Coding Assessment

Problem 1 - Split the bill

It's tricky keeping track of who is owed what when spending money in a group. Write a function to balance the books.

- The function should take one parameter: an object/dict with two or more name-value pairs which represent the members of the group and the amount spent by each.
- The function should return an object/dict with the same names, showing how much money the members should pay or receive.

Further points:

- The values should be positive numbers if the person should receive money from the group, negative numbers if they owe money to the group.
- If value is a decimal, round to two decimal places.

Example

3 friends go out together: A spends $20, B spends $15, and C spends $10. The function should return an object/dict showing that A should receive $5, B should receive $0, and C should pay $5.

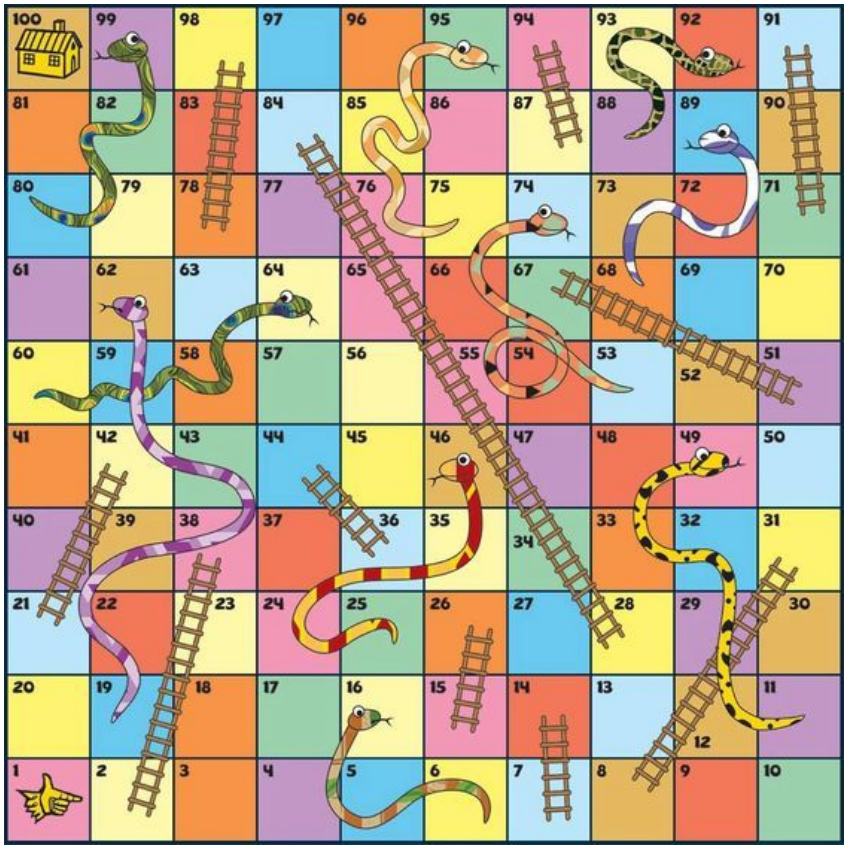
```
var group = {
    A: 20,
    B: 15,
    C: 10
}


splitTheBill(group) // returns {A: 5, B: 0, C: -5}
```

Problem 2 - Snakes and ladders

Make a simple class called **SnakesLadders**. The test cases will call the method **play(die1, die2)** independently of the state of the game or the player turn. The variables **die1** and **die2** are the die

thrown in a turn and are both integers between 1 and 6. The player will move the sum of die1 and die2.

The Board



Rules

1. There are two players and both start off the board on square 0.

2. Player 1 starts and alternates with player 2.

3. You follow the numbers up the board in order 1=>100

4. If the value of both die are the same then that player will have another go.

5. Climb up ladders. The ladders on the game board allow you to move upwards and get ahead faster. If you land exactly on a square that shows an image of the bottom of a ladder, then you may move the player all the way up to the square at the top of the ladder. (even if you roll a double).

6. Slide down snakes. Snakes move you back on the board because you have to slide down them. If you land exactly at the top of a snake, slide move the player all the way to the square at the bottom of the snake or chute. (even if you roll a double).

7. Land exactly on the last square to win. The first person to reach the highest square on the board wins. But there's a twist! If you roll too high, your player "bounces" off the last square and moves back. You can only win by rolling the exact number needed to land on the last square. For example, if you are on square 98 and roll a five, move your game piece to 100 (two moves), then "bounce" back to 99, 98, 97 (three, four then five moves.)

Returns

Return **Player n Wins!**. Where **n** is winning player that has landed on square 100 without any remaining moves left.

Return **Game over!** if a player has won and another player tries to play.

Otherwise return **Player n is on square x.** Where n is the current player and x is the square they are currently on.

Problem 3 - Seats in Theater

Task

Your friend advised you to see a new performance in the most popular theater in the city. He knows a lot about art and his advice is usually good, but not this time: the performance turned out to be awfully dull. It's so bad you want to sneak out, which is quite simple, especially since the exit is located right behind your row to the left. All you need to do is climb over your seat and make your way to the exit.

The main problem is your shyness: you're afraid that you'll end up blocking the view (even if only for a couple of seconds) of all the people who sit behind you and in your column or the columns to your left. To gain some courage, you decide to calculate the number of such people and see if you can possibly make it to the exit without disturbing too many people.

Given the total number of rows and columns in the theater (nRows and nCols, respectively), and the row and column you're sitting in, return the number of people who sit strictly behind you and in your column or to the left, assuming all seats are occupied.

Example:

For nCols = 16, nRows = 11, col = 5 and row = 3, the output should be

The theater looks like this:

Input/Output

- `[input]` integer `nCols`

  An integer, the number of theater's columns.

  Constraints: 1 ≤ nCols ≤ 1000.

- `[input]` integer `nRows`

  An integer, the number of theater's rows.

  Constraints: 1 ≤ nRows ≤ 1000.

- `[input]` integer `col`

  An integer, the column number of your own seat (1-based).

  Constraints: 1 ≤ col ≤ nCols.

- `[input]` integer `row`

  An integer, the row number of your own seat (1-based).

  Constraints: 1 ≤ row ≤ nRows.

- `[output]` an integer

The number of people who sit strictly behind you and in your column or to the left.