EASC2410 Lecture 13

Statistics Basics: Probability and Distribution

Dr. Binzheng Zhang
Department of Earth Sciences

Review of Lecture 12:

- What is debugging?
- How to avoid debugging?
  - Start small - decompose your large project to small pieces
  - Keep it working - adding new codes to functional pieces
- How to debug?
  - Understanding error messages
  - Use the print() function

In Lecture 13, you will learn:

- What is Statistics?
- What is the expectation from a probability?
- What is a probability distribution function?
- How to use Python to generate probability distributions

What is Statistics?

Statistics is the way we analyze, interpret and model data. To do it properly we need to understand a few concepts:

Important concepts in Statistics

1) **accuracy** versus **precision**
2) **population** versus **sample**
3) **probability** and **expectation**
4) **probability distribution functions**
    a. **theoretical**
    b. **empirical**

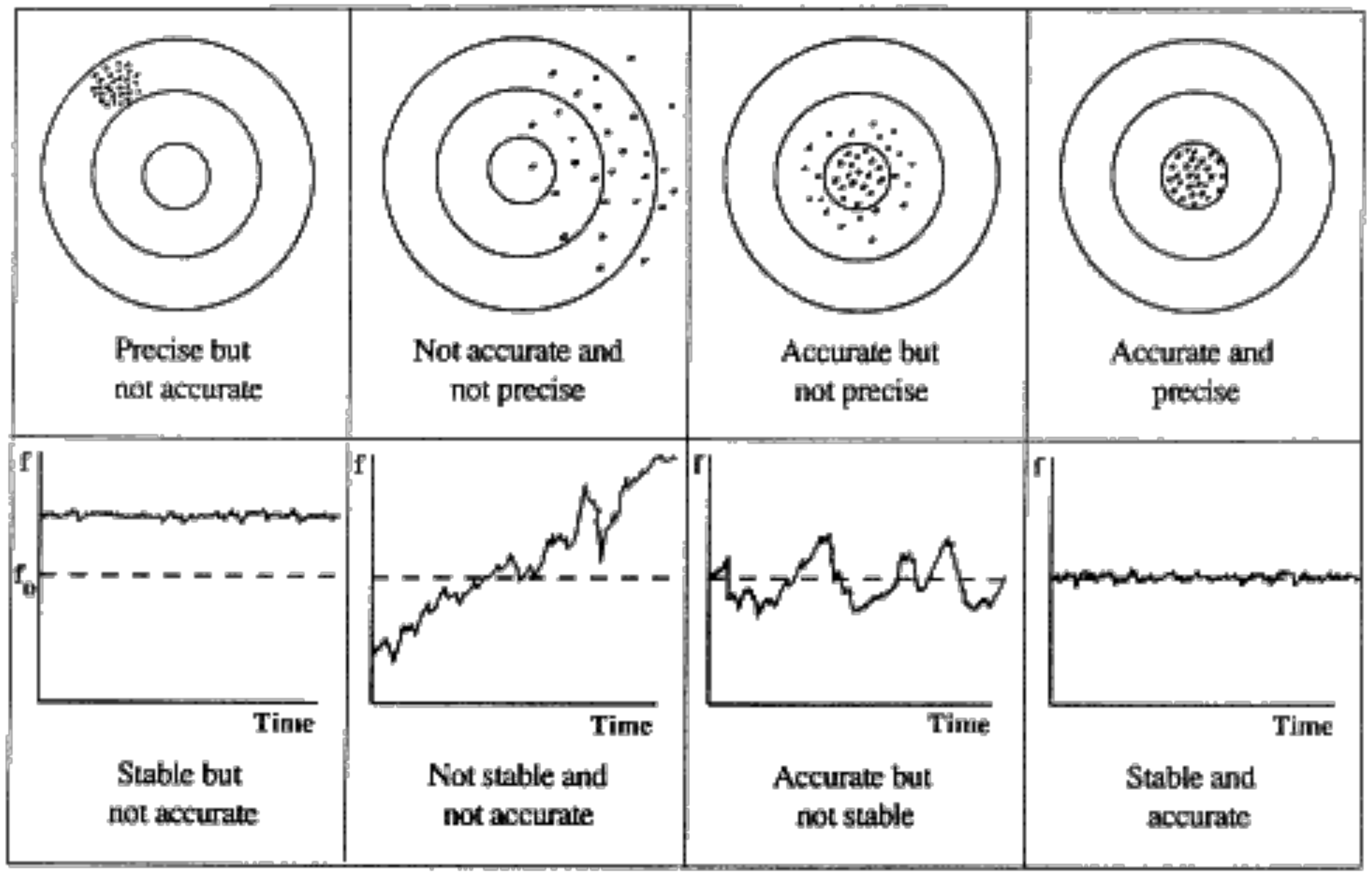
Accuracy and Precision

Definitions: accuracy is how close your data are to the "truth"
precision is the reproducibility of your data.

Example 1: Mathematical approximation of π:

- Accurate and precise: 3.141592653
- Accurate but imprecise: 3.1
- Inaccurate but precise: 3.5893627002
- Inaccurate and imprecise: 4

Example 2: Data collections:



Population and Sample

Definitions:

- A **population** in statistics is every possible outcomes of a given measurement.
- A population can still be (and usually is) a subset of a larger population

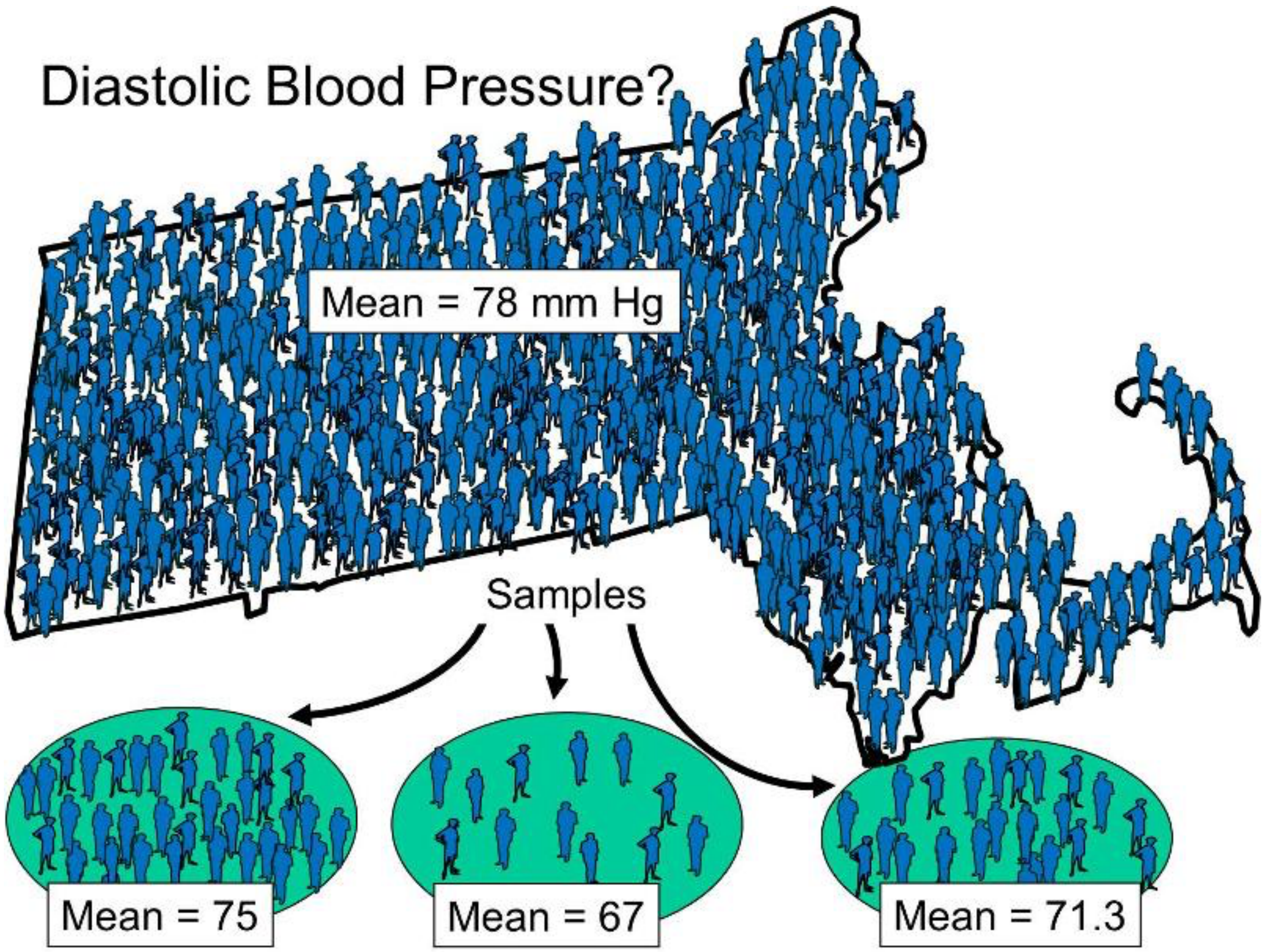
Examples:

