Kernel: SageMath 10.7

Probabilistic Simulation

MATH 242 Modern Computational Math

Sometimes we want to answer questions of the form "what is the probability that...?" or "what is the average number of...?" These questions may have an exact (that is, theoretical) answer, but this answer might be very hard to find. However, we can often use computational simulation to approximate the answer to such questions.

We will begin our study of probabilistic simulation today with some simple problems, before considering a more complicated problem in the next few class sessions.

Simulating coin flips

Suppose we want to simulate an unfair coin that lands on heads with probability p, where $0 < p < 1$. For this, we can use the **random()** function, which returns a random decimal number between 0 and 1.

In [3]:
```
random()
```

Out[3]:
```
0.5728244830246304
```

Let p be a fixed probability value between 0 and 1. When we call **random()**, we may get a value less than p, and this happens with probability p! This suggests that we can simulate a coin flip as follows.

In [7]:
```
# simulate one coin flip
p = 0.7
r = random()
print(r)
result = r < p
print(result)   # True indicates "heads" and False indicates "tails"
```

Out[7]:
```
0.28550796977275283
True
```

Now simulate k flips and determine the number of heads that occur.

In [9]:
```
# simulate the number of heads that occur in k flips of a coin that lands on heads with
probability p
k = 100
result = [ random() < p for i in range(k) ]
print(result)
sum(result)
```

Out[9]:
```
[True, True, True, False, True, True, False, False, True, True, False, True, False, True,
True, True, True, True, True, True, False, False, True, False, True, True, True, False,
True, True, True, False, False, True, True, True, False, True, True, True, False, False,
True, True, False, True, True, False, True, True, True, True, True, True, False, True,
True, True, True, False, False, True, True, True, True, False, False, False, True,
True, True, False, True, True, False, True, True, True, True, True, True, False, True,
False, True, True, True, True, False, True, True, False, True, True, True, True, True,
True, True]
```

```
72
```

Write a function that returns the number of heads that result from k flips of a coin that lands on heads with probability p.

In [10]:
```
# function coinFlips
# input: probability p
```

```
#        number of coin flips k
# output: number of heads in k simulated flips of a coin that lands on heads with
probability p
def coinFlips(p, k):
    result = [ random() < p for i in range(k) ]
    # print(result)
    return sum(result)
```

Try out our coin flips function:

In [13]:
```
coinFlips(0.7, 100)
```

Out[13]: 64

In [14]:
```
coinFlips(0.2, 100)
```

Out[14]: 19

Now write a function that returns the proportion of heads that result from k flips of a coin that lands on heads with probability p.

In [17]:
```
# function proportionHeads
# input: probability p
#        number of coin flips k
# output: proportion of heads in k simulated flips of a coin that lands on heads with
probability p
def proportionHeads(p, k):
    result = [ random() < p for i in range(k) ]
    # print(result)
    return n( sum(result)/k )
```

Confirm that the proportionHeads function works:

In [18]:
```
proportionHeads(0.7, 10000)
```

Out[18]: 0.699900000000000

The Law of Large Numbers

In the context of coin flips, the **law of large numbers** says that as the number of coin flips increases, the sample proportion of heads converges to the theoretical proportion (p) of heads.

In [20]:
```
p = 0.4
print( proportionHeads(p, 10) )
print( proportionHeads(p, 100) )
print( proportionHeads(p, 1000) )
print( proportionHeads(p, 10000) )
print( proportionHeads(p, 100000) )
print( proportionHeads(p, 1000000) )
```

Out[20]:
```
0.400000000000000
0.420000000000000
0.425000000000000
0.395800000000000
0.400760000000000
0.400059000000000
```

