_weights[i] += x  # Redistribute weights evenly among remaining coursework

    # Drop lowest mark if specified
    if drop_lowest_mark:
        lm_index = coursework_marks.index(min(coursework_marks))
        x = coursework_weights[lm_index] / (len(coursework_weights) - 1)
        del coursework_marks[lm_index]  # Remove lowest mark
        del coursework_weights[lm_index]  # Remove weight associated with lowest mark
        for i in range(len(coursework_weights)):
            coursework_weights[i] += x  # Redistribute weights evenly among remaining coursework

    # Calculate final mark based on weighted sum of coursework marks
    mark = sum((coursework_marks[i] * coursework_weights[i]) / 100 for i in range(len(coursework_marks))) * 100
    final_mark = round(mark, 1)  # Round final mark to one decimal place
    return final_mark

def calculate_grade(final_mark):
    # Determine grade based on final mark
    if final_mark >= 90:
        return "A*"
    elif 80 <= final_mark < 90:
        return "A"
    elif 70 <= final_mark < 80:
        return "B"
    elif 60 <= final_mark < 70:
        return "C"
    elif 50 <= final_mark < 60:
        return "D"
    else:
        return "F"
```

```
# Get module name from user input
module_name = input("Enter the module name: ")

# Get number of students and create a list of student names
num_students = int(input("Enter the number of students: "))
students = [f"Student {i+1}" for i in range(num_students)]

# Get number of courseworks and create a list of coursework names
num_courseworks = int(input("Enter the number of courseworks in module: "))
courseworks = [f"Coursework {i+1}" for i in range(num_courseworks)]
weightings = []

# Get weightings for each coursework
for coursework in courseworks:
    while True:
        x = float(input(f"Enter the weighting for {coursework}: "))
        if 0 <= x <= 1:
            weightings.append(x)
            break
        else:
            print("Error. Please enter a value between 0 and 1")

# Check if total weightings add up to 1
if sum(weightings) != 1:
    print("Error. The coursework weightings do not add up to 1")

# Iterate over each student
for student in range(1, num_students + 1):
    print(f"\nStudent {student}:")
    coursework_marks = []
    # Get marks for each coursework
    for coursework in courseworks:
        while True:
            mark = float(input(f"Enter the mark out of 100 for {coursework}: "))
            if 0 <= mark <= 100:
                coursework_marks.append(mark)
                break
            else:
                print("Error. Please enter a value between 0 and 100")

    # Check for extenuating circumstances or dropping lowest mark
    extenuating_circumstance = None
    drop_lowest_mark = False
    ec_choice = input("Does the student have extenuating circumstances for any
coursework? (Y/N): ")
```

```
    if ec_choice.lower() == 'y':
        extenuating_circumstance = int(input("Enter the coursework number with
extenuating circumstances: "))
    else:
        drop_choice = input("Would you like to drop the lowest mark? (Y/N): ")
        if drop_choice.lower() == 'y':
            drop_lowest_mark = True

    # Calculate final grade and determine grade
    final_mark = calculate_final_grade(coursework_marks.copy(), weightings.copy(),
extenuating_circumstance, drop_lowest_mark)
    grade = calculate_grade(final_mark)
    print(f"Final Mark for {module_name}: {final_mark}")
    print(f"Grade: {grade}")
```

Problem 2

```
import random

# Function to prompt a player to make a guess for the sum of the dice
def make_guess(player_name):
    return int(input(f"{player_name}, make your guess: "))

# Function to print the graphical representation of a dice face based on its value
def print_diceface(value):
    faces = {
        1: ["     ",
            "  .  ",
            "     "],
        2: [".    ",
            "     ",
            "    ."],
        3: [".    ",
            "  .  ",
            "    ."],
        4: [".   .",
            "     ",
            ".   ."],
        5: [".   .",
            "  .  ",
            ".   ."],
        6: [".   .",
            ".   .",
            ".   ."]
    }

    for face in faces[value]:
        print(face)