- For example, for a study into HKU undergraduate students, the statistical population will be all the HKU students enrolled in undergrad courses
- A sample will be a subset of the possibilities, for example, the students enrolled in EASC2410.

Population
Sample

Population **Sample**

Is a sample representative of the population?



Probability

Definitions: Probability is the measure of how likely it is for a particular event to occur. If something A is impossible, it has a probability of zero. Mathematically, we use the following expression to say event A has a zero probability:

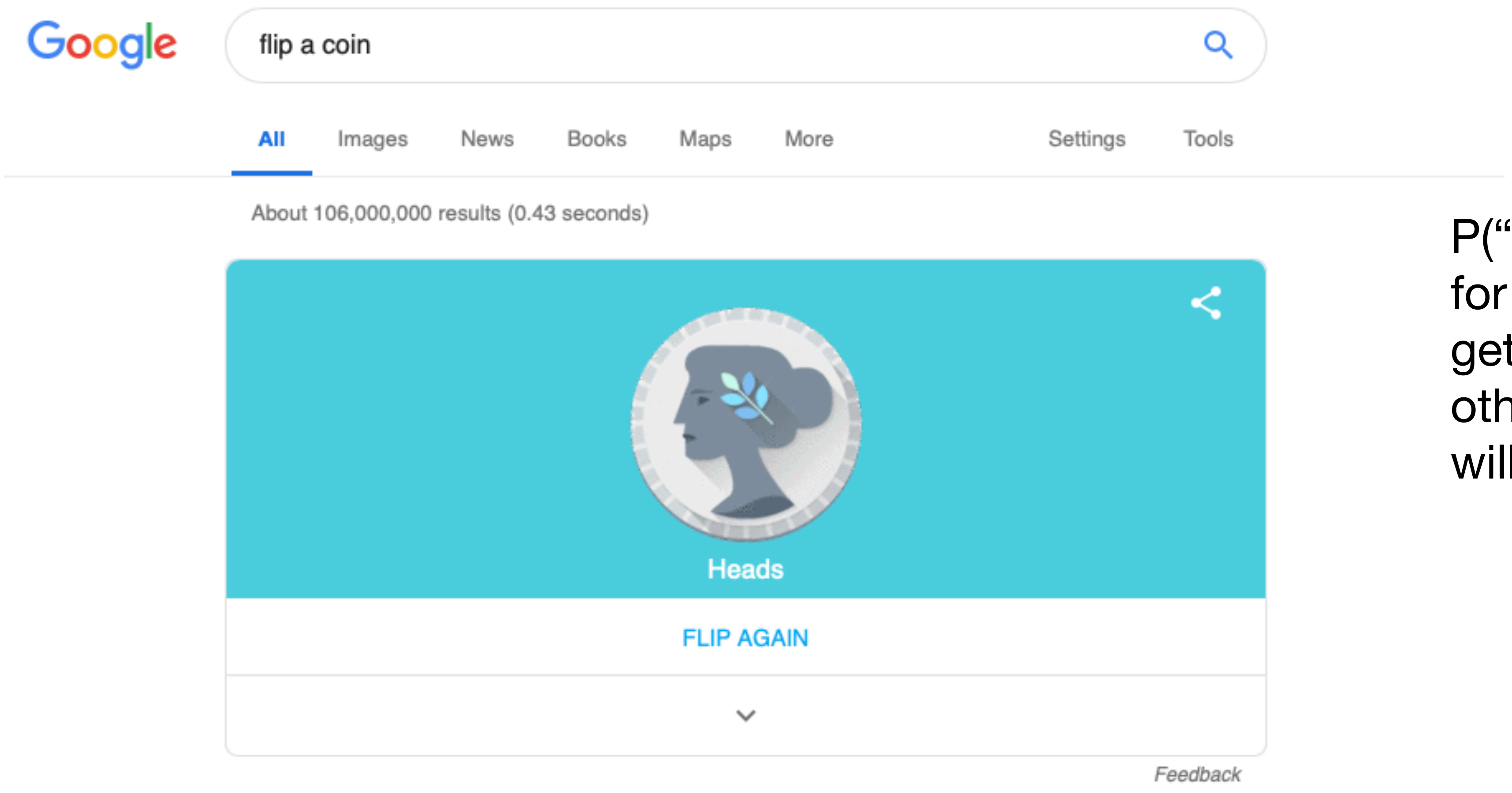
$$P(A) = 0$$

If another event B is a certainty, it has a probability

$$P(B) = 1$$

Examples: When you flip a coin, the probability of getting "head" is

$$P(\text{"head"}) = 0.5$$



P("head") = 0.5 means that for 50% of the time, you will get a "head". And for the other 50% of the time, you will get a "Tail"

Expectation

Definitions: The expected value of X, where X is a discrete random event (variable), is a weighted average of the possible values that X can take, each value being weighted according to the probability of that event occurring. The expected value of X is usually written as E(X):

$$E(X) = S \times P(X = x)$$

So the expected value is the sum of:
[(each of the possible outcomes) × (the probability of the outcome occurring)].

In more concrete terms, the expectation is what you would expect the outcome of an experiment to be on average. In most data analysis, it is the mean of a data set.