Central Limit Theorem

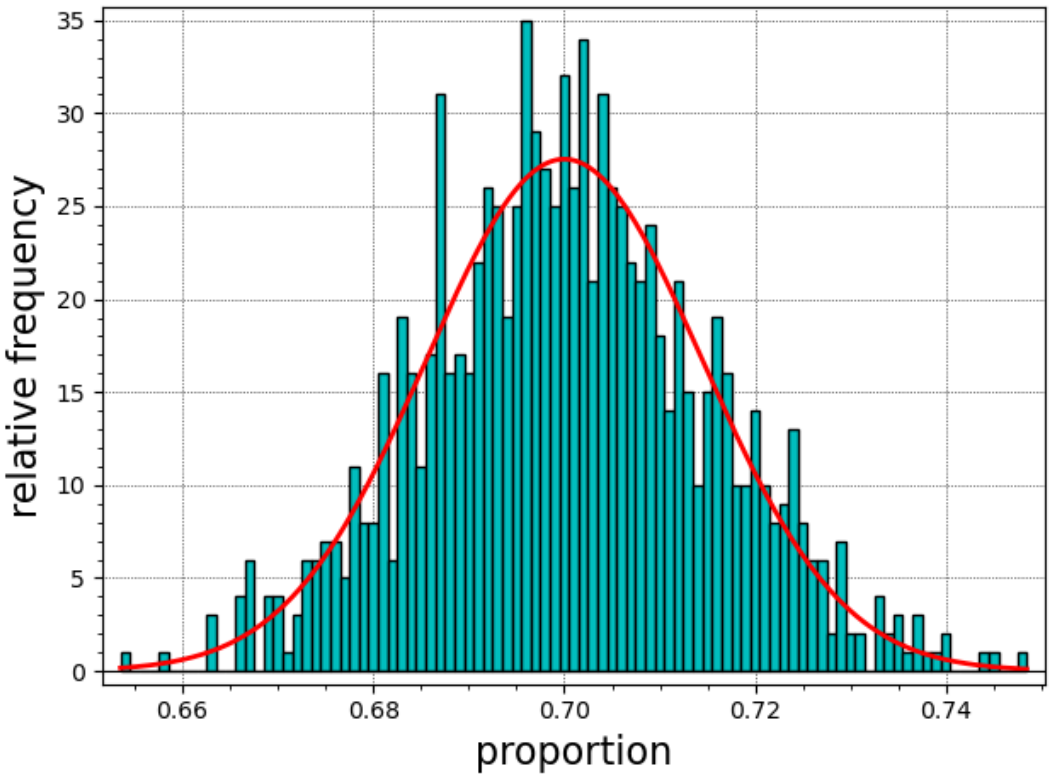
In the context of coin flips, the **central limit theorem** says that as the number of coin flips increases, the *distribution* of the sample proportion becomes normally distributed. The mean of the distribution is the theoretical proportion p, and the variance decreases as the number of coin flips increases.

Let's observe the central limit theorem by plotting the distribution of the proportion of heads in k flips of our unfair coin.

In [22]:
```
# compute N sample proportions, each of k coin flips
N = 1000
p = 0.7
k = 1000
samples = [n(proportionHeads(p, k), digits=3) for i in range(N)]
print(samples)

# make a histogram of the proportions
numBins = round((max(samples) - min(samples))*k + 1)
plotMin = min(samples) - 1/(2*k)
plotMax = max(samples) + 1/(2*k)
H = histogram(samples, density=True, color='c', edgecolor='k', range=[plotMin, plotMax],
bins=numBins, axes_labels=["proportion","relative frequency"], frame=True,
gridlines=True, zorder=2)

# display normal distribution
normpdf(x, mu, sigma) = 1/(sqrt(2*pi)*sigma)*exp(-(x - mu)^2/(2*sigma^2))

sigma = math.sqrt(p*(1-p)/k)
P = plot(normpdf(x, p, sigma), (x, plotMin, plotMax), color='red', thickness=2)

show(H + P)
```

