Lab 8 – Fun and Games

The Way of the Program - Fall 2023

---

1. Download **lab8_startcode.py** from our class web page (under Labs) and open it in IDLE. The **dicetest()** program uses a for-loop to simulate repeatedly throwing two independent dice. Run it a few times with 20 trials to see what it does.

   Next, modify dicetest so that it counts and reports the number of times each possible *dice-sum* value ranging from 2 to 12 occurs. It should also print out the corresponding percentages for each dice-sum. You can use the included **randomtest()** program (which we wrote in class) as a guide in modifying your dicetest program. For instance, dicetest should use a list to keep track of the tally of counts obtained for each of the possible dice-sum values. A simple approach is to use a list initialized to *thirteen* zeros, instead of ten zeros as in randomtest, and then just ignore the first two positions in the list (which correspond to dice-sum values of 0 and 1, which will never arise). How likely is rolling a 7, compared to rolling a 2 or a 12?

2. Using a while-loop, write a program called **roll()** that simulates repeatedly rolling a single die until a six is obtained. Your program should print out the number of rolls that were required. You should call `random.randrange(1, 7)` to simulate the throw of a single die. For example, a typical run of the program might look like this:

```
>>> roll()
You rolled a 3
You rolled a 5
You rolled a 5
You rolled a 6
It took 4 rolls to get a six
```

3. Using a while-loop, write a program called **doubles()** that simulates repeatedly rolling *two* dice until both come up with the same number, using a while-loop. In addition to printing out the dice values, if the program rolls double-ones it should print out "Snake eyes!" If double-sixes are rolled, it should print out "Box cars!" Otherwise, it should just print out "Doubles!" For example, a sample run might look like this:

```
>>> doubles()
We will simulate rolling a pair of dice until we get doubles...
5 3
1 4
2 6
1 1
Snake eyes!
```

4. Write a program called **threekind()** that performs a simulation to estimate the probability of rolling *three-of-a-kind* in a single roll of three six-sided dice. For this problem, you should use a for-loop to simulate a fixed number of trials (not a while-loop), and keep track of how many of those trials yielded three equal dice values. Print out the probability as a percentage. Then add more code to also keep track of the number of *two-of-a-kind* rolls encountered (where exactly two of the three values are the same). What is the estimated probability of rolling two-of-a-kind?

5. The **path(*n*)** function simulates the "3*n*+1" sequence, as discussed in class. Modify it so that it keeps track of the *maximum* value reached during the "journey" from *n* to 1, and prints it out at the end. How high up does 27 go? Can you find any numbers under 1000 that rise above 100,000 during their journey?

6. Write a function called **search(*limit*)** that tests all numbers from 2 up to some *limit*, to find out which one takes the longest number of steps to reach 1. Hint: first define a function called **countSteps(*n*)** that is similar to path, but does not print anything out; instead, it should just return the number of steps taken by *n*. Your search function should then use countSteps as a helper. Which number between 2 and 1000 takes the longest number of steps to reach 1 (and how many steps does it take)? What about for numbers up to 100,000?

7. The game of Rock-Paper-Scissors is played by two opponents, and consists of a series of rounds. On each round, the players simultaneously say either "rock", "paper", or "scissors", and the winner of the round is determined as follows:
    - If Rock and Scissors are chosen, Rock wins because Rock dulls Scissors.
    - If Paper and Rock are chosen, Paper wins because Paper covers Rock.
    - If Scissors and Paper are chosen, Scissors wins because Scissors cut Paper.
    - If the choices are the same, no one wins.

    Write a program to simulate a game of Rock-Paper-Scissors between you and the computer. Call your program **rps()**. On each round, the computer should make a random choice and then ask you for your choice, after which it should reveal its choice and report the winner of the round. The game continues until the user enters `quit`. Your program should behave as shown. Notice that invalid choices by the user should be rejected.

```
>>> rps()
Welcome to Rock-Paper-Scissors!

Rock, Paper, or Scissors? paper
You chose paper, I chose scissors
Scissors cut paper, so I win! Ha ha!

Rock, Paper, or Scissors? rock
You chose rock, I chose scissors
Rock dulls scissors, so you win. Hmmm.

Rock, Paper, or Scissors? lizard
Hey, that's not a valid response!

Rock, Paper, or Scissors? scissors
We both chose scissors, so nobody wins.
```

8. Write a program called **craps()** that simulates the casino game of Craps, which is played as follows. You start by rolling two dice and looking at the total. The game then proceeds as follows, based on that first roll:
    - You rolled 2, 3, or 12. Rolling one of these totals on your first roll is called *crapping out* and means that you lose.
    - You rolled 7 or 11. When either of these totals comes up on your first roll, it is a *natural*, and you win.
    - You rolled one of the other totals (4, 5, 6, 8, 9, or 10). In this case, the value you rolled is called your *point*, and you continue to roll the dice until either you roll your point a second time, in which case you win, or you roll a 7, in which case you lose. If you roll any other value (including 2, 3, 11, and 12, which are no longer treated specially), you just keep on rolling until your point or a 7 appears.

9. Did you know that you can estimate the value of π by simply throwing darts randomly at a dart board and counting the number of hits? It's true! Suppose you have a circular dart board that fits just inside of a square cabinet. If you throw darts at random, the proportion that actually hit the dart board inside the cabinet (as opposed to missing the dart board and just hitting the corners of the cabinet) will be determined by the relative area of the dart board and the cabinet. If n is the total number of random darts that land within the confines of the square cabinet, and h is the number of those that land on the circular dart board inside, it is easy to see that π is approximately $4h/n$. Imagine a 2×2 square centered at (0, 0). Its area is $2^2 = 4$. A circle inscribed in this square has radius 1, so its area is π. The ratio of the circle's area to the square's area is thus $\pi/4$, which is approximately the same as the fraction of darts h/n that land inside the circle.

    Using this idea, write a program called **darts()** that asks the user for the number of darts to throw and then performs a simulation to estimate the value of π. You can call `random.uniform(-1, 1)` to generate a random floating-point value uniformly distributed in the range (-1, 1) corresponding to the x and y coordinates of a random point within the square. The point lies inside the inscribed circle if the distance from the point to the center of the circle is less than the circle's radius; that is, if $x^2 + y^2 \leq 1$. How many darts does it take to get a reasonably good approximation for π?

10. Now add graphics to your dart simulation: have your program draw the circle inscribed in the square and plot the random points generated, in addition to reporting the approximation of π. Points that fall inside the circle should be drawn in red, and points outside the circle should be drawn in blue.