Discussion section exercises for the week of September 24.

These problems are similar to the simulated dice game demonstrated in class, and posted on the website. Here we will simulate the tossing of a fair coin, and use it to perform some probability experiments. To get you started, a call to

```
random.randint(0,1)
```

will return 0 or 1 with equal probability. We can treat 0 as 'tails', and 1 a 'heads'.

1. Write a function definition

   ```
   def num_heads(num_tosses):
   ```

   that performs `num_tosses` tosses of the simulated coin and returns the number of tosses that were heads. (You just need a single `for` loop, which will be quite simple.) For example, run this module, and type

   ```
   num_heads(1000)
   ```

   at the prompt in the Python Shell, you will see the number of heads that came up in one thousand tosses of the coin. Note that each time you type this, you will get a different answer, which should be close to 500.

2. You can do this part in the Python shell. Call `num_heads` many different times, as above, trying out values like 100,200, 1000, 10000, etc. for the number of tosses. (I was able to comfortably go up to ten million, which took less than twenty seconds.) For each such call, compute both the *absolute* and *relative* difference from getting exactly half heads. What does this mean? For example, suppose you toss 100 times and get 47 heads. Then the absolute difference is

   $$|50 - 47| = 3$$

   and the relative difference is

   $$\left|\frac{1}{2} - \frac{47}{100}\right| = 0.03$$

   What kind of behavior do you observe in both the absolute and the relative difference as we increase the number of tosses?

3. Now let's do a different kind of experiment. We'll perform, say, ten tosses of the coin, but do this repeatedly, and find out how often we get exactly, say, seven heads. To carry this out, write a function definition

```
def heads_repeated(coins,trials,target):
```

that tosses `coins` coins `trials` times, and returns the number of times that we got exactly `target` heads. Your function should call the function `num_heads` that you wrote for the previous exercise. For example, if I call

```
heads_repeated(10,1000,7)
```

and get the answer 109, it means that I tossed ten coins one thousand times, and that in 109 of those trials, I got exactly 7 heads. Again, you need just a single for loop. Try this out with 10, 20, 50, 100, …,1000 coins, doing a few thousand trials each time, and find out how often *exactly half* the coins in each trial come up heads. (So, for example, if the number of coins is 10, then the target will be 5.) Describe what you find (which may be different from what you expected).

4. If you've gotten this far, write a function definition

```
def longest_run(num_tosses):
```

That returns the length of the longest run of consecutive heads or tails in the given number of tosses. For example, if you toss ten times and get

0,1,1,0,0,1,0,0,0,1

then the length of the longest run is 3. This is trickier to code. HINT: Keep track of several variables, called, say, `longest` and `current_run`. The variable `longest` holds the value of the longest run encountered so far, and `current_run` contains the length of the current run. For example, with the tosses displayed above, the variable `current_run` will take on the values 1,1,2,1,2,1,1,2,3,1

and `longest` will take on the values
1,1,2,2,2,2,2,2,3,3.
You might also want to keep track of the value of the previous toss.
Once again, you need just a single `for` loop, and the code is reasonably short; but the challenge is figuring out where and when to initialize and update the values of these variables. It's a good idea, while developing the code, to print out the values of these variables as well as the value of the toss itself.

When you get it all working, run it several times each with ten, twenty, fifty, one hundred, two hundred,.. tosses. What do you find for the length of the longest run? (Students often find the result surprising.)