Out[22]:
```
[0.693, 0.710, 0.704, 0.682, 0.696, 0.705, 0.687, 0.692, 0.698, 0.686, 0.713, 0.690, 0.698,
0.703, 0.700, 0.697, 0.679, 0.697, 0.686, 0.679, 0.685, 0.726, 0.710, 0.683, 0.691, 0.712,
0.695, 0.695, 0.723, 0.703, 0.712, 0.698, 0.694, 0.720, 0.715, 0.726, 0.686, 0.692, 0.697,
0.699, 0.700, 0.703, 0.677, 0.713, 0.693, 0.726, 0.709, 0.704, 0.726, 0.705, 0.712, 0.687,
0.680, 0.673, 0.699, 0.704, 0.698, 0.719, 0.676, 0.710, 0.705, 0.710, 0.687, 0.676, 0.708,
0.702, 0.702, 0.697, 0.709, 0.706, 0.686, 0.703, 0.696, 0.700, 0.709, 0.687, 0.724, 0.691,
0.729, 0.701, 0.731, 0.714, 0.688, 0.693, 0.682, 0.687, 0.689, 0.695, 0.687, 0.711, 0.709,
0.684, 0.684, 0.695, 0.710, 0.708, 0.687, 0.687, 0.712, 0.703, 0.700, 0.702, 0.689, 0.688,
0.686, 0.686, 0.745, 0.740, 0.715, 0.713, 0.696, 0.706, 0.699, 0.698, 0.704, 0.685, 0.699,
0.687, 0.704, 0.694, 0.714, 0.678, 0.672, 0.679, 0.693, 0.678, 0.740, 0.705, 0.711, 0.706,
0.686, 0.696, 0.686, 0.693, 0.700, 0.724, 0.687, 0.689, 0.724, 0.724, 0.708, 0.700, 0.696,
0.689, 0.714, 0.707, 0.691, 0.683, 0.698, 0.687, 0.703, 0.700, 0.700, 0.715, 0.687, 0.692,
0.689, 0.697, 0.709, 0.706, 0.704, 0.713, 0.693, 0.712, 0.706, 0.683, 0.707, 0.691, 0.696,
0.700, 0.692, 0.703, 0.675, 0.688, 0.697, 0.692, 0.671, 0.694, 0.722, 0.704, 0.721, 0.667,
0.708, 0.674, 0.696, 0.712, 0.724, 0.686, 0.667, 0.688, 0.680, 0.701, 0.710, 0.684, 0.696,
0.689, 0.692, 0.700, 0.722, 0.715, 0.712, 0.695, 0.713, 0.709, 0.702, 0.673, 0.723, 0.705,
0.702, 0.708, 0.678, 0.684, 0.708, 0.693, 0.694, 0.666, 0.731, 0.703, 0.709, 0.718, 0.697,
0.711, 0.688, 0.703, 0.706, 0.698, 0.696, 0.690, 0.709, 0.704, 0.684, 0.690, 0.702, 0.702,
0.723, 0.698, 0.694, 0.713, 0.687, 0.685, 0.704, 0.695, 0.699, 0.707, 0.699, 0.681, 0.710,
0.683, 0.717, 0.700, 0.691, 0.718, 0.698, 0.699, 0.687, 0.709, 0.706, 0.710, 0.700, 0.707,
0.701, 0.683, 0.698, 0.685, 0.735, 0.716, 0.691, 0.698, 0.700, 0.700, 0.711, 0.702, 0.713,
0.686, 0.716, 0.695, 0.687, 0.721, 0.704, 0.696, 0.702, 0.674, 0.692, 0.689, 0.705, 0.733,
0.715, 0.687, 0.658, 0.696, 0.706, 0.698, 0.703, 0.677, 0.715, 0.713, 0.699, 0.686, 0.696,
0.675, 0.716, 0.701, 0.667, 0.720, 0.727, 0.711, 0.697, 0.719, 0.697, 0.695, 0.706, 0.684,
0.680, 0.692, 0.724, 0.725, 0.712, 0.696, 0.707, 0.689, 0.706, 0.681, 0.698, 0.695, 0.670,
0.691, 0.691, 0.721, 0.696, 0.700, 0.676, 0.705, 0.700, 0.711, 0.715, 0.710, 0.707, 0.720,