Example 1:

Suppose you play the coin flipping game with a friend, the rule follows:

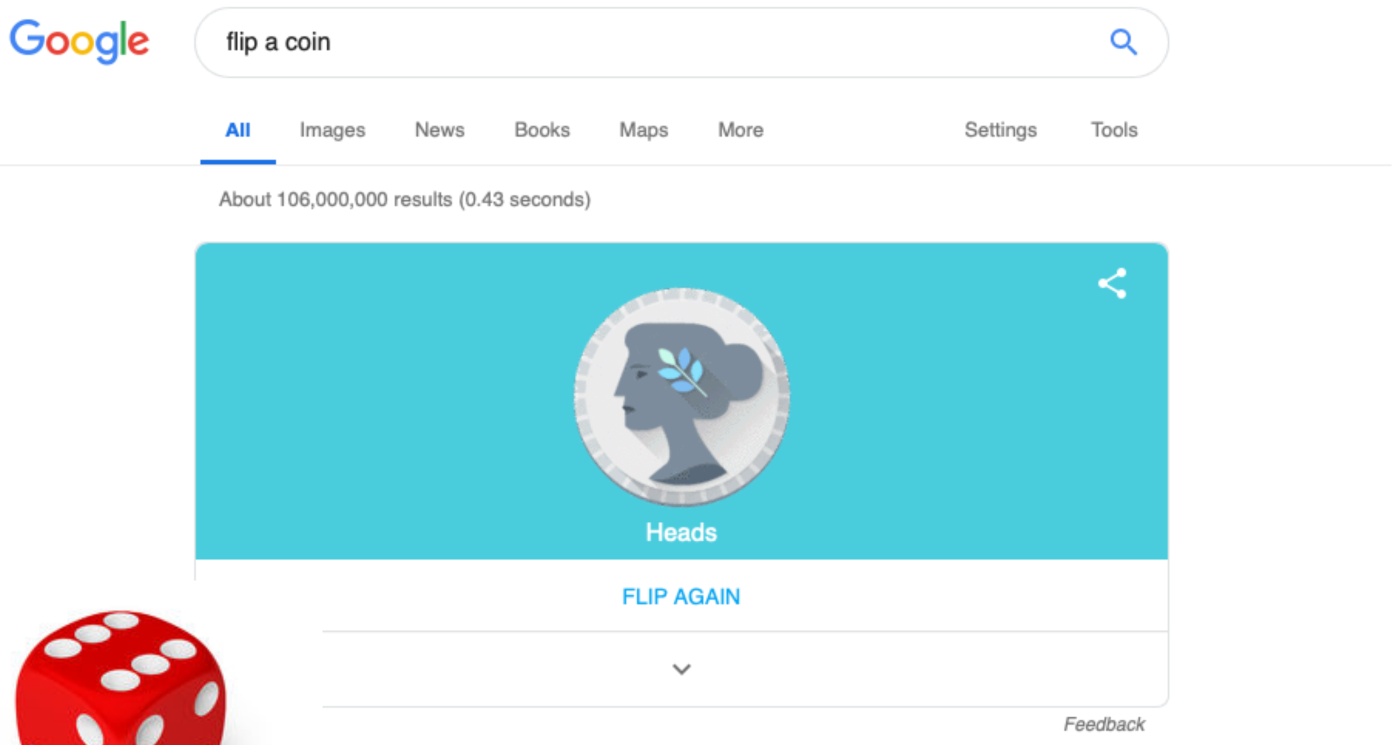
- If you get "Head", you win $10
- if you get "Tail", you lose $10

What's the expectation from this game?

$E = 10^* (50\%) + (-10)^*50\% = 0$

It's a fair game!

Question: What's the expectation of rolling a fair dice'



About 106,000,000 results (0.43 seconds)

Expectation (House Advantage)

Not all the games are fair!

The Roulette game:

- **Assume it's a fair wheel**
- **36 slots with 36 numbers**
- **if you bet the correct number, the dealer gives you 34 times (or lower) of the money you bet**
- **otherwise you lose your bet**

Why it's not fair? let's take a look at the expectation of winning as a player when you bet $1:

$$E(player) = (\underbrace{+\$34}_{\text{Amount of win}}) \times \overbrace{\frac{1}{36}}^{\text{Chance of win}} + (\underbrace{-\$1}_{\text{Amount of lose}}) \times \underbrace{\frac{35}{36}}_{\text{Chance of lose}} = -\$\frac{1}{36}$$

The expectation is always negative for players, welcome to the real world…

Expectation (Game theory)

Now let's play a seemingly fair game. You met a beautiful lady (or gentleman) in a bar, and she offered you to play a game by comparing two coins: if both of you show "Head", you win \$3; if both of you show "Tail", you win 1\$; otherwise you lose \$2. See the following chart:

		x	1-x
	You / Lady	Head	Tail
y	Head	3	-2
1-y	Tail	-2	1

Now the question is, is there a way that she could make you lose money all the time? - the answer seems to be NO, but unfortunately, it is YES. Here the "Expectation" tells you why

Let's say the chance of you showing "Head" is x, which means the chance of you showing "Tail" is 1-x; similarly, the chance of lady showing "Head" is why, leaving her chance of showing "Tail" is 1-y. The expectation of you winning money is calculated as:

$$E(you) = 3 \times xy + 1 \times (1-x)(1-y) - 2 \times (1-x)y - 2 \times x(1-y)$$
$$= 8xy - 3y - 3x + 1$$

Now let's let *E*(you) < 0

$$E(you) = 8xy - 3y - 3x + 1 = y(8x-3) - 3x - 1 < 0$$

Expectation (Game theory)

Now let's play a seemingly fair game. You met a beautiful lady in a bar, and she offered you to play a game by comparing two coins: if both of you show "Head", you win \$3; if both of you show "Tail", you win 1\$; otherwise you lose \$2. See the following chart:

		x	1-x
	You / Lady	Head	Tail
y	Head	3	-2
1-y	Tail	-2	1

Now let's let $E(you) = 8xy - 3y - 3x + 1 = y(8x-3) - 3x - 1 < 0$

Which means $y(8x-3) < 3x - 1$

Then it becomes a middle-school algebra problem:

- if $8x - 3 > 0 \Rightarrow x > \dfrac{3}{8}$ we have $y < \dfrac{3x-1}{8x-3} < \dfrac{2}{5}$ or $y < \dfrac{2}{5}$
- if $8x - 3 < 0 \Rightarrow x < \dfrac{3}{8}$ we have $y > \dfrac{3x-1}{8x-3} > \dfrac{1}{3}$ or $y > \dfrac{1}{3}$

The above calculation means no matter what value of x is, as long as $\dfrac{1}{3} < y < \dfrac{2}{5}$, E(you) < 0!

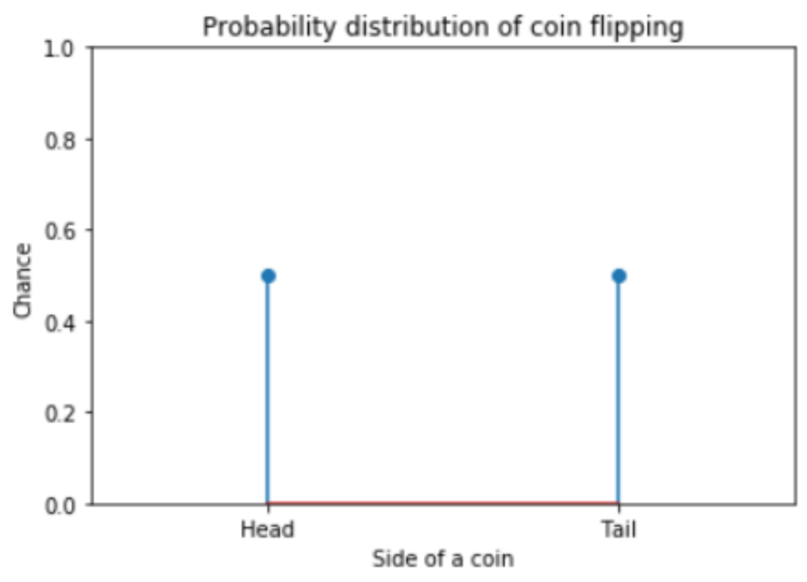
This is basically how stock market works

