Python In The REAL World Pt.106

Written by Greg Walters

This month I've decided to continue our discussion of dealing with data. This time, we will look at the "Law" of Truly Large Numbers.

Why did I decide to put Law in quotes? Because it's not really a law:

• There IS a Law of Large numbers that basically states that if you perform the same experiment a large number of times, the average of the results should be close to the expected outcome.

• The Law of Truly Large Numbers states that "with a large enough sample of data, many odd 'coincidences' are likely to happen." (http://skepdic.com/lawofnumbers.html)

This month, we will experiment to see if we can experience either of these two "laws".

Let's first take a look at random numbers. Computers CAN NOT, by themselves, generate TRULY random numbers. They can get pretty close, and most of us are fine with close enough. But what exactly are random numbers?

A random number is a number that is independent – with no correlations between any successive numbers.

Probability theory says, basically, that if you have two outcomes that are equally likely to occur (the heads of a coin in this case), there is an equally likely chance that either will occur, or in the case of a coin toss, 50% that it will end up with heads and 50% that it will end up with tails.

Michael Crichton's Jurassic Park (either the book or the movie, but the movie is more fun in my opinion) has a good (but simplified) discussion on Chaos Theory where Ian Malcolm (played by Jeff Goldblum) describes the direction that a drop of water, running across the hand of Doctor Ellie Sattler (played by Laura Dern) will take. The same can be said about a coin striking the floor or the palm of your hand. It can skew the result just enough to make it more random.

```
# seed random number generator
seed(1)
# "flip" 10 times and do it twice.
todo = 10
loops = 2
```

Now, let's create a VERY simple Python program to check this out. We will use the numpy library for the random number generator, rather than the built-in Python random number generator. While both are pretty much the same thing, the numpy library has some additional options that make it a better choice for future work. It's not good enough for serious cryptography use, but for what we need, it's fine. Because of the f-string formatting, you will need to use Python 3.7 or greater.

```
from numpy.random import seed
from numpy.random import randint
```

Of course, we start with the imports. In the next line of code, we set the seed value of the random generator to a value of one. If you do this, you will get the same values that I do. To run this independently from me, just comment out the seed(1) line (above).

Now, we'll run the loop ten times and generate 10 random numbers between 0 and 1 (zero = Tails and 1 = Heads). The randint function gets a minimum value, a maximum value and the number of results to return in a list. The reason we use a value of 2 for the maximum value is that numpy takes this value and always returns values 1 less than the maximum.

Now step through the list of returned numbers and count the number of zeros and ones.

```
for loop in range(loops):
    flips = randint(0, 2, todo)
    print(flips)
```

```
heads = 0
tails = 0
for flip in flips:
    if flip == 0:
        tails += 1
    else:
        heads += 1

print(f'Heads: {heads} - Tails: {tails}')
pctHeads = (heads/todo) * 100
print(f'Percentage of Heads: {pctHeads}%')
```

Name your program as cointoss.py and run it. You should see the following output…

```
$ python cointoss.py

[1 1 0 0 1 1 1 1 1 0]
Heads: 7 - Tails: 3
Percentage of Heads: 70.0%
[0 1 0 1 1 0 0 1 0 0]
Heads: 4 - Tails: 6
Percentage of Heads: 40.0%
```

It's not what you would expect to be. You would expect a 50% number of Heads each time. Take a coin and try it. You will find a similar result. It won't be 50% each time. Remember the Chaos Theory?

Now, the value of 10 "flips" is a fairly low number of samples. Let's try it with a larger sample size. Change the todo value to 1000 and re-run your program.

I'm going to shorten the output (shown below) to save space, but here is what you should see…

This time, our results were much closer to 50%, but not really close enough. What would it look like if we do a series of 100000 flips? Change the todo variable to 100000 and re-run the program.

```
[1 1 0 ... 0 0 0]
Heads: 49771 - Tails: 50229
Percentage of Heads: 49.771%
[0 0 0 ... 0 0 1]
Heads: 49943 - Tails: 50057
Percentage of Heads: 49.943%
```

Now we are very close to what we expect the result to be, close enough to say that, yes we do get almost 50% distribution. In addition, we have now seen the Law of Large Numbers take effect.