# Main code where the game logic is implemented

print("The aim of this game is to be the best at guessing the sums of three dice throws... You all have 5 lives each, and whoever lasts till the end is the winner!", end="\n\n")

# Generate a list of player names 1 through 6
players = [f"Player {i+1}" for i in range(6)]
# Initialize each player's remaining guesses to 5
guesses = {player: 5 for player in players}

# Main game loop, continues until only 1 or 0 players left
while len(players)> 1:
    round_guesses = {}
    # Loop through each player to get their guesses for the current round
    for player in players:
        if guesses[player] > 0:
            guess = make_guess(player)
            round_guesses[player] = guess

    # Roll three dice and calculate their sum
    value_1 = random.randint(1, 6)
    value_2 = random.randint(1, 6)
    value_3 = random.randint(1, 6)

    roll_result= value_1 + value_2 + value_3

    # Print the graphical representation of each dice face
    print_diceface(value_1)
    print()
    print_diceface(value_2)
    print()
    print_diceface(value_3)
    print()

    print(f"The sum of the dice is: {roll_result}")

    # Check each player's guess against the roll result and update their guesses accordingly
    for player, guess in round_guesses.items():
        if guess == roll_result:
            print(f"{player} guessed correctly!")
            print()
        else:
            guesses[player] -= 1
            print(f"{player}'s guess was incorrect. You have {guesses[player]} guesses left.")
            print()

    # Remove players who have run out of guesses from the active player list
    players = [player for player in players if guesses[player] > 0]

    # Declare the winner or indicate if there's no winner
if len(players) == 1:
        print(f"{players[0]} is the winner!")
else:
        print("No winner. All players are out of guesses.")
```

Sara Boumrah

Problem 3

```
def count(value, data):
    # Initialize count to 0
    count = 0
    # Iterate over each element in data
    for num in data:
        # If element matches the value, increment count
        if num == value:
            count += 1
    return count

def search(value, data):
    # Iterate over each index and element pair in data
    for i, num in enumerate(data):
        # If element matches the value, return its index
        if num == value:
            return i
    # If value not found, return -1
    return -1

def sort(data):
    n = len(data)  # Get the length of the data list
    for i in range(n):  # Iterate over each element of the list
        for j in range(0, n-i-1):  # Iterate over each element except the last i elements
            if data[j] > data[j+1]:  # If current element is greater than the next one
                # Swap the elements to sort them in ascending order
                data[j], data[j+1] = data[j+1], data[j]

def delete(value, data):
    i = 0
    # Iterate over data
    while i < len(data):
        # If element matches value, remove it
        if data[i] == value:
            data.pop(i)
        else:
            i += 1

# Ask the user to input some integers
data = []
num_inputs = int(input("Enter the number of integers you want to input: "))
# Gather user input for specified number of integers
for _ in range(num_inputs):
    num = int(input("Enter an integer: "))
    data.append(num)

# Ask the user for the desired operation
while True:
    print("\nChoose one of the following options:")
    print("1. Count")
    print("2. Search")
    print("3. Sort")
    print("4. Delete")
    print("5. Exit")
    # Prompt user for choice
    choice = int(input("Enter your choice (1-5): "))

    if choice == 1:
        # Get value from user and count occurrences
        value = int(input("Enter the value to count: "))
        print("Number of elements with value", value, ":", count(value, data))
    elif choice == 2:
        # Get value from user and search for it
        value = int(input("Enter the value to search: "))
        index = search(value, data)
        # Display result of search
        if index != -1:
            print("Value", value, "found at index", index)
        else:
            print("Value", value, "not found in the list.")
    elif choice == 3:
        # Sort the list
        sort(data)
        print("Sorted list:", data)
    elif choice == 4:
        # Get value from user and delete all occurrences
        value = int(input("Enter the value to delete: "))
        delete(value, data)
        print("List after deleting all occurrences of", value, ":", data)
    elif choice == 5:
        # Exit the program
        print("Exiting program.")
        break
    else:
        # Handle invalid choice
        print("Invalid choice. Please enter a number between 1 and 5.")
```

Problem 4