0.701, 0.695, 0.704, 0.692, 0.712, 0.701, 0.682, 0.696, 0.733, 0.704, 0.723, 0.702, 0.698,
0.715, 0.708, 0.702, 0.712, 0.696, 0.709, 0.707, 0.694, 0.703, 0.707, 0.714, 0.681, 0.688,
0.706, 0.708, 0.714, 0.709, 0.701, 0.694, 0.719, 0.689, 0.680, 0.690, 0.700, 0.703, 0.675,
0.675, 0.710, 0.663, 0.719, 0.709, 0.679, 0.688, 0.705, 0.729, 0.713, 0.718, 0.717, 0.691,
0.686, 0.687, 0.720, 0.696, 0.708, 0.714, 0.688, 0.699, 0.706, 0.698, 0.744, 0.708, 0.666,
0.681, 0.679, 0.693, 0.692, 0.702, 0.695, 0.705, 0.705, 0.693, 0.702, 0.669, 0.721, 0.698,
0.681, 0.678, 0.700, 0.702, 0.690, 0.674, 0.691, 0.693, 0.676, 0.696, 0.707, 0.689, 0.722,
0.704, 0.683, 0.705, 0.697, 0.691, 0.709, 0.677, 0.703, 0.697, 0.691, 0.684, 0.701, 0.702,
0.692, 0.681, 0.710, 0.687, 0.692, 0.702, 0.712, 0.698, 0.726, 0.701, 0.700, 0.704, 0.694,
0.717, 0.691, 0.693, 0.694, 0.686, 0.701, 0.670, 0.727, 0.702, 0.701, 0.690, 0.693, 0.694,
0.725, 0.736, 0.702, 0.691, 0.682, 0.693, 0.675, 0.707, 0.691, 0.695, 0.685, 0.715, 0.715,
0.698, 0.663, 0.706, 0.697, 0.708, 0.725, 0.711, 0.687, 0.714, 0.702, 0.717, 0.702, 0.719,
0.705, 0.724, 0.710, 0.722, 0.724, 0.703, 0.675, 0.672, 0.697, 0.693, 0.704, 0.684, 0.674,
0.707, 0.673, 0.709, 0.708, 0.709, 0.734, 0.717, 0.709, 0.701, 0.748, 0.716, 0.692, 0.716,
```

```
0.717, 0.681, 0.684, 0.734, 0.693, 0.696, 0.699, 0.706, 0.729, 0.696, 0.698, 0.666, 0.693,
0.685, 0.718, 0.707, 0.700, 0.735, 0.713, 0.696, 0.738, 0.685, 0.739, 0.694, 0.693, 0.686,
0.695, 0.703, 0.705, 0.712, 0.695, 0.681, 0.692, 0.720, 0.683, 0.683, 0.719, 0.720, 0.698,
0.729, 0.719, 0.707, 0.699, 0.702, 0.690, 0.693, 0.697, 0.703, 0.680, 0.699, 0.691, 0.708,
0.678, 0.686, 0.722, 0.704, 0.729, 0.704, 0.718, 0.702, 0.723, 0.720, 0.721, 0.709, 0.699,
0.691, 0.705, 0.699, 0.680, 0.675, 0.715, 0.692, 0.684, 0.724, 0.670, 0.694, 0.700, 0.721,
0.699, 0.688, 0.703, 0.696, 0.716, 0.729, 0.693, 0.710, 0.680, 0.725, 0.687, 0.697, 0.673,
0.697, 0.704, 0.705, 0.705, 0.695, 0.706, 0.705, 0.698, 0.694, 0.693, 0.680, 0.717, 0.676,
0.704, 0.709, 0.715, 0.706, 0.709, 0.700, 0.704, 0.684, 0.692, 0.705, 0.710, 0.685, 0.690,
0.719, 0.697, 0.667, 0.681, 0.701, 0.697, 0.720, 0.698, 0.699, 0.704, 0.703, 0.694, 0.708,
0.683, 0.712, 0.695, 0.696, 0.724, 0.701, 0.711, 0.692, 0.677, 0.708, 0.699, 0.717, 0.692,