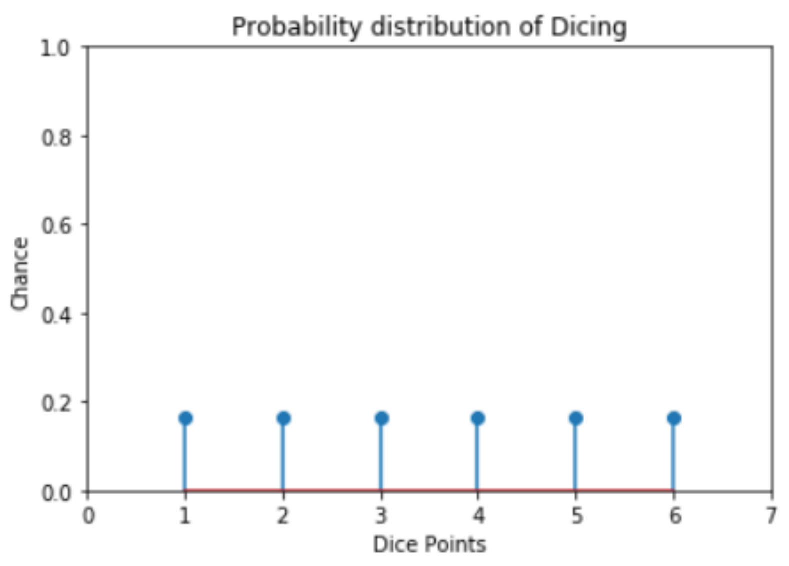
Probability Distributions

Definition:

*A **probability distribution** is a list of all of the possible outcomes of a random variable along with their corresponding probability values.*

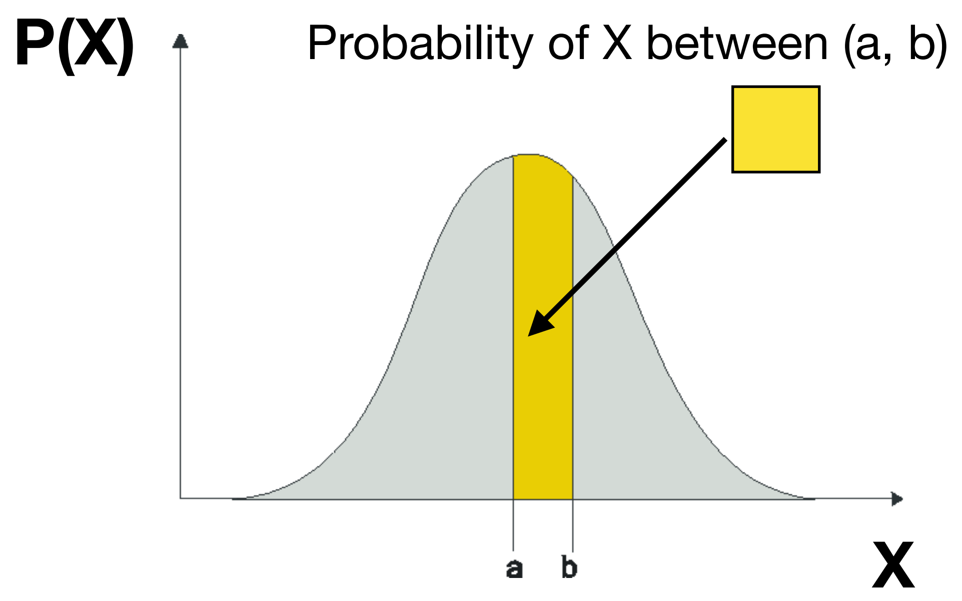
Discrete distributions:
discrete univariate probability distribution with finite support.





A function that represents a discrete probability distribution is called a **probability mass function.**

Continues distributions:
continuous univariate probability distribution with infinite support.



A function that represents a continuous probability distribution is called a **probability density function.**

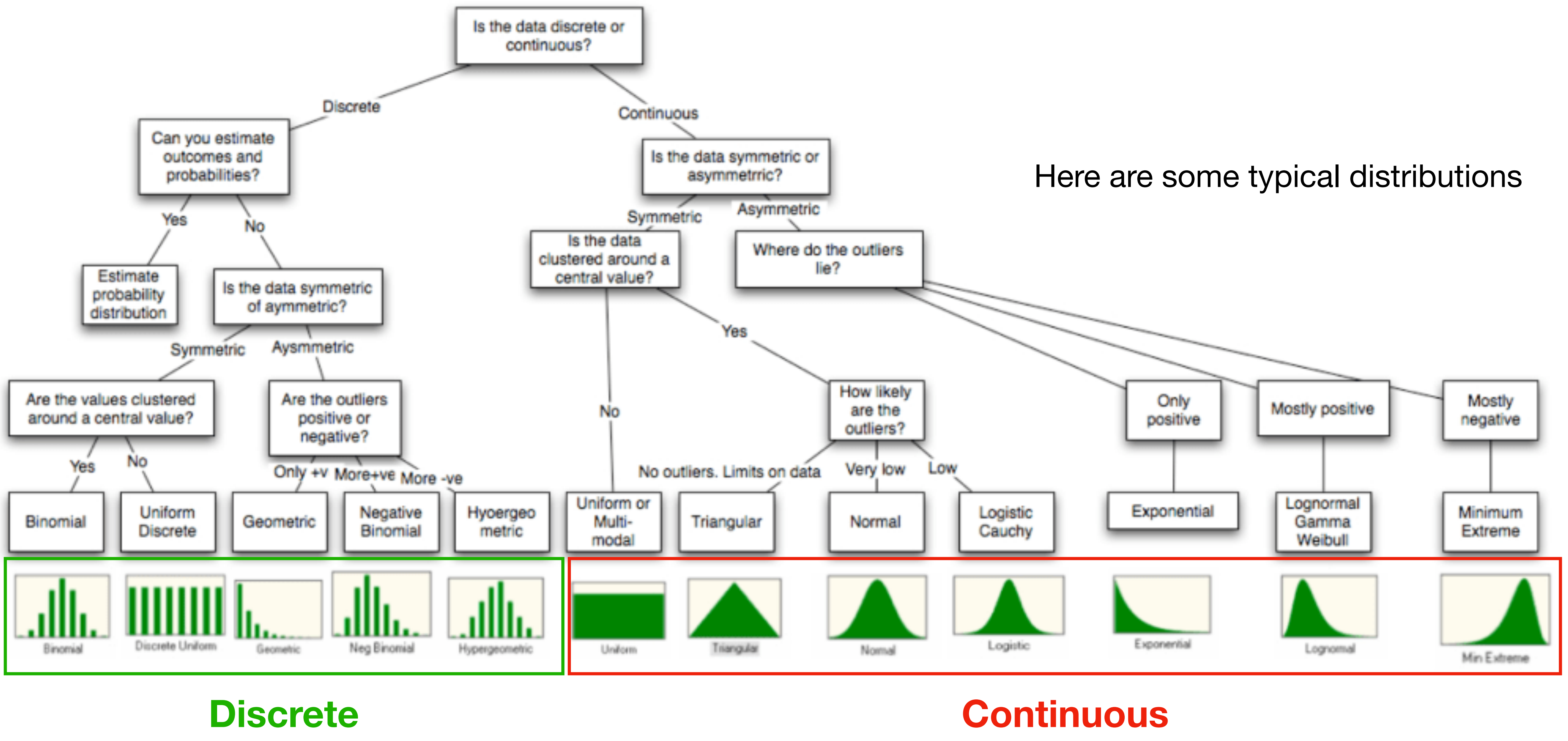
The probability between **a** and **b** is the **area** surrounded by the probability distribution curve.