```
import random

# Function to print the game board with player positions
def print_board(P, Q):
    # Initialize the game board with empty cells
    board = [["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"],
             ["|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|", "|___|"]
             ]

    # Place player P's token on the board
    if P == Q:
        board[P[0]][P[1]] = "|P_Q|"
    else:
        board[P[0]][P[1]] = "|_P_|"
        board[Q[0]][Q[1]] = "|_Q_|"

    # Print the board
    for row in board:
        print("".join(row))

# Function to print the face of a dice given its value
def print_diceface(value):
    # Mapping of dice values to their corresponding faces
    faces = {
        1: ["   ",
            " . ",
            "   "],
        2: [".  ",
            "   ",
            "  ."],
        3: [".  ",
            " . ",
            "  ."],
        4: [". .",
            "   ",
            ". ."],
        5: [". .",
            " . ",
            ". ."],
        6: [". .",
            ". .",
            ". ."]
    }

    # Print the face of the dice with the given value
    for face in faces[value]:
        print(face)


# Function to handle player's move
def player_move(player_turn, dice_value, player_position):
    if player_turn:
        # Print the face of the rolled dice
        print_diceface(dice_value)
        # Check if player is at the starting row and within the board range
        if player_position[0] == 0 and player_position[1] <= 5:
            # If the dice value is greater than the remaining steps to the finish line
            if dice_value > player_position[1]:
                print(f"Invalid move - need dice value of less than or equal to {player_position[1]}")
            else:
                # Move the player
                player_position[1] -= dice_value
        else:
            # Movement for other areas of the board
            while dice_value > 0:
                if player_position[0] % 2 == 0:  # Even row
                    if player_position[1] == 0:
                        player_position[0] -= 1
                        dice_value -= 1
                    elif 1 <= player_position[1] <= 7:
                        player_position[1] -= 1
                        dice_value -= 1
                    else:
                        print("Error")
                elif player_position[0] % 2 == 1:  # Odd row
                    if player_position[1] == 7:
                        player_position[0] -= 1
                        dice_value -= 1
                    elif 0 <= player_position[1] <= 6:
                        player_position[1] += 1
                        dice_value -= 1
                    else:
                        print("Error")
        # Switch player's turn
        return not player_turn
    else:
        return player_turn


# Function to check if a player has reached the finish line
def check_for_winner(player_position, current_player):
    if player_position == [0, 0]:
        print()
        print(f"Player {current_player} is the winner! ")
        return True

# Initialize players' positions
P_position = [7, 0]
Q_position = [7, 0]

# Print game instructions and initial board
print("This is a two player game. Each round the player must roll a dice to determine the moves that player makes per round. Whoever gets to the finish line first is the winner!", end="\n\n")
print_board(P_position, Q_position)

GameOver = False



# Main game loop
while not GameOver:
    P_turn = True
    Q_turn = True
    # Player P's turn
    while P_turn:
        # Roll the dice for Player P
        P_dice = random.randint(1, 6)
        input("Player P's turn. Press enter to roll the dice")
        # Perform Player P's move
        P_turn = player_move(P_turn, P_dice, P_position)
        # Print the updated board
        print_board(P_position, Q_position)
        # Check for a winner after Player P's turn
        if check_for_winner(P_position, "P"):
            GameOver = True
            break
    #if Player P wins, the main game loop is exited before entering player Q's loop.
    if GameOver ==True:
        break

    # Player Q's turn
    while Q_turn:
        # Roll the dice for Player Q
        Q_dice = random.randint(1, 6)
        input("Player Q's turn. Press enter to roll the dice")
        # Perform Player Q's move
        Q_turn = player_move(Q_turn, Q_dice, Q_position)
        # Print the updated board
        print_board(P_position, Q_position)
        # Check for a winner after Player Q's turn
        if check_for_winner(Q_position, "Q"):
            GameOver = True
            break
```

Problem 5

```
N = int(input("Enter the amount of numbers in the Stern-Brocot sequence you want to generate: "))

