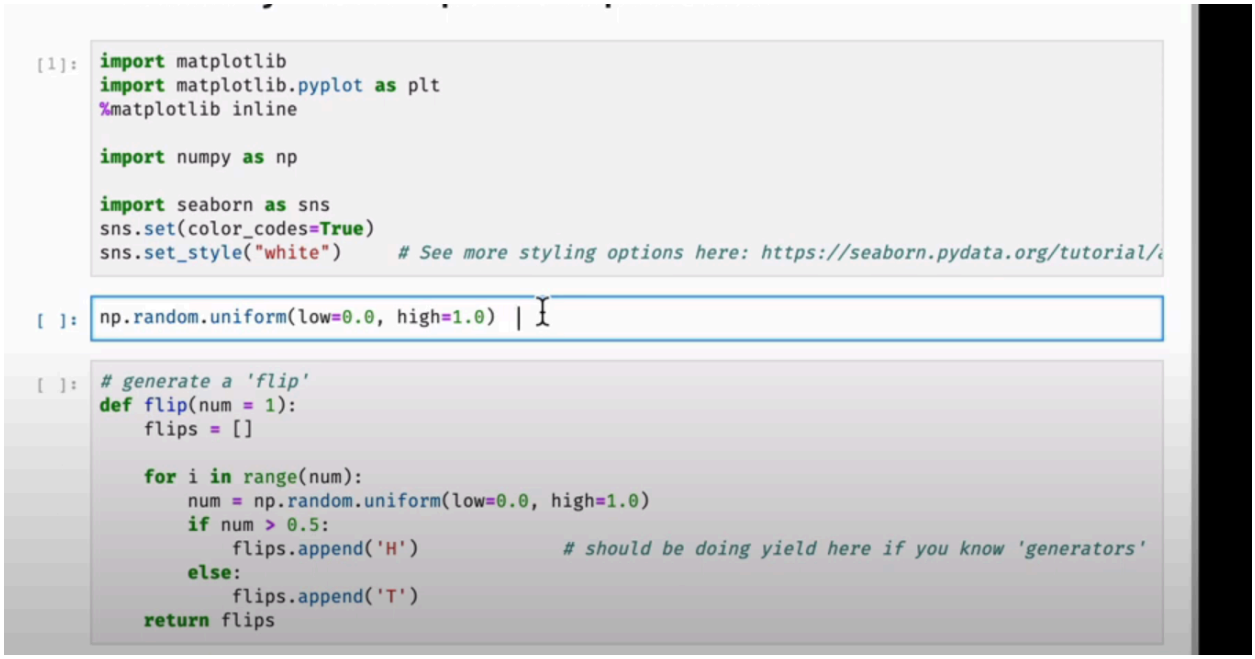
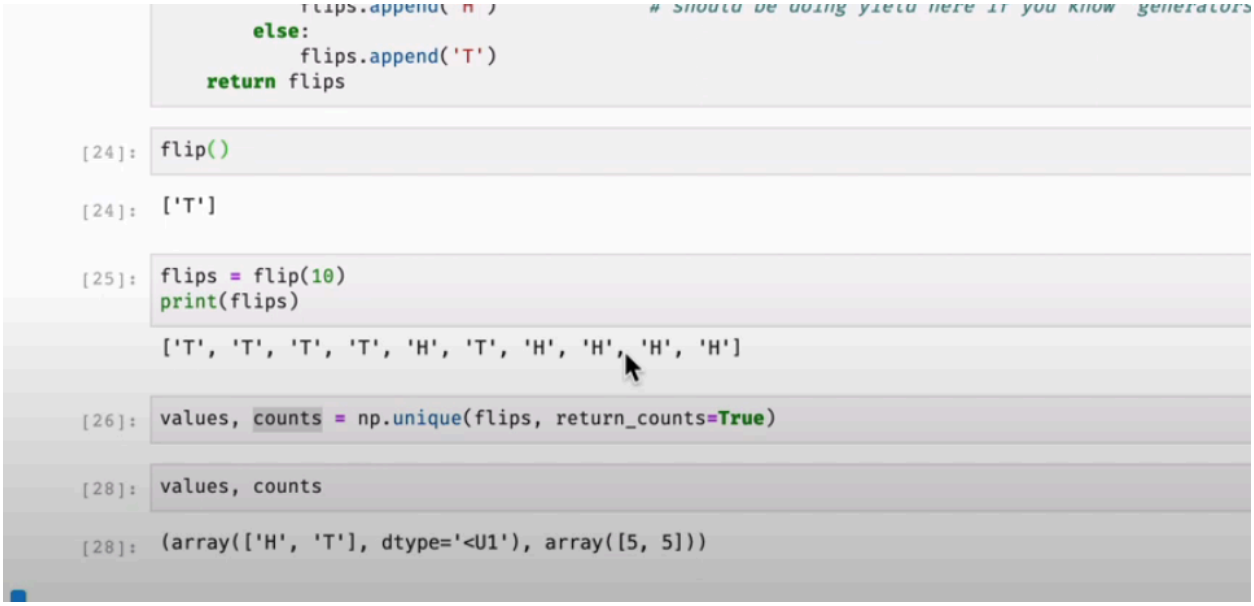
Probability and Statistics with PYTHON lecture 9

- 

```
[1]: import matplotlib
     import matplotlib.pyplot as plt
     %matplotlib inline

     import numpy as np

     import seaborn as sns
     sns.set(color_codes=True)
     sns.set_style("white")     # See more styling options here: https://seaborn.pydata.org/tutorial/a

[ ]: np.random.uniform(low=0.0, high=1.0)

[ ]: # generate a 'flip'
     def flip(num = 1):
         flips = []

