# Program 1: Working with Data

New Attempt

**Due** Feb 19 by 11:59pm **Points** 20 **Submitting** a file upload **File Types** py and txt
**Available** after Feb 1 at 12am

In this assignment, students shall use Python to create and then detect if a pseudo-random number generator produces results that appear fair.

This assignment will give students experience with:

- Creating a Python script in a file
- Displaying data to the terminal window through use of the print command
- Using modules in a script (i.e. random, timeit) to add functionality to a program
- Working with and creating functions in Python using parameters and return values
- Generating and analyzing a List (Array) of data
- Performing an empirical timing test to witness the impact of complexity on an algorithm

## Details

The user will launch the script using the command:

```
python3 ./randList.py
```

When the program begins it displays a brief welcome with the student's name (I do not need, nor want, your RedID for privacy reasons). It may prompt the user to enter a starting test size, but this is an optional feature with a small amount of extra credit. By default, the program will begin with a test size of 16 coin flip events.

The script calls its coin-flip function which returns a list of integer outcomes. The program may assign a 'heads' value to either 0 or 1, but the choice is arbitrary. Returning the list of random events is the major evaluation point here. The coin-flip function itself shall accept an optional argument (default=10) which specifies the number of times the function should flip the coin/roll the dice. It may also take optional arguments specifying the range of values (0 start, 1 end) should the student choose to expand the program from a simple coin flip to something like a dice roll (d6 or d20). This will allow this single function to produce lists of arbitrary length. Students may choose to create a 'fair' coin-flip function, but where is the fun in that? Alternatively, students may create a function that cheats . . . just a little. Our goal will be to see if we can give our dice a slight edge without looking terribly obvious in the results. Perhaps, if your dice simulator rolls a 1, it allows itself to reroll that single event once.

Using the list generated by the coin-flip function, the program shall estimate the true probability of the coin producing a 'heads' event. Establish this using:

```
p = h/(h+t)
```

The script shall also scan the generated array for the maximum number of continuous heads and tails events. Think of this as the largest run of the same number in the array.

After calculating the heads and tails events for the current iteration (it starts at 16), the program shall double the current test size. Thus, the first step goes from 16 to 32 events. We want to measure how much longer the increased size takes. The program shall perform at least 12 of these ever-increasing tests, (default: 16, 32, 64, ..., 16384, 32768). With each epoch of the test, the program shall display information about the results. Each test shall report its size (number of events), the number of heads events, the number of tails events, the estimate of the event's probability, the largest number of 'head' events in a row, the largest continuous number of 'tail' events in a row, and the time it took to perform the current test.

For example, the program *may* produce output that looks like this:

```
n=00001024,h=00000508,t=00000516,p=0.495610,ch=09,ct=08,time=0.0091455
n=00002048,h=00001020,t=00001028,p=0.497804,ch=10,ct=10,time=0.0178327
n=00004096,h=00002050,t=00002046,p=0.500366,ch=12,ct=10,time=0.0357954
n=00008192,h=00004115,t=00004077,p=0.502258,ch=12,ct=12,time=0.0722982
n=00016384,h=00008087,t=00008297,p=0.493561,ch=13,ct=14,time=0.1312068
n=00032768,h=00016330,t=00016438,p=0.498337,ch=12,ct=18,time=0.2615956
n=00065536,h=00032938,t=00032598,p=0.502586,ch=16,ct=15,time=0.5300924
n=00131072,h=00065756,t=00065316,p=0.501675,ch=19,ct=16,time=1.0329348
n=00262144,h=00131213,t=00130931,p=0.500536,ch=18,ct=16,time=2.0782921
n=00524288,h=00262226,t=00262062,p=0.500155,ch=16,ct=18,time=4.1558857
n=01048576,h=00524295,t=00524281,p=0.500006,ch=19,ct=20,time=8.4246300
n=02097152,h=01049428,t=01047724,p=0.500406,ch=24,ct=20,time=16.724115
```

## Extra Credit

For up to 5 extra points of extra credit, modify the operation so it no longer simulates a simple coin toss and, instead, simulates a dice roll. The user will enter the number of sides on the dice (2=coin), and the program will create an array of values simulating dice rolls. For a 20 sided dice, this will include numbers 1-20 (or 0-19 if you're in love with zero index values). In this solution, students should display the longest continual run of the minimum value in the range as well as the largest continual run of a single instance of one of the other numbers. Identify the continual number as well, so if 4 appeared 13 times, indicate as much. If the most common continuous number was 6, use that instead.

## Delivery

Students shall submit both the python script they developed for this assignment as well as an example of its actual output in an ASCII coded text file.