Question: for a continuous probability distribution, what's the probability of P(a)?

Theoretical Distributions (reference only)

- A **probability distribution** is a list of outcomes and their associated probabilities.
- We can write small distributions with tables but it's easier to summarise large distributions with functions.
- A function that represents a discrete probability distribution is called a **probability mass function**.
- A function that represents a continuous probability distribution is called a **probability density function.**
- Functions that represent probability distributions still have to obey the rules of probability
- The output of a *probability mass function is a probability* whereas *the area under the curve produced by a probability density function represents a probability*.
- Parameters of a probability function play a central role in defining the probabilities of the outcomes of a random event.

Here are some typical distributions



Theoretical Distributions: Uniform

A uniform distribution is pretty much what it sounds like. All outcomes within the bounds of a,b are equally likely. Outside those bounds, the probability is zero. For example, when playing dice, what is the likelihood of getting a particular number of dots (1 in 6).

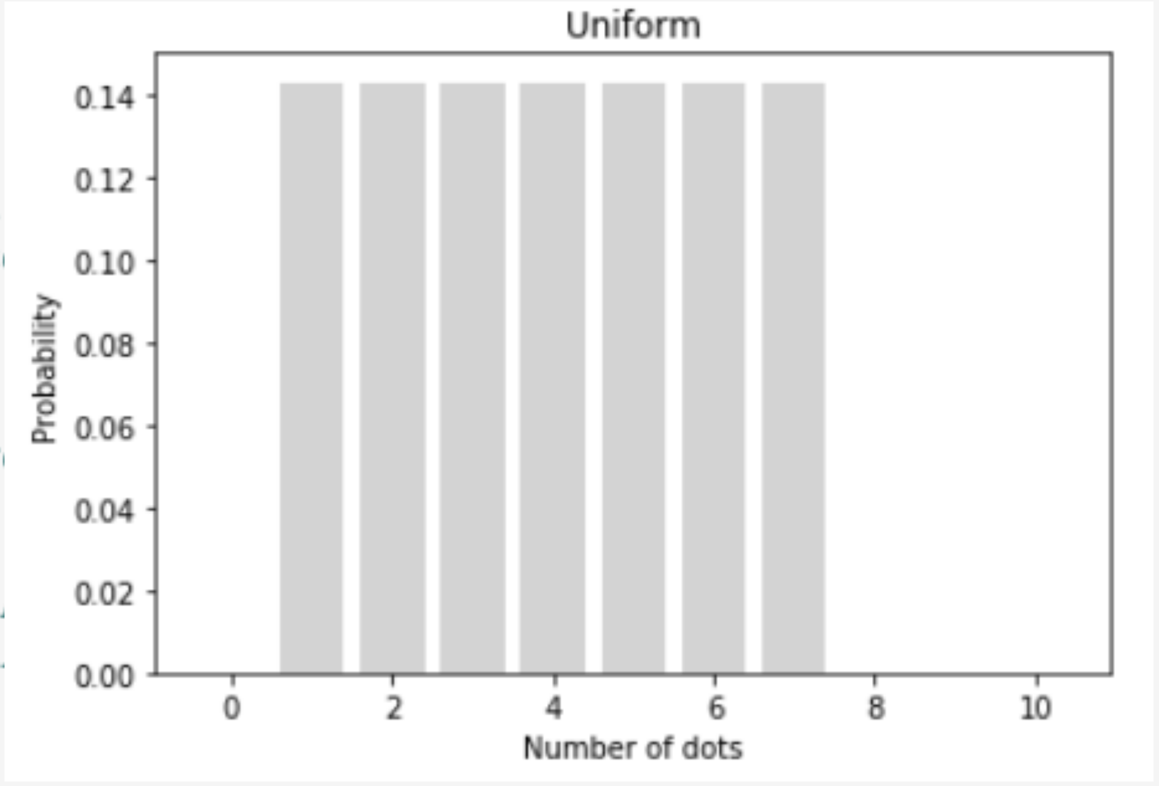
The analytical form for a uniform distribution is:

$$P(x) = f(x, a, b) = \frac{1}{b-a}, a < x < b$$

where a and b are bounds (a < x <b). P(x) is the probability of getting a value of x

Now let's use Python to generate a theoretical Uniform distribution

```python
def Uniform(a,b):
    """
    function for calculating P from a uniform distribution.
    """
    Prob = (1./(b-a))
    return Prob

 # calculate the probability of returning a value x with
xs=range(11) # range of possible number of dots from 1 t
a,b=1,8 # bounds for the uniform distribution
Probability=[] # container for the theoretical results.
for x in xs: # step through test values
    if x not in range(a,b): # if x<a or x>b,  there is z
        Probability.append(0) # save the probability
    else: # otherwise
        Probability.append(Uniform(a,b)) # get the proba
plt.bar(xs,Probability,color='lightgrey') # plot as a ba
plt.title('Uniform')
plt.xlabel('Number of dots')
plt.ylabel('Probability');
```



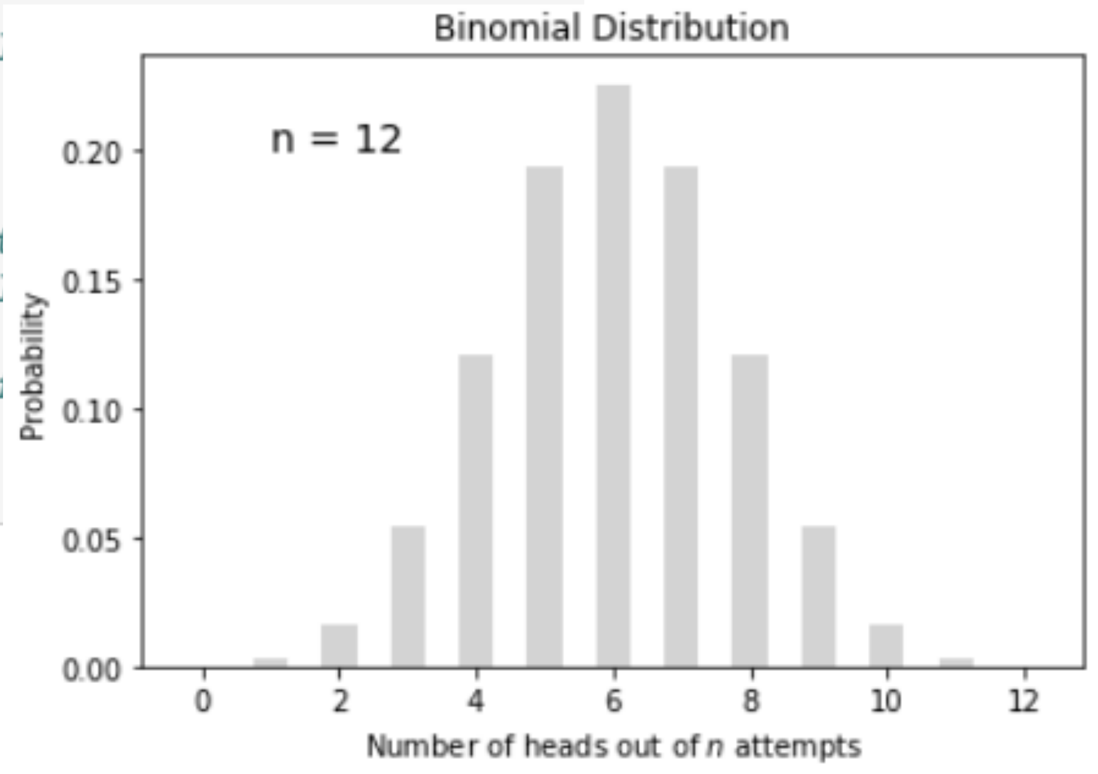
Theoretical Distributions: Binomial

the ***binomial*** distribution which describes the probability of a particular outcome when there are only two possibilities (yes or no, heads or tails, 1 or 0). For example, in a coin toss experiment (heads or tails), if we flip the coin n times, what is the probability of getting x 'heads'? We assume that the probability of a head for any given coin toss is 50%; put another way p = 0.5.

