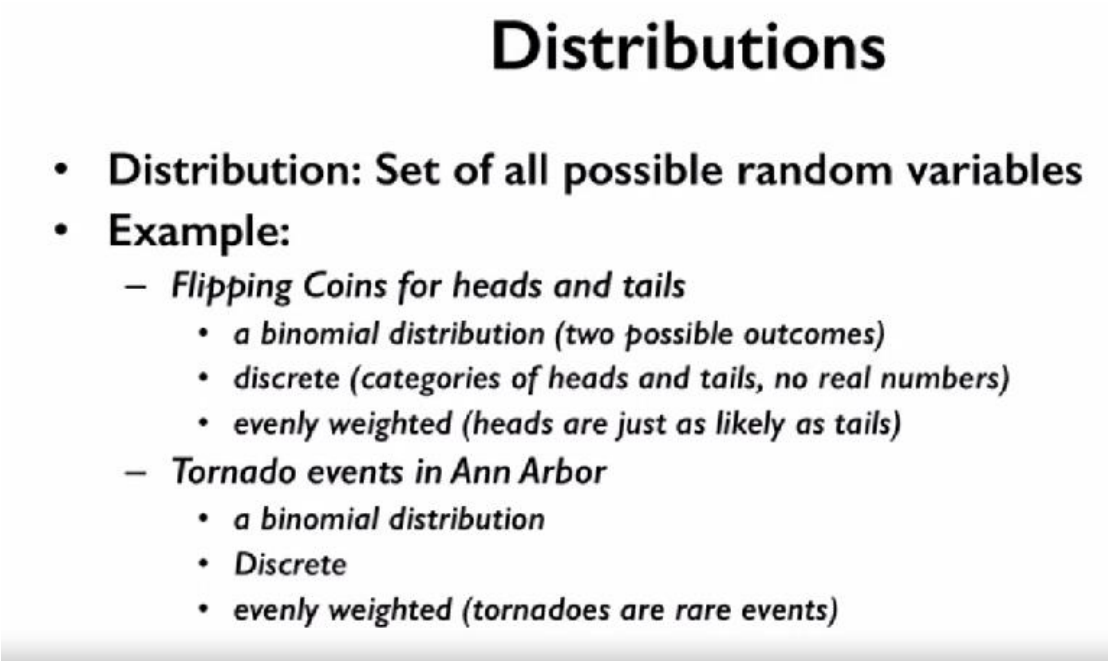
Distributions

Let's start with a discussion of distributions.
And we'll start with the most common using a coin.
When a coin is flipped it has a probability of landing heads up and a probability of landing tails up.
If we flip a coin many times we collect a number of measurements of the heads and tails that landed face up and intuitively we know that the number of times we get a heads up will be equal about the number of times we get a tails up for a fair coin.
If you flipped a coin a hundred times and you received heads each time you'd probably doubt the fairness of that coin.
We can consider the result of each flip of this coin as a random variable.

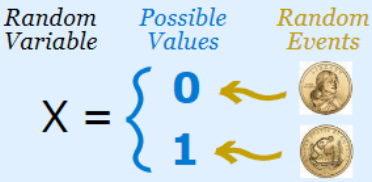


A Random Variable is a set of possible values from a random experiment.
Random Variables can be either Discrete or Continuous:

- Discrete Data can only take certain values (such as 1,2,3,4,5)
- Continuous Data can take any value within a range (such as a person's height)

Example: Tossing a coin: we could get Heads or Tails.

Let's give them the values **Heads=0** and **Tails=1** and we have a Random Variable "X":



In short:

$$X = \{0, 1\}$$

Note: We could choose Heads=100 and Tails=150 or other values if we want! It is our choice.

When we can consider the set of all possible random variables together we call this a distribution.

In this case the distribution is called binomial since there's two possible outputs a heads or a tails.
It's also an example of a discreet distribution since there are only categories being used a heads and a tails and not real numbers.

NumPy actually has some distributions built into it allowing us to make random flips of a coin with given parameters.
Let's give it a try.

Here we ask for a number from the NumPy binomial distribution.
We have two parameters to pass in.
The first is the number of times we want it to run.
The second is the chance we get a zero, which we will use to represent heads here.
Let's run one round of this simulation.

```
import pandas as pd
import numpy as np
```

```
np.random.binomial(1, 0.5)
```

```
0
```

Great so if you're following along in a Jupyter notebook you either got a zero or a one.
And half of you got a value that agreed with the one that I got.

What if we run the simulation a thousand times and divided the result by a thousand.

Well you see a number pretty close to 0.5 which means half of the time we had a heads and half of the time we had a tails.

```
np.random.binomial(1000, 0.5)/1000
```

```
0.475
```

Of course an even weighted binomial distribution is only one simple example.
We can also have unevenly weighted binomial distributions.
For instance what's the chance although we're tornado today while I'm filming.

It's pretty low even though we do get tornadoes here.
So maybe there a hundredth of a percentage chance.
We can put this into a binomial distribution as a weighting in NumPy.
If we run this 100,000 times we see there are pretty minimal tornado events.

```
chance_of_tornado = 0.01/100
np.random.binomial(100000, chance_of_tornado)
```

```
8
```

So you might be wondering why I'm talking about such a simple and intuitive distribution.
I mean we all understand flipping a coin for when we had to make important decisions as children.

But what I want to demonstrate is that the computational tools are starting to allow us to simulate the world which helps us answer questions.
I could have shown you the math behind this so we could have worked out the probabilities.
But a simulation is essentially another form of inquiry.
Let's take one more example.
Let's say the chance of a tornado here in Ann Arbor on any given day, is 1% regardless of the time of year.
That's higher than realistic but it makes for a quicker demo.
And let's say if there's a tornado I'm going to get away from the windows and hide, then come back and do my recording the next day.
So what's the chance of this happening two days in a row?

Well we can use the binomial distribution in NumPy to simulate this.

Here we create an empty list and we create a number of potential tornado events by asking the NumPy binomial function using our chance of tornado.
We'll do this a million times which is just shy of 3,000 years' worth of events.

This process is called sampling the distribution.

Now we can write a little loop to go through the list and look for any two adjacent pairs of ones which means that there were two days that had back to back tornadoes.

We see that this ends up being roughly 102 day tornado events over the 3,000 years.
Which frankly is still too many for me.

```
chance_of_tornado = 0.01

tornado_events = np.random.binomial(1, chance_of_tornado, 1000000)

two_days_in_a_row = 0
for j in range(1,len(tornado_events)-1):
    if tornado_events[j]==1 and tornado_events[j-1]==1:
        two_days_in_a_row+=1

print('{} tornadoes back to back in {} years'.format(two_days_in_a_row, 1000000/365))
```

```
91 tornadoes back to back in 2739.72602739726 years
```

My point here though is that modern computational power allows us to very quickly simulate the effects of different parameters in a distribution.
Leading to a new way of understanding the problem.
You don't have to work out all the math you can quite often simulate the problem instead and observe the results.

In the next lecture we'll talk a little bit more about distributions.