0.686, 0.689, 0.724, 0.689, 0.716, 0.696, 0.707, 0.716, 0.692, 0.712, 0.715, 0.681, 0.676,
0.737, 0.702, 0.696, 0.722, 0.718, 0.678, 0.687, 0.697, 0.727, 0.681, 0.710, 0.688, 0.691,
0.712, 0.718, 0.716, 0.683, 0.694, 0.676, 0.694, 0.718, 0.689, 0.696, 0.698, 0.690, 0.688,
0.699, 0.691, 0.737, 0.727, 0.719, 0.701, 0.698, 0.673, 0.705, 0.728, 0.717, 0.684, 0.727,
0.725, 0.696, 0.682, 0.707, 0.716, 0.667, 0.716, 0.683, 0.688, 0.724, 0.700, 0.712, 0.723,
0.716, 0.717, 0.692, 0.706, 0.690, 0.695, 0.701, 0.709, 0.723, 0.721, 0.727, 0.687, 0.720,
0.714, 0.654, 0.707, 0.730, 0.687, 0.702, 0.711, 0.706, 0.690, 0.722, 0.704, 0.690, 0.691,
0.679, 0.688, 0.683, 0.700, 0.700, 0.692, 0.720, 0.701, 0.681, 0.717, 0.692, 0.712, 0.698,
0.704, 0.669, 0.701, 0.667, 0.687, 0.697, 0.687, 0.688, 0.694, 0.701, 0.689, 0.697, 0.700,
0.669, 0.723, 0.724, 0.706, 0.733, 0.733, 0.707, 0.704, 0.699, 0.716, 0.707, 0.700, 0.706,
0.697, 0.669, 0.690, 0.715, 0.702, 0.705, 0.718, 0.689, 0.693, 0.706, 0.697, 0.711, 0.683,
0.725, 0.695, 0.704, 0.712, 0.707, 0.708, 0.703, 0.693, 0.699, 0.708, 0.705, 0.708, 0.701,
0.699, 0.695, 0.723, 0.700, 0.720, 0.696, 0.707, 0.701, 0.696, 0.713, 0.701, 0.663, 0.681,
0.698, 0.711, 0.701, 0.729, 0.679, 0.688, 0.681, 0.717, 0.681, 0.702, 0.670, 0.701, 0.711,
0.679, 0.692, 0.713, 0.696, 0.683, 0.720, 0.702, 0.687, 0.725, 0.687, 0.673, 0.721, 0.685,
0.700, 0.672, 0.702, 0.690, 0.696, 0.689, 0.715, 0.687, 0.695, 0.690, 0.677, 0.721, 0.683,
0.699, 0.687, 0.716, 0.695, 0.704, 0.730, 0.704, 0.705, 0.709, 0.699, 0.711, 0.708, 0.713,
0.707, 0.706, 0.702, 0.697, 0.725, 0.692, 0.703, 0.712, 0.674, 0.706, 0.716, 0.700, 0.717,
0.682, 0.717, 0.685, 0.707, 0.678, 0.706, 0.737, 0.714, 0.709, 0.702, 0.719, 0.683, 0.684,
0.696, 0.697, 0.701, 0.693, 0.704, 0.701, 0.710, 0.688, 0.692, 0.702, 0.708, 0.693, 0.710,
0.702, 0.700, 0.683, 0.705, 0.708, 0.684, 0.697, 0.720, 0.709, 0.735, 0.695, 0.697, 0.687,
0.678, 0.700, 0.714, 0.695, 0.713, 0.717, 0.718, 0.712, 0.694, 0.678, 0.705, 0.695, 0.720,
0.716, 0.683, 0.699, 0.721, 0.728, 0.705, 0.722, 0.699, 0.696, 0.704, 0.693, 0.691, 0.696,
0.717, 0.691, 0.702, 0.687, 0.706, 0.703, 0.709, 0.678, 0.684, 0.692, 0.685, 0.690, 0.696,
0.695, 0.712, 0.713, 0.681, 0.684, 0.674, 0.697, 0.694, 0.716, 0.689, 0.702, 0.716, 0.666,
0.697, 0.704, 0.704, 0.711, 0.705, 0.710, 0.716, 0.678, 0.698, 0.726, 0.686, 0.683]
```



