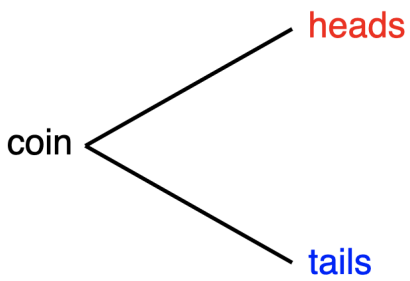
Flipping Coins

In this lesson, we will discuss a common problem with random variables: flipping coins.

WE'LL COVER THE FOLLOWING

Flipping once

Now that we have discussed random number generators, let's look at the most common example used in probability: **flipping a coin**.



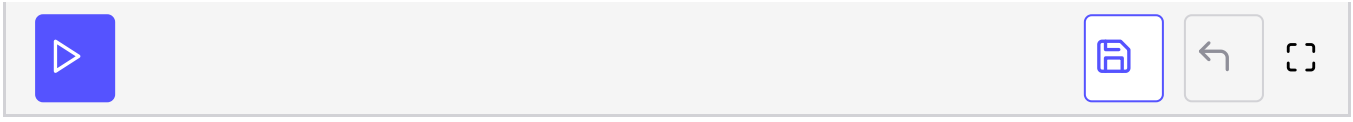
Let's flip a coin 100 times and count the number of heads, denoted by 0, and the number of tails, denoted by 1:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

flips = rnd.randint(0, 1+1, 100)    #generating random numbers between 0 and 1
headcount = 0
tailcount = 0
for i in range(100):
  if flips[i] == 0:
```

```
    headcount += 1
  else:
    tailcount += 1


print('number of heads:', headcount)
print('number of tails:', tailcount)
```

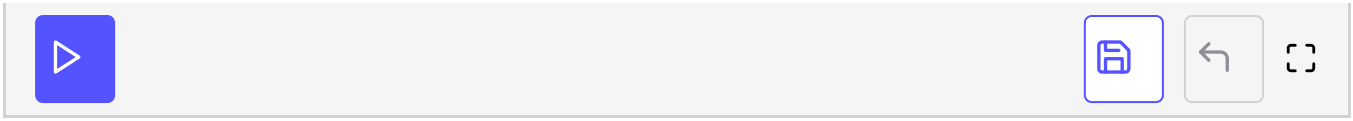


First of all, note that the number of heads and tails adds up to 100. Also, note how we counted the heads and tails. We created counters `headcount` and `tailcount`, looped through all flips, and added 1 to the appropriate counter. Instead of a loop, we could have used a condition for the indices combined with a summation, as follows:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

flips = rnd.randint(0, 1+1, 100)
headcount = len(flips[flips == 0])  #length of array with zeros
tailcount = len(flips[flips == 1])  #length of array with ones

print('number of heads:', headcount)
print('number of tails:', tailcount)
```



In lines 6 and 7, we use `==` condition on the array `flips` which returns an array with all the values that satisfy the condition.

- `flips[flips == 0]` returns an array of all the zeros in `flips`. The length of this array is stored in `headcount`.
- `flips[flips == 1]` returns an array of all the ones in `flips`. The length of this array is stored in `tailcount`.

The code above is easy, but if we do an experiment with more than two outcomes, it may be cumbersome to count the non-zero items for every possible outcome. So, let's try to rewrite this part of the code using a loop. For this specific case, the number of lines of code doesn't decrease, but when we have an experiment with many different outcomes this will be much more efficient. Note that `dtype = 'int'` sets the array to integers.

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

n = 2    # number of outcomes
outcomes = np.zeros(n, dtype='int') # Two outcomes. heads are stored in outcome[0], tails in
flips = rnd.randint(0, n, 100)

for i in range (n):
  outcomes[i] = len(flips[flips == i])    # stores the length of of each outcome in outcomes
  print('outcome ', i, ' is ', outcomes[i])
```

Lines 9 - 11 take care of the number of outcomes even if there are more than 2 outcomes.

Flipping multiple times

Next, we are going to flip a coin twice,100 times, and count the number of tails. We generate a random array of We generate a random array of 0 (heads) and 1 (tails) with two rows (representing two coin flips) and 100 columns. The sum of the two rows represents the number of tails. The `np.sum` function takes an array and by default sums all the values in the array and returns one number. In this case we want to sum the rows. For that, the sum function has a keyword argument called axis. `axis = 0` sums over index `0` of the array (the rows), `axis = 1` sums over the index `1` of the array (the columns), etc.

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

rnd.seed(55)    # to make sure the output of rnd is consistent
flips = rnd.randint(low=0, high=1+1, size=(2, 100)) # random array of dimensions 2 x 100
tails = np.sum(flips, axis=0)
number_of_tails = np.zeros(3, dtype='int')

for i in range(3):
  number_of_tails[i] = np.count_nonzero(tails == i)

print('number of 0 tails:', number_of_tails[0])
print('number of 1 tail:', number_of_tails[1])
print('number of 2 tails:', number_of_tails[2])
```

Plotting results

Bar graph

For the representation of the data, the outcome of the experiment can be plotted with a bar graph.

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

flips = rnd.randint(low=0, high=1+1, size=(2, 100)) # random array of dimensions 2 x 100
tails = np.sum(flips, axis = 0)
number_of_tails = np.zeros(3, dtype = 'int')
for i in range(3):
    number_of_tails[i] = np.count_nonzero(tails == i)

plt.bar(range(0, 3), number_of_tails)
plt.xticks(range(0, 3))
plt.xlabel('number of tails')
plt.ylabel('occurence in 100 trials');
```

Cumulative probability

We compute the experimental probability of getting 0 tails, 1 tail, and 2 tails through division by the total number of trials

> 💡 One trial is two coin flips.

The three probabilities add up to 1. The cumulative probability distribution is obtained by cumulatively summing the probabilities using the `cumsum()` function of `numpy`.

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

flips = rnd.randint(low=0, high=1+1, size=(2, 100)) # random array of dimensions 2 x 100
tails = np.sum(flips, axis = 0)
number_of_tails = np.zeros(3, dtype='int')
for i in range(3):
    number_of_tails[i] = np.count_nonzero(tails == i)

# number_of_tails was computed two code cells back
```

```
prob = number_of_tails / 100
# So cum_prob[0] = prob[0], cum_prob[1] = prob[0] + prob[1], etc.


cum_prob = np.cumsum(prob)
print('cumulative probability ', cum_prob)
```

The first value is the probability of throwing 0 tails. The second value is the probability of 1 or fewer tails, and the third value is the probability of 2 or fewer tails. The probability is computed as the number of tails divided by the total number of trials.

The cumulative probability distribution is plotted with a bar graph:

```
import numpy as np
import matplotlib.pyplot as plt
import numpy.random as rnd

flips = rnd.randint(low=0, high=1+1, size=(2, 100)) # random array of dimensions 2 x 100
tails = np.sum(flips, axis = 0)
number_of_tails = np.zeros(3, dtype = 'int')
for i in range(3):
    number_of_tails[i] = np.count_nonzero(tails == i)

# number_of_tails was computed two code cells back
prob = number_of_tails / 100
# So cum_prob[0] = prob[0], cum_prob[1] = prob[0] + prob[1], etc.
cum_prob = np.cumsum(prob)
print('cumulative probability ', cum_prob)

plt.bar(range(0, 3), cum_prob)
plt.xticks(range(0, 3))
plt.xlabel('number of tails in two flips')
plt.ylabel('cumulative probability');
```

In the next lesson, we will learn about the implementation of a Bernoulli variable in Python.