Coin Flips and Die Rolls

In [1]:
```
# import numpy
import numpy as np
```

1. Two fair coin flips produce exactly two heads

In [2]:
```
# simulate 1 million tests of two fair coin flips
tests = np.random.randint(2, size=(int(1e6), 2))

# sums of all tests
test_sums = tests.sum(axis=1)

# proportion of tests that produced exactly two heads
(test_sums == 0).mean()
```

Out[2]: 0.25034899999999999

2. Three fair coin flips produce exactly one head

In [3]:
```
# simulate 1 million tests of three fair coin flips
tests = np.random.randint(2, size=(int(1e6), 3))

# sums of all tests
test_sums = tests.sum(axis=1)

# proportion of tests that produced exactly one head
(test_sums == 2).mean()
```

Out[3]: 0.37620500000000001

3. Three bias coin flips with P(H) = 0.6 produce exactly one head

In [4]:
```
# simulate 1 million tests of three bias coin flips
# hint: use np.random.choice()
tests = np.random.choice([0, 1], size=(int(1e6), 3), p=[0.6, 0.4])

# sums of all tests
test_sums = tests.sum(axis=1)

# proportion of tests that produced exactly one head
(test_sums == 2).mean()
```

Out[4]: 0.28728900000000002

4. A die rolls an even number

In [5]:
```
# simulate 1 million tests of one die roll
tests = np.random.choice(np.arange(1, 7), size=int(1e6))

# proportion of tests that produced an even number
(tests % 2 == 0).mean()
```

Out[5]: 0.499473

5. Two dice roll a double

In [6]:
```
# simulate the first million die rolls
first = np.random.choice(np.arange(6), size=int(1e6))

# simulate the second million die rolls
second = np.random.choice(np.arange(6), size=int(1e6))

# proportion of tests where the 1st and 2nd die rolled the same num
ber
(first == second).mean()
```

Out[6]: 0.16652500000000001