MTH 3300
Fall 2022

Programming Problem 6: montecarlosimulation.py

Directions: Download the template file provided on Blackboard. Then open Spyder, load these template files, and write the following program. Submit your source code via Gradescope, in `.py` format; do NOT send any other files. READ THE INSTRUCTIONS on submitting your work in the Course Documents section of Blackboard.

Be sure to read the SPECIFICATIONS carefully! And write comments!

You are playing a game where you roll a fair 6-sided die, and based on the outcome of your roll: you get \$1 (if you roll a 1 or a 2), \$5 (if you roll a 3), \$10 (if you roll a 4), \$20 (if you roll a 5), or \$100 (if you roll a 6). You can roll the die n times, and if the sum of your earnings is at least $\$x$ (that is, *g*reater than *OR equal to* $\$x$), you win and can keep your money! But if you lose, you don't get to keep the cash AND have to pay \$100 for playing. (That is, your earnings end up being *negative* 100 dollars.)

For example, if $n = 2$ and $x = 10$, a single play of the game consists of rolling the die twice. Suppose we roll a 1 and then a 6. This means we have total earnings of \$1 + \$100 = \$101 for the play, and since this is greater than or equal to $x = 10$, we win and get to keep the money. On the other hand, (with the same values of n and x), if we roll a 1 and then a 3, we lose since \$1 + \$5 = \$6, which is not greater than or equal to \$10. In this case, our total earnings are −\$100 for the single play.

(For the program, n and x will be inputs when you run your code on Gradescope.)

Your job: using a simulation, estimate

1. the probability that you win (meaning your total earnings sum to at least $\$x$), and
2. if you play this game repeatedly, the average amount you will win (where the amount you win in a game could be negative).

Here is a bit more direction: you should simulate 1,000,000 plays of this game. Each simulated play involves rolling a six-sided die n times. The die is equally like to land on any of its six sides. Your earnings are the sum of the amounts you earn for each roll, provided you make at least $\$x$ total (\$1 if you roll a 1 or a 2, \$5 if you roll a 3, \$10 if you roll a 4, \$20 if you roll a 5, or \$100 if you roll a 6); these should be summed each time you play. When you do not make at least $\$x$ with your n rolls, your earnings for that play are −100 dollars.

As you proceed through these 1,000,000 simulated plays, you should keep track of the number of simulations where you win: the probability of winning should be approximately equal to $\frac{\text{number of winning simulations}}{\text{total number of simulated games}}$. You should also keep track of the average amount of money (in dollars) you've won per game (again, losing a \$100 counts as "winning" −100 dollars).

After completing the simulations, print out your program's estimates for both. (This is set up for you already in the provided template file and specified to print to the desired number of decimal places.)

Do not change anything provided in the template, and insert your code in the provided area (between the hashtag lines). You should use the variables provided in your code.

Specifications: your program must

- NOT ask the user for any additional input. (Ignoring the input lines already provided in the template.)
- print out an estimate of the probability that n rolls of the die result in earnings (as specified according to the rolls above) that sum to a value greater than or equal to $\$x$ (represented by the variable `dollars` in the template file).
- also print out an estimate of the average amount of winnings per game, where winnings for a single trial are calculated by either the sum of the earnings or −100, the latter when the sum of the n values is less than $\$x$.
- **use 1,000,000 simulations with random numbers – no credit for theoretical solutions**. Your answers should **not** be exactly the true probability/average. (And if you view the answer to more decimal places, you should **not** even get the same answers each time you run it.) If this point isn't obvious, please ask for clarification!

This problem has three invisible and five visible test cases on Gradescope. Each test case is worth one point, and two points will be manually assigned for following all specifications. Additional points may be deducted for incorrect code or code not following specifications, even if test cases pass. (Due to the nature of randomness involved in this program, I had to crudely round the final outputs, which may result in false positive on some test cases, so your code will be reviewed manually for correctness.)

Negative 100 points will be assigned to submissions that fail to fill out the collaborators/resources section.