# Starting point of the sequence
stern_brocot = [1,1]

# Variable to keep track of the index in the sequence
i= 0

# Generating the sequence until it has N numbers
while len(stern_brocot) < N:
    a = stern_brocot[i]
    b = stern_brocot[i + 1]
    c = a + b # Calculate the sum of the current and next number
    stern_brocot.append(c)
    stern_brocot.append(b)
    i +=1

# Trimming the sequence to have only the desired amount of numbers
stern_brocot= stern_brocot[:N]

# Copying the sequence for further processing
sb_copy = stern_brocot.copy()

# Removing the last number if the sequence length is odd
if len(sb_copy)%2 != 0:
    sb_copy.pop(-1)

r_list = []
i= 0

# Converting the numbers into fractions
while i < N-1:
    a= sb_copy[i]
    b= sb_copy[i + 1]
    c= f"{a}/{b}" # Create a string representation of the fraction
    r_list.append(c) # Add the fraction to the list
    i +=1

if len(stern_brocot) == 1:
    print(f"The sequence is: {stern_brocot}")
    print("There are no rational numbers for this sequence.")
else:
    print(f"The sequence is: {stern_brocot}")
    print(f"The rational numbers are: {r_list}")
```

Problem 6

```
# Ask the user to input the coefficients of the quintic equation
print("input the coefficients of your quintic equation: ")
a= int(input("a= "))
b= int(input("b= "))
c= int(input("c= "))
d= int(input("d= "))
e= int(input("e= "))
f= int(input("f= "))

# Ask the user to input an interval
print("input an interval [l,h] where f(l) and f(h) yield opposing signs: ")
l= int(input("l= "))
h= int(input("h= "))

soln= False
NMAX = 5000
TOL = 0.000001

# Loop through a certain number of iterations, specified by NMAX
for N in range(1, NMAX, 1):
   # Calculate the midpoint of the interval
   x= (l+h)/2

   # Calculate the value of the equation at the midpoint
   fx= a*(pow(x,5)) + b*(pow(x,4)) + c*(pow(x,3)) + d*(pow(x,2)) + e*x + f

   # Calculate the value of the equation at the lower end of the interval
   fl= a*(pow(l,5)) + b*(pow(l,4)) + c*(pow(l,3)) + d*(pow(l,2)) + e*l + f

   # Check if the value of the equation is zero or if the interval is very small
   if fx == 0 or (h-l)/2 < TOL:
      soln = True
      break

   # Check if the value of the equation at the midpoint and lower end have the same sign
   if fx*fl > 0:
      # If they have the same sign, update the lower end of the interval to the midpoint
      l= x
   else:
      # Otherwise, update the upper end of the interval to the midpoint
      h= x

if soln:
   # If a solution is found, print the value of the solution
   print(f"x= {x}")
else:
   print("Method failed.")
```

Problem 7

```
hidden_list_a = [0,0,0,0,0,0,0,0]
hidden_list_b = [0,0,0,0,0,0,0,0]

# setting the values for list a and b which the user will slowly reveal
list_a = [6,8,2,1,9,3,5,7]
list_b = [5,3,1,9,7,6,8,2]

print(hidden_list_a)
print(hidden_list_b)


while hidden_list_a.count(0) > 0 and hidden_list_b.count(0) > 0:
   choice1 = int(input("Which position do you want to check in the first row?: ")) - 1
   chosen_num1 = list_a[choice1]

   a= hidden_list_a.copy()
   a.pop(choice1)
   a.insert(choice1, chosen_num1)


   print(a)
   print(hidden_list_b)

   choice2 = int(input("Now guess where that number is in the second row: ")) -1

   chosen_num2 = list_b[choice2]
   b= hidden_list_b.copy()
   b.pop(choice2)
   b.insert(choice2, chosen_num2)
   print(a)
   print(b)

   if chosen_num1 == chosen_num2:

      hidden_list_a = a
      hidden_list_b = b
   else:
      print("Try again!")
      print(hidden_list_a)
      print(hidden_list_b)
      continue
```