$$P(x) = f(x, n, p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, x = 0,1,2,\ldots,n$$

Now let's use Python to generate a theoretical Binomial distribution

```python
from scipy.special import factorial

def  Binomial(x,n,p):
    """
    Binomial distribution function
    """
    Prob=(factorial(n)/(factorial(x)*factorial(n-x)))*(p**(x))*(1.-p)**(n-x)
    return Prob

n,p = 12,0.5 # number of attempts in each trial, probabi
xs = np.arange(13)
Probability = Binomial(xs,n,p)

plt.bar(xs,Probability,width=.5,color='lightgrey') # plot
plt.xlabel('Number of heads out of $n$ attempts') # add l
plt.ylabel('Probability')
# place a note in upper left in axes coordinates with a f
plt.text(1,.2, 'n = %i'%(n),  fontsize=14)
plt.title('Binomial Distribution');
```



Note: $n! = 1 \times 2 \times 3 \times \cdots \times (n-1) \times n$

Theoretical Distributions: Normal

Probably the most common distribution in real-life data is the so-called "normal" distribution (also known as a Gaussian distribution after the guy who thought it up).

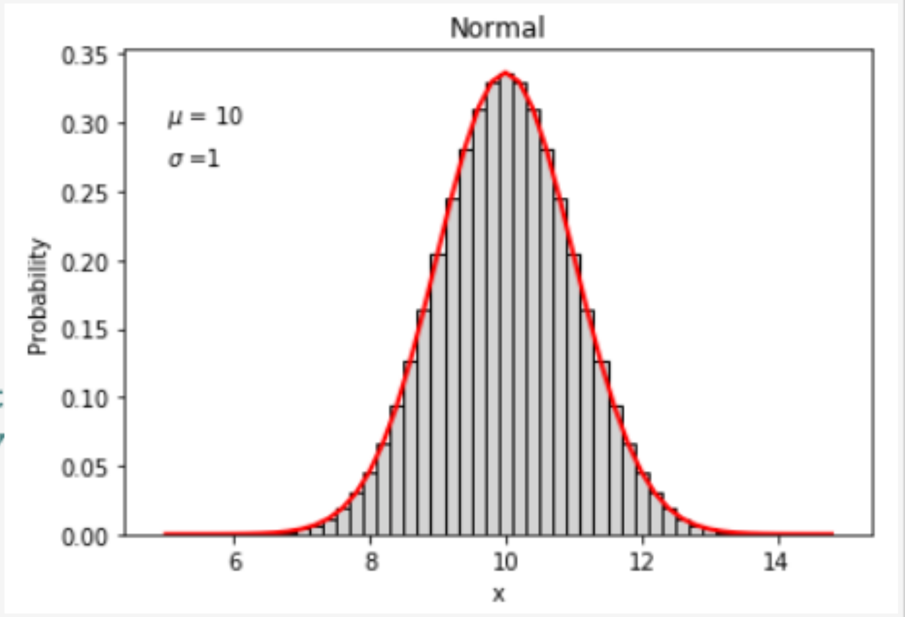
A normal (or Gaussian) distribution describes data, like measurement data, that have uncertainty associated with them - they are more or less precise. There is some 'true' answer but all the measurements have some error.

Imagine measuring the width of a sedimentary bed, or the length of a fossil thigh bone or the distance between two points. The measurement data will have some *average* (a.k.a. *central tendency*) and some degree of *spread*.

For normal distributions, the **average** is the *arithmetic mean* μ and the **spread** is the *standard deviation* σ :

$$P(x) = f(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}, -\infty < x < +\infty$$

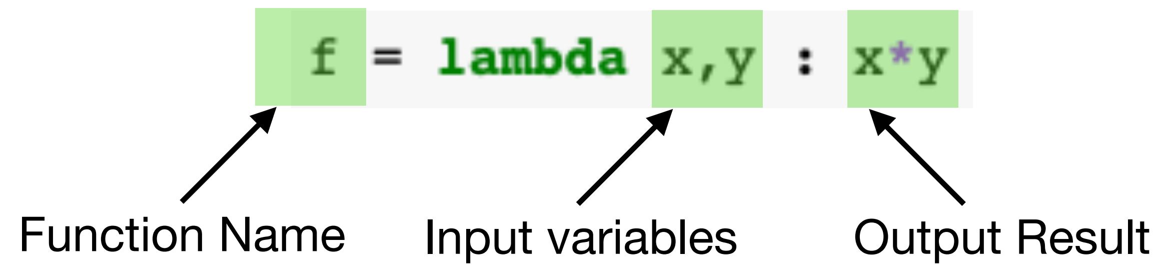
```python
def Normal(x,mu,sigma):
    """
    Normal Distribution Function
    """
    Prob = (1./(sigma*np.sqrt(2.**np.pi)))*np.e**(-(x-mu)**2/(2.*sigma**2))
    return Prob

mu,sigma,incr=10,1,.2 # set the mean,  standard deviation and bin width
xs=np.arange(5,15,incr) # make an array of test values
Probability=Normal(xs,mu,sigma) # get probabilities
plt.bar(xs,Probability,width=incr,color='lightgrey', edgecolor='k') # make t
plt.plot(xs,Probability,'r-',linewidth=2) # plot as a continuous probability
plt.xlabel('x')
plt.ylabel('Probability')
plt.text(5,.3,'$\mu$ = '+str(mu)) # stick on some notes
plt.text(5,.27,'$\sigma$ ='+str(sigma))
plt.title('Normal');
```



Lambda Functions

Syntax:

The syntax of a **lambda** function consists of the word **lambda** followed by an *argument list*, a colon (:), and an *expression*. Here is a simple example of an anonymous function that returns the product of the argument list:



```
f = lambda x,y : x*y
```

Example:

Let’s define a Lambda function for the Binomial distribution function B(x,n,p):

$$P(x) = f(x, n, p) = \frac{n!}{x!(n-x)!} p^x (1-p)^n - x, x = 0,1,2,\ldots,n$$

```
from scipy.special import factorial

Binomial = lambda x,n,p :(factorial(n)/(factorial(x)*factorial(n-x)))*(p**(x))*(1.-p)**(n-x)
```