But what about the "Law of Truly Large Numbers"? One of the examples that is often used to explain this would be (http://improbability-principle.com/the-laws-of-the-improbability-principle/):
"*In July 1975, a taxi in Hamilton, Bermuda knocked Erskine Lawrence Ebbin from his moped, killing him. The year before, his brother Neville Ebbin had been killed by the same driver driving the same taxi and carrying the same passenger while riding the same moped on the same street.*"

In another example,"At a typical football game with 50,000 fans, most fans are likely to share their birthday with about 135 others in attendance. (The notable exception will be those born on February 29. There will only be about 34 fans born on that day.)" I'm guessing that this example uses an American football game, as opposed to true football, but the result would most likely be the same regardless. Let's code another example to test this...

```
from numpy.random import seed
from numpy.random import
randint
import datetime
# seed random number
generator
seed(1)
```

Again, we start off with our imports (we added datetime for this example) and set the seed value. Next we set the number of random numbers in our list to be 50000 and create an empty list.

```
todo = 50000
dates = []
```

```
[1 1 0 0 1 1 1 1 1 0 0 1 0 1 1 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 1 1 1 1 1...
 0 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 0 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0
 0]
Heads: 478 - Tails: 522
Percentage of Heads: 47.8%
[0 0 0 1 0 0 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 1 0 0 1 0 0 0 0...
 1 0 0 1 0 1 0 0 1 1 1 0 1 1 1 0 0 0 1 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 1 0 0
 0]
Heads: 497 - Tails: 503
Percentage of Heads: 49.7%
```

Now we loop through a series of statements that pick valid dates at random. (I use Kite for my programming and they provided the base example for this code. I modified it slightly). Once we have the date, we append that to the list (top right)

```
for tdo in range(todo):
    start_date = datetime.date(1970, 1, 1)
    end_date = datetime.date(2020, 6, 1)

    time_between_dates = end_date - start_date
    days_between_dates = time_between_dates.days
    random_number_of_days = randint(0, days_between_dates)
    random_date = start_date + datetime.timedelta(days=random_number_of_days)
    dates.append(random_date)
```

Finally, we create a date (I picked my son's birthday) to see if it is in the list and print the number of times it occurred, if in fact it did.

```
datetocheck =
datetime.date(1986, 6, 24)

print(f'Found
{dates.count(datetocheck)}
occurrences')
```

You might not be surprised that we got at least a few matches…

```
$python birthdays.py
Found 3 occurrences
```

We could even modify the code to do this a number of times, keep track of the results and at the end, provide an average of the occurrences. I named this "birthdays2.py"

Here's the result (shortened of course):

```
$ python birthdays2.py
```

```
from numpy.random import seed
from numpy.random import randint
import datetime
# seed random number generator
seed(1)
todo = 50000
sampleloops = 100
samples = []
for loop in range(sampleloops):
    dates = []
    for tdo in range(todo):
        start_date = datetime.date(1970, 1, 1)
        end_date = datetime.date(2020, 6, 1)
        time_between_dates = end_date - start_date
        days_between_dates = time_between_dates.days
        random_number_of_days = randint(0, days_between_dates)
        random_date = start_date + \
            datetime.timedelta(days=random_number_of_days)
        dates.append(random_date)
    datetocheck = datetime.date(1986, 6, 24)
    found = dates.count(datetocheck)
    samples.append(found)
    print(f'Found {dates.count(datetocheck)} occurrences')
print(f'Results: {samples}')
print(f'Average is {sum(samples)/len(samples)}')
```

```
Found 4 occurrences
...
Found 4 occurrences
Results: [4, 3, 5, 5, 6, 3,
1, 3, 3, 4, 0, 2, 1, 0, 5, 2,
3, 3, 3, 3, 4, 3, 3, 6, 5,
3, 3, 1, 3, 2, 4, 4, 2, 4, 2,
2, 2, 4, 0, 1]
Average is 2.96
```

I hope that this has given you an appreciation of Large Numbers and Truly Large Numbers and random numbers in general.

I've put the code files up on PasteBin:

Cointoss.py
https://pastebin.com/nXTZ6PLR
Birthdays.py
https://pastebin.com/u5ja3L3E
Birthdays2.py
https://pastebin.com/sfv4RvHi