Assignment P5

Due Date: April 19

Purpose
Spring is the best time write programs. Who needs sunshine and warmth? And you now have sufficient Python knowledge that you can finally play a real game! To do this, you will write several functions that use pass-by-value parameters and return statements. You will also have to learn a little about a casino game.

Problem
The game of roulette is one of the easier games of chance to figure out. Trouble is, if you really play roulette at a casino, you may find your money leaving your wallet or purse very quickly. Instead, why not play virtual roulette? That way, you will lose only your virtual money. With this in mind, GousieSoft has hired you to write just such a game.

Basic Rules of European Roulette
The game of roulette (European version) consists of a spinning wheel with a ball that can land in one of 36 different slots representing red or black numbers plus one green slot for 0 (the American version adds another green slot with 00, which gives the house more of an advantage). The cloth-covered table, or “layout,” contains all of the betting areas, which consist of all the numbers plus labels for various other options. See Figure 1 for an image of a wheel and a layout (taken from www.777.com).

![Figure 1: European roulette layout and wheel.](image)

The idea of the game is to place a bet on a particular space on the layout, in the hope that the ball will land on that number and/or color on the roulette wheel. If the ball does land on the same number and/or color, the player wins, the amount depending on how the bet was placed. Go to [http://freecasinogamesdoc.com/european-roulette.htm](http://freecasinogamesdoc.com/european-roulette.htm) to try out roulette for free; pay particular attention to what happens with your current balance at each turn.

If you understand roulette, then you know there are many ways to bet (wager), with the corresponding number of ways to win. The simplest method is to place your wager on one number; if the ball lands on that number, then you win 36 times your bet (see odds, below). Another easy way to bet is to wager on red or black. If the ball lands on any number with your color, you double your wager. It then gets more complicated. For example, you may split your wager among as many numbers as you wish. Or you can put one wager on two, three, or four numbers at once. In this case, the winnings are dependent on how many numbers on which you wagered. You can also bet
on an entire row of numbers; if the ball lands on any, you win an amount dependent on the number of values in the row. However, we are going to implement a simpler version of European roulette.

**Modified European roulette rules for this project**

To make this project a little easier, you will implement some modified roulette rules:

- The wheel is a standard European roulette wheel, as shown in Figure 1. There are 36 numbers in red and black, and one green representing 0.
- The player may wager in increments of $5.00; thus, each chip is worth $5.
- A bet can be any number of chips up to the total amount the player has.
- The player may place only one or two bets at a time:
  - a wager may be on one number **only**, including 0.
  - a wager may be on red or black.
  - a wager may be on one row.

See below for a sample program run.

- If the ball lands on the chosen number, then the player wins 36 times the bet (or 35 times the bet PLUS the original bet; the payout odds are said to be 35:1).
- If the ball lands on a number in a chosen row, then the player wins 3 times the bet (or 2 times the bet PLUS the original bet; payout odds are 2:1).
- If the ball lands on the chosen color, then the player wins 2 times the bet (payout odds are 1:1, or “even”).

**Input**

Before the game begins, the player will input the starting bankroll, which will be a multiple of 5. Play then begins. Before each spin of the wheel, the player places one or two bets, each evenly divisible by 5 (you do not have to do error checking for this). The bet should be subtracted from the bankroll. An input of 0 for the first bet or a bankroll less than $5 indicates the game is over.

After the bet, the program should prompt for the number or color on which to place the wager. Valid input is 0-36 representing the numbers on the layout, **B** or **R** for black or red, respectively, and **X**, **Y**, and **Z** for the three rows, where **X** represents the row starting at 1. This means that your input has to handle numbers and characters (see example below).

**Output**

After the player places his/her wager(s), the program should show the **number and color** of the slot where the ball landed on the wheel. The program should then determine and display if the player won or lost, as well as the updated bankroll.

---

1This is \((1 - p)/p\), for you statisticians.
Sample Game

Bankroll: $120.00

Bet 1 amount : 10
Number to bet on: 29
Bet 2 amount : 10
Number to bet on: 31

RESULT: 23 Red - NO MATCH

Bankroll: $100.00

Bet 1 amount : 10
Number to bet on: B
Bet 2 amount : 0

RESULT: 8 Black - WIN!

Bankroll: $110.00

Bet 1 amount : 0

Final bankroll: $110.00. Thanks for playing!

Specifics

- You must design the program so that main() is only a “manager;” that is, it should call functions to do all the work. To this end, no input statements are allowed in main().

- Use only pass-by-value parameters for this project.

- You must have a function called wager() that gets the bet and updates the bankroll. It should have only one parameter: the bankroll.

- You must have a function called getSpin() that spins the roulette wheel and returns both the number and color. Use the Python function randrange() (see text) to find a random number.

- You must have a function called checkWin() that determines if the player won and that updates the bankroll. You must determine the parameters for this, as well as what to return.

This one is a bit involved, because the player can place a bet on a number, on red/black, or a row. So the function must accept either a number or a character, and you don’t know which is coming (huh??). Once you get the input, the bet could be converted to a number as in the following chart:
<table>
<thead>
<tr>
<th>Input</th>
<th>What Is It?</th>
<th>Numeric Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-36</td>
<td>0-36</td>
<td>0-36</td>
</tr>
<tr>
<td>B</td>
<td>Black</td>
<td>37</td>
</tr>
<tr>
<td>R</td>
<td>Red</td>
<td>38</td>
</tr>
<tr>
<td>X</td>
<td>Row 1</td>
<td>39</td>
</tr>
<tr>
<td>Y</td>
<td>Row 2</td>
<td>40</td>
</tr>
<tr>
<td>Z</td>
<td>Row 3</td>
<td>41</td>
</tr>
</tbody>
</table>

These numbers will make it easier to determine if you have a winning bet. *Note: you do not have to use this method! There are other good solutions.*

Checking if a number matches is very easy. Checking if the color matches is a bit arduous, but also easy. Checking if a row matches is easy if you use Python’s `in` functionality (see text).

- Other function(s) are up to you. Be sure to use return types and/or parameters correctly. This means using good programming practice, not just “making it work” in Python.

- All functions should have a descriptive comment, including pre- and post-conditions.

- If you find yourself writing the same (or essentially the same) code repeatedly, that indicates you need another function which you can then call repeatedly, using different arguments if need be.

- Do **not** use any global variables.

**Notes**

Once again, I urge you to start small and work your way to the final solution. For example, you might start by writing a very short `main()` and then adding in one function at a time. I suggest working on the input first, since if the input doesn’t work, then nothing else will, either. Be certain that a function works properly before going on to the next. In any case, you must have some sort of plan (algorithm) before trying to code any part of the problem!

For full credit, the game should work for two bets at a time. However, you can get a good score if the game only takes one bet at a time. This is a bit easier to implement. In any case, you might want to start with a version that handles only one bet at a time, and then add in the second bet if you have additional time to work on the project.

While you are coding and debugging, call the `print()` function liberally so that you know that your values are correct up to that point. You can always delete extra statements later.

As usual, email your source code to me, using the usual naming conventions. Hand in hard copy of your source code in the envelope on my door on April 20th at the latest.

*If you think a little bit harder, you’ll understand.*

– Professor Moorthy (a grad school professor of mine)