Summary of common random number functions in Sage

The following examples demonstrate random number functions that you might want to use in class or on the homework.

In [23]:
```
# random decimal number between 0 and 1: random()
random()
```

Out[23]: 0.32669302656864685

In [24]:
```
# random number from a range: randrange(min, max, step)
randrange(10, 100, 2)
```

Out[24]: 84

In [27]:
```
# random selection from a list: choice(list)
testList = [2, 3, 5,7, 11]
choice(testList)
```

Out[27]: 5

The Birthday Problem

Here is a classic probability problem that we can approach via simulation.

Suppose that each person's birthday is chosen at random from 365 days in a year. (Ignore leap years, and suppose that birthdays are uniformly distributed.) How many people do you need so that the probability that two people share the same birthday is more than 0.5?

In [28]:
```
# how to count certain items in a list
testList = [1,3,4,2,1,4,1,2]
testList.count(1)
```

Out[28]: 3

In [2]:
```
# one way to generate random birthdays
numPeople = 50
birthdays = [randrange(365) for i in range(numPeople)]
print(birthdays)
```

Out[2]:
```
[363, 80, 354, 178, 25, 340, 304, 31, 84, 220, 158, 331, 277, 140, 221, 115, 81, 234, 75,
98, 179, 150, 5, 80, 146, 172, 22, 239, 167, 125, 176, 243, 101, 147, 76, 176, 132, 160,
245, 212, 158, 31, 341, 111, 165, 276, 342, 218, 280, 128]
```

In [3]:
```
# one way to count the number of people born on each day of the year
counts = [birthdays.count(i) for i in range(365)]
print(counts)
```

Out[3]:
```
[0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 2, 1, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0, 2, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 2, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0]
```

In [4]:
```
# one way to check whether a list of numbers contains a value larger than 1
max(counts) > 1
```

Out[4]: True

In [5]:
```
# function to run one simulation of the birthday problem
# input: number of people to simulate
# output: True if any two people share a birthday; False otherwise
def birthdaySim(numPeople):
    # simulate birthdays
    birthdays = [randrange(365) for i in range(numPeople)]

    # count the number of people born on each day
    counts = [birthdays.count(i) for i in range(365)]

    # return True if there is some count larger than 1; False otherwise
    return max(counts) > 1
```

In [6]:
```
# testing
print( [birthdaySim(10) for i in range(10)]  )
print( [birthdaySim(20) for i in range(10)]  )
print( [birthdaySim(50) for i in range(10)]  )
```

Out[6]:
```
[False, False, False, False, True, False, False, False, False, False]
[False, False, False, False, False, False, True, False, True, True]
[True, True, True, True, True, True, True, True, True, True]
```

In [9]:
```
# estimate the probability of a duplicate birthday
def probDuplicate(numPeople, numTrials):
    # run the simulations
    results = [birthdaySim(numPeople) for i in range(numTrials)]

    # return the proportion of the simulations in which a duplicate birthday occurred
    return n( sum(results)/numTrials )
```

In [10]:
```
#testing
print( probDuplicate(10, 100) )
print( probDuplicate(20, 100) )
print( probDuplicate(50, 100) )
```

Out[10]:
```
0.150000000000000
0.350000000000000
0.940000000000000
```

In [12]:
```
# now find the probability of a duplicate birthday for various numbers of people
numTrials = 10000
for numPeople in range(10, 30):
    prob = probDuplicate(numPeople, numTrials)
    print(f"For {numPeople}, the probability of a duplicate birthday is approximately
{prob.n(digits=4)}.")
```

Out[12]:
```
For 10, the probability of a duplicate birthday is approximately 0.1218.
For 11, the probability of a duplicate birthday is approximately 0.1421.
For 12, the probability of a duplicate birthday is approximately 0.1760.
For 13, the probability of a duplicate birthday is approximately 0.1945.
For 14, the probability of a duplicate birthday is approximately 0.2249.
For 15, the probability of a duplicate birthday is approximately 0.2472.
For 16, the probability of a duplicate birthday is approximately 0.2796.
For 17, the probability of a duplicate birthday is approximately 0.3170.
For 18, the probability of a duplicate birthday is approximately 0.3542.
For 19, the probability of a duplicate birthday is approximately 0.3814.
For 20, the probability of a duplicate birthday is approximately 0.4109.
For 21, the probability of a duplicate birthday is approximately 0.4435.
For 22, the probability of a duplicate birthday is approximately 0.4752.
For 23, the probability of a duplicate birthday is approximately 0.5091.
For 24, the probability of a duplicate birthday is approximately 0.5373.
For 25, the probability of a duplicate birthday is approximately 0.5667.
For 26, the probability of a duplicate birthday is approximately 0.6078.
For 27, the probability of a duplicate birthday is approximately 0.6192.
For 28, the probability of a duplicate birthday is approximately 0.6566.
For 29, the probability of a duplicate birthday is approximately 0.6712.
```

The simulation shows that the smallest number of people such that there is at least a 50% probability of some shared birthday is 23 people!

Extensions

If you found the answer to the birthday problem above, then consider one or more of the following questions.

Given N people:

- What is the probability that at least K people share the same birthday?
- What is the probability that exactly K people share the same birthday?
- What is the probability that two people share consecutive birthdays?
- What is the probability that K people have birthdays within Δ days of each other.

To make these problems more concrete, you may choose specific values for parameters such as N and K.

In [0]:

In [0]:

In [0]: