Lab 2: Monte Carlo Birthday

Stuart Harley

Introduction: Monte Carlo simulations perform a simulation of a large number of experiments and determine the ratio of successes to the overall number. These experiments must be stochastic, e.g. they can have different outcomes and the outcomes are distributed randomly among all the possible outcomes according to some distribution function. So, if we run N experiments, and observe how many times a particular outcome of interest is observed M, we can estimate that the probability of that outcome occurring is M/N.

Learning Outcomes:

- Use python and Jupyter Notebooks to create a Monte Carlo simulation to predict the probablitiy of common birthdays occurring in groups of various sizes.
- Write python methods functions
- Generate documentation and explanations in a Jupyter Notebook

Importing Libraries

In [1]:
```
import random
import time
```

generate_birthdays method: generates n number of random birthdays and checks if there is a repeated birthday in the list of birthdays

In [2]:
```
def generate_birthdays(n = 23):
    """
    Generates a list of n random birthdays represented by ints between 1 and 3
65

    param n: the number of birthdays to generate. The default argument is 23.
    return: a list of n random birthdays
    """
    birthdays = []
    for x in range (n):
        birthdays.append(random.randint(1, 365))
    return birthdays
```

check_for_common_birthday method: checks if there is a repeated birthday in a list of birthdays

```
In [3]: def check_for_common_birthday(birthdays):
            """
            Checks if there is at least 1 common birthday in a list of birthdays

            param birthdays: a list of birthdays
            return: an int value 1 if there is a common birthday, 0 if not
            """
            birthday_set = set()
            for birthday in birthdays:
                birthday_set.add(birthday)
            if len(birthday_set) != len(birthdays):
                return 1
            else:
                return 0
```

get_probability_of_common_birthday method: Calculates the probability of a group of size G having a common birthday based on a number of simulations N. (This is a type of Monte Carlo simulation)

```
In [4]: def get_probability_of_common_birthday(G, N):
            """
            Calculates the probability of a group of size G having a common birthday b
        ased on a number of simulations N.
            This method runs a simulation N times and counts the number of times there
        is a common birthday.
            It then returns the precentage by dividing the count by the total number o
        f simulations.

            param G: the group size
            param N: the number of simulations
            return: a probability of a group of size G having a common birthday based
         on N number of simulations
            """
            count = 0
            for x in range (N):
                birthdays = generate_birthdays(G)
                count += check_for_common_birthday(birthdays)
            return count/N*100
```

Demonstration of the help command (Shows the Docstring)

```
In [5]: help(get_probability_of_common_birthday)
```

User calculation cell: Allows the user to select a group size variable (G) and a number of simulations variable (N) and prints out the probability of a common birthday based on the simulations

In [6]:
```
G = input("Enter in a group size: ")
N = input("Enter a number of simulations to run: ")
G = int(G)
N = int(N)
probability = get_probability_of_common_birthday(G, N)
print("The probability of a group of size " + str(G) + " having a common birth
day is " + str(probability) + "%.")
```

```
Enter in a group size: 20
Enter a number of simulations to run: 1000000
The probability of a group of size 20 having a common birthday is 41.2564%.
```

calc_smallest_group_size_above_50 method: Calculates the smallest group size to have greater than a 50% probability of having a common birthday based on the number of simulations run.

In [7]:
```
def calc_smallest_group_size_above_50(N):
    """
    Calculates the smallest group size to have greater than a 50% probability
 of having a common birthday.

    param N: number of simulations to run for each group size (smaller numbers
will have less reliable results)
    return: the smallest group size to have more than a 50% probability of hav
ing a common birthday based on the
    number of simulations run.
    """
    current_group_size = 1
    while get_probability_of_common_birthday(current_group_size, N) < 50:
        current_group_size += 1
    return current_group_size
```

Used for testing calc_smallest_group_size_above_50 and benchmarking

In [8]:
```
%%time
print(calc_smallest_group_size_above_50(10000))
```

```
23
Wall time: 7.91 s
```

Benchmarking of calculating the smallest group size

Running on the cluster, doing 10,000 simulations per group size: ~2.9 seconds

Running on my laptop, doing 10,000 simulations per group size: ~7.8 seconds

There is a definite difference between the times it took to run this cell on Rosie vs. on my laptop. This would seem to imply that some of the program is parallelizable so it can be run on multiple threads on Rosie, where it is unable to do so on my laptop. Either that or the processors are just faster on Rosie and are able to do the calculations quicker. However, I believe that this program should be highly parallelizable and should probably be able to be done quicker, so I imagine that you must have to specifically tell Rosie how to parallelize the program in order for her to do that.

Conclusion:

1. The probability that a group size of 20 has a common birthday is ~41%. To get this value I ran 1,000,000 simulations and averaged the results. This amount of simulations was a bit overkill, but it did not take too long and it allowed for a very accurate result.

1. The smallest group size to have a probability of greater than 50% that two people share the same birthday is a group size of 23. To get this value I ran the simulation 10,000 times for every increasing group size starting a 1. 23 was the first group size to have an average rate of a common birthday greater than 50%.

1. I found that an N size of at least 10,000 was necessary to provide confidence in my answers. I found that running only 1,000 simulations could provide much greater varying amounts than running 10,000 simulations, but that after 10,000 the results are much more constant.

1. A Hash Collision is caused when two input strings of a hash function produce the same hash result. Because hash functions have infinite input length and a predefined output length, there is inevitably going to be the possibility of two different inputs that produce the same output hash. However, this is very rare. This problem relates to hash collisions because, similarily running a simluation of each group size just once can give you the same result. This is similar to how a simply method for determining a hash code is more liekly to create collisions. However, when you run the simulation multiple times, you get a much better answer. This is similar to a more complex method of determining a hash code that is far less likely to create collisions.