         for i in range(num):
             num = np.random.uniform(low=0.0, high=1.0)
             if num > 0.5:
                 flips.append('H')             # should be doing yield here if you know 'generators'
             else:
                 flips.append('T')
         return flips
```

- We start off with generating a random number from 0 to 1
- We want to find the probability of a coin that is tossed if it will be a heads or a tails
- We write a function into it to turn it into a coin flip
    - If random number is greater than 0.5 then take it as heads
    - If random number is lesser than 0.5 then take it as tails



```
            flips.append('H')
        else:
            flips.append('T')
    return flips
```

```
[24]: flip()
```

```
[24]: ['T']
```

```
[25]: flips = flip(10)
      print(flips)
```

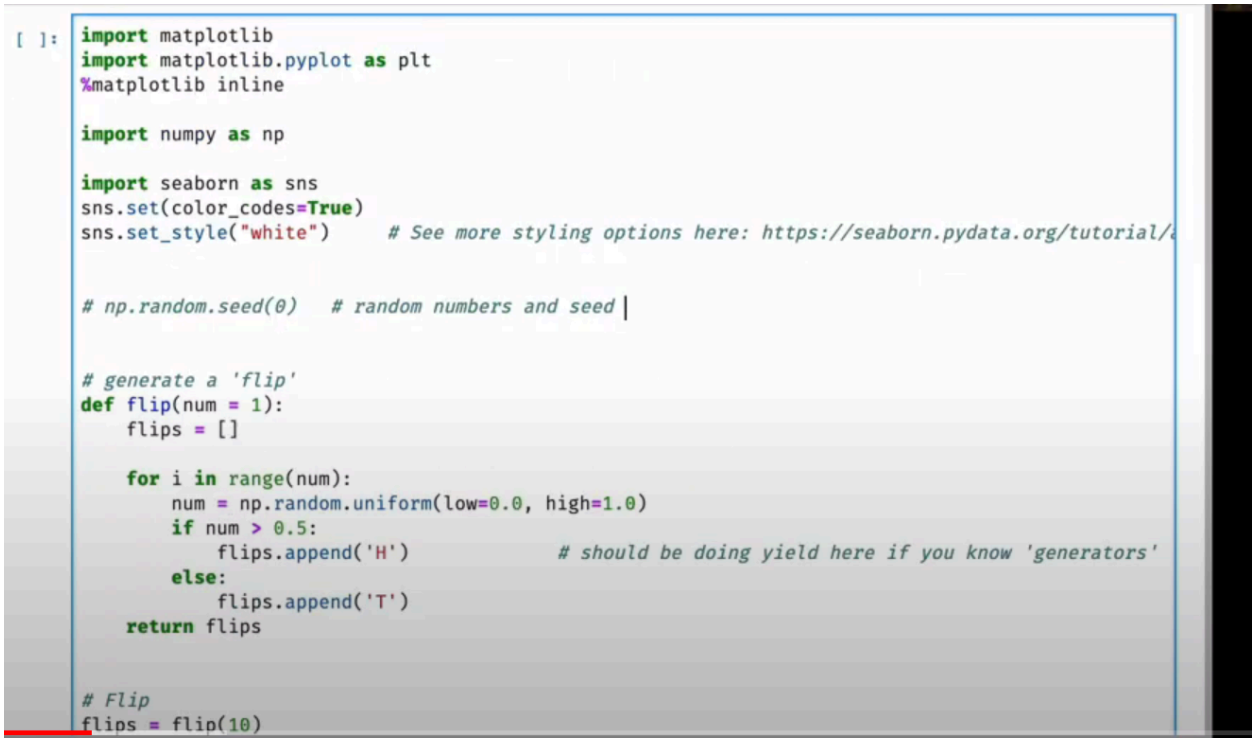
```
['T', 'T', 'T', 'T', 'H', 'T', 'H', 'H', 'H', 'H']
```

```
[26]: values, counts = np.unique(flips, return_counts=True)
```

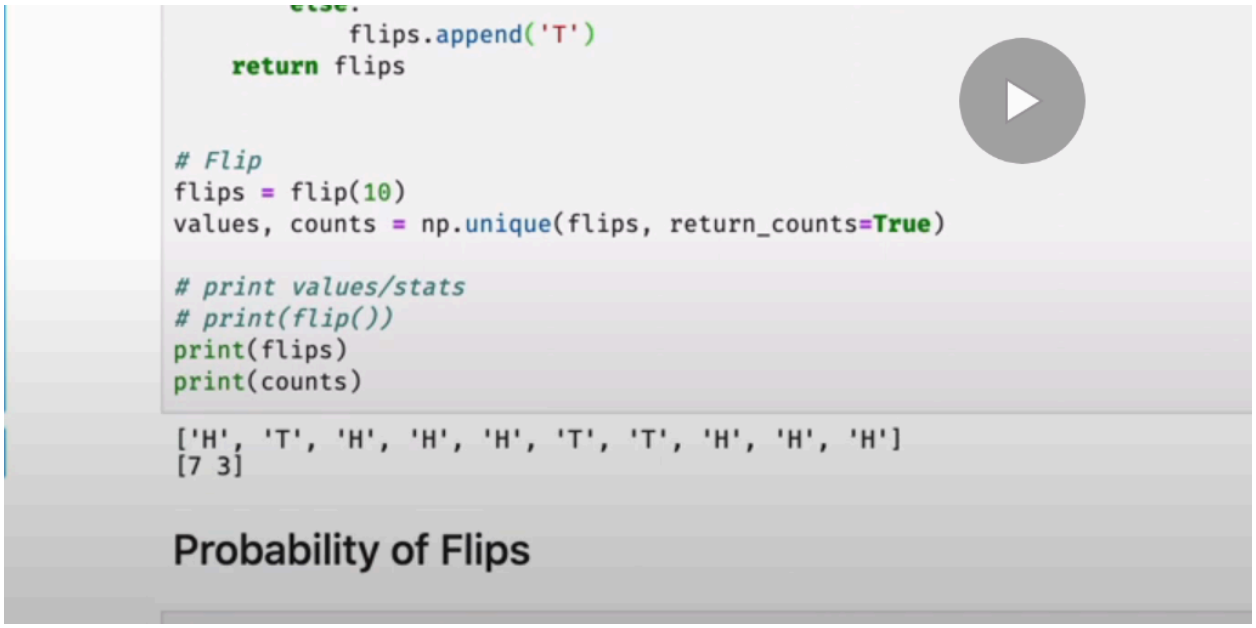
```
[