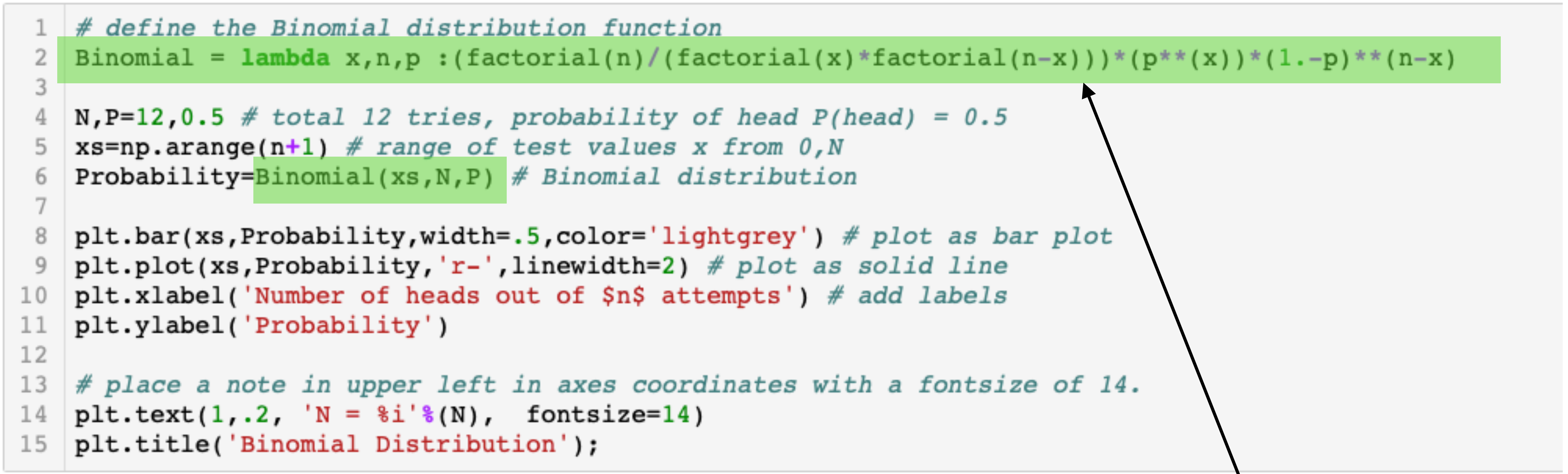
Now if we want to know the probability of getting 8 “heads” in a sequence of 12 coin-flipping (P=0.5):

```
Binomial(8,12,0.5)
```

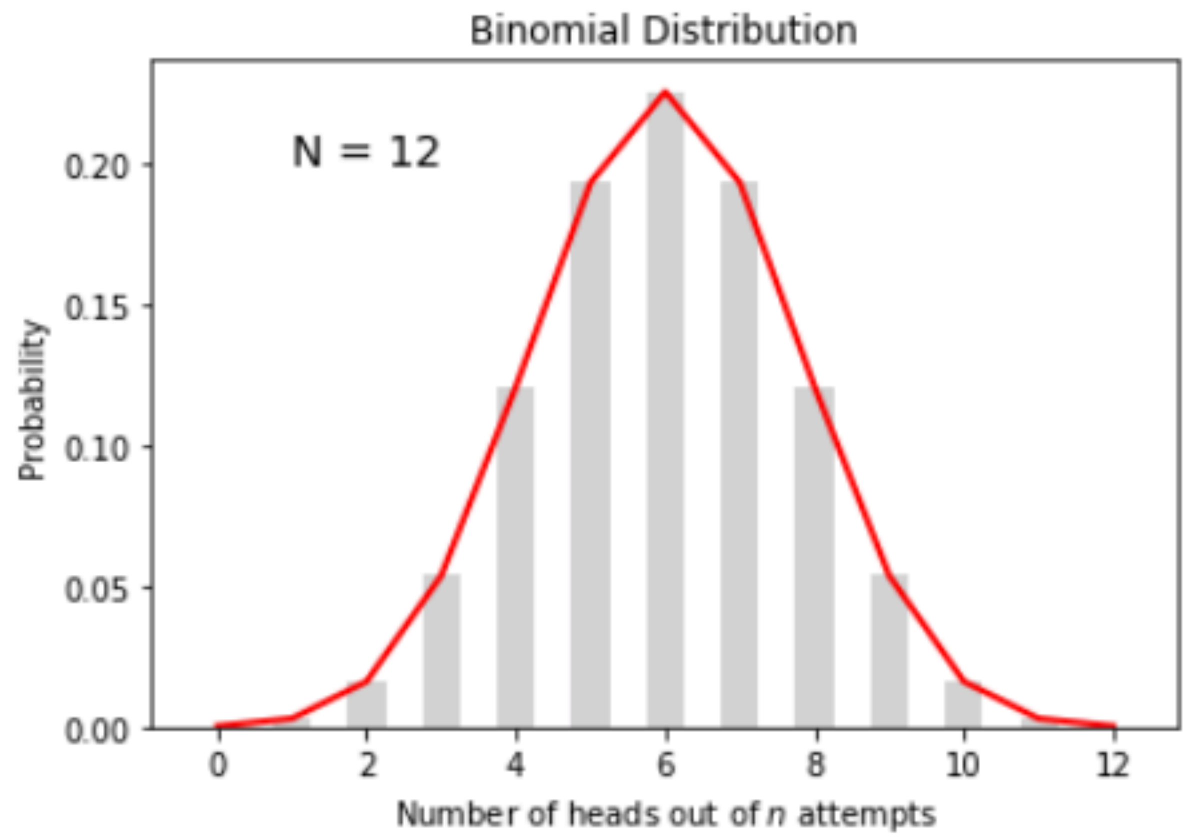
```
0.120849609375
```

Lambda Functions

The Lambda function is useful in generating theoretical distribution functions:



```python
# define the Binomial distribution function
Binomial = lambda x,n,p :(factorial(n)/(factorial(x)*factorial(n-x)))*(p**(x))*(1.-p)**(n-x)

N,P=12,0.5 # total 12 tries, probability of head P(head) = 0.5
xs=np.arange(n+1) # range of test values x from 0,N
Probability=Binomial(xs,N,P) # Binomial distribution

plt.bar(xs,Probability,width=.5,color='lightgrey') # plot as bar plot
plt.plot(xs,Probability,'r-',linewidth=2) # plot as solid line
plt.xlabel('Number of heads out of $n$ attempts') # add labels
plt.ylabel('Probability')

# place a note in upper left in axes coordinates with a fontsize of 14.
plt.text(1,.2, 'N = %i'%(N),  fontsize=14)
plt.title('Binomial Distribution');
```



$$P(x) = f(x,n,p) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}, x = 0,1,2,...,n$$

Empirical Distributions: Binomial

One great feature about computers is that we can simulate a data sample to compare to our theoretical predictions. We can use the module **numpy.random** to generate examples of simulated data sets in a process called ***Monte Carlo simulation***. We encountered **numpy.random( )** in previous lectures when we used the **random.random( )** function. In this lecture we will discover a few more, starting with **random.binomial( )** which generates samples from a *binomial* distribution.

Let's first import all the functions from the **random** module

```
from numpy import random
```

To generate some data, you could either patiently do the experiment with a coin toss, or we can just use the **random.binomial** function to simulate 'realistic' data.

random.binomial( ) requires 2 parameters, n and p with an optional keyword argument **size** (if **size** is not specified, it returns a single trial). Each call to **random.binomial( )** returns the number of heads flipped in a single trial of n coin tosses, given the probability P = 0.5

Let's try n = 12, p = 0.5, This little code block returns the number of heads out of n attempts. You will get a different answer every time you run it. It also gives the probabilty of getting that result from our lambda function **Binomial( )**.

```
n, p = 12,0.5 # same as before
x=random.binomial(n,p,size=1) # size = 1 by default
print (x, 'heads, with a likelihood of: ',Binomial(x,n,p))
```

```
[8] heads, with a likelihood of:  [0.12084961]
```

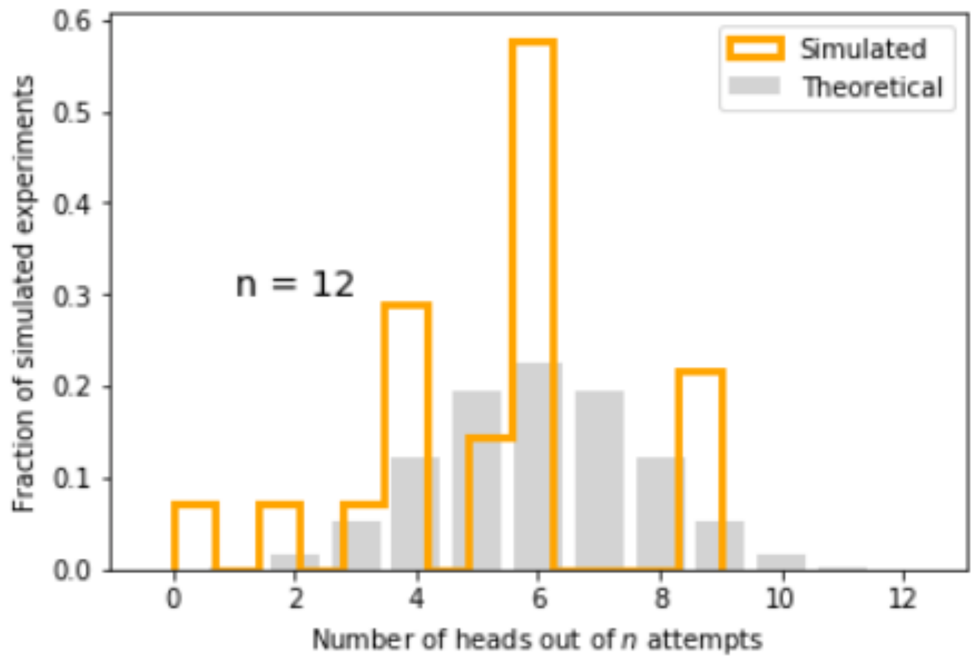
As the number of times you repeat this 'experiment' approaches infinity, the distribution of outcomes will approach the theoretical distribution (i.e., you will get an average of 8 heads out of 12 attempts 12% of the time). This is basically a **Monte Carlo Simulation**

Empirical Distributions: Binomial

Monte Carlo Simulations

So let's compare the results simulated via Monte Carlo for some number of experiments (Nmc) with the theoretical distribution. To do this, we pretend that each student in the class (Nmc=20) flips a coin *n*=12 times and reports the number of heads. We can collect the number of heads flipped by each student in a list called **Simulated**.

```python
from numpy import random
# Theoretical Binomial distribution
Binomial = lambda x,n,p :(factorial(n)/(factorial(x)*factorial(n-x)))*(p**(x))*(1.-p)**(n-x)

n,p = 12,0.5 # same as before
xs=np.arange(n+1) # range of test values x from 0,N
Probability=Binomial(xs,n,p) # Theoretical Binomial distribution

Nmc=20 # number of simulated experiments each with n attempts
Simulated=random.binomial(n,p,size=Nmc) # simulating the coin-flipping using the binomial() function
plt.bar(xs,Probability,color='lightgrey', label='Theoretical') # theoretical curve as bar graph
plt.hist(Simulated,density=True,color='orange',histtype='step',linewidth=3, label='Simulated',bins=13) # note the no
                  #------------#  this option normalizes the total to be unity

plt.xlabel('Number of heads out of $n$ attempts')
plt.ylabel('Fraction of simulated experiments')
plt.text(1,.3, 'n = %i'%(n),  fontsize=14)
plt.legend();
```



```python
print(Simulated)
```

```
[6 6 5 9 4 6 5 4 2 6 9 0 4 9 6 4 6 6 3 6]
```

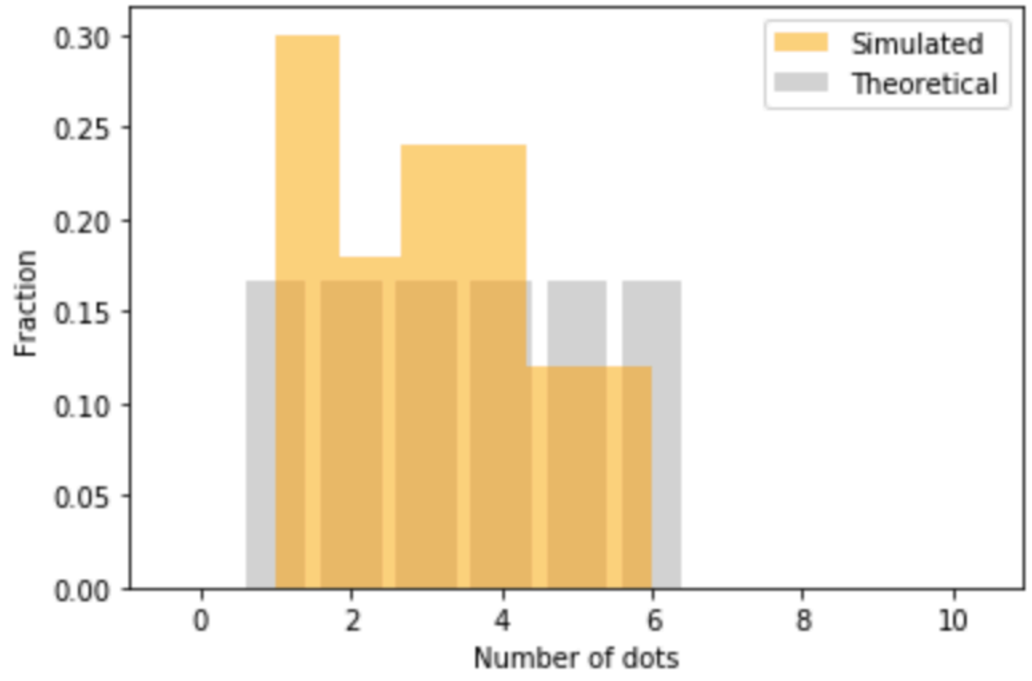
1. The orange distribution does not look like a Binomial distribution? What's the problem here?
2. According to the **NumPy** documentation for **density**: If it is set to 'True', the result is the value of the probability density function at the bin, normalized such that the integral over the range is 1. The sum of the histogram values will not equal 1 unless bins of unity width are used (that's why **bins = 13** is used here).

Empirical Distributions: Uniform

And now we can look at the Monte Carlo simulation of an empirical distribution using **random.uniform( )**. So here it is.

```python
a,b=1,7 # keep the same bounds
Nmc=20 # number of "students" rolling the dice
Simulated=random.uniform(a,b,Nmc).astype('int') # get Nmc test values in one go.  :)
# the .astype(int) makes this an array of integers, as only integers are possible outcomes

# plot the theoretical uniform distribution between 1 and 7
plt.bar(xs,Probability, color='lightgrey',label='Theoretical')
# plot results as histogram normed to sum to unity and use histtype of 'step' to make it see-through
plt.hist(Simulated,density=True,histtype='bar',alpha = 0.5,color='orange',linewidth=3.,label='Simulated',bins=6)
plt.xlabel('Number of dots')
plt.ylabel('Fraction')
plt.legend();

# print the simulated values
print(Simulated)
```

```
[1 5 2 6 4 1 3 1 2 4 2 6 1 1 3 4 5 3 3 4]
```



Again, the simulated distribution (orange bars) does not look like a Uniform distribution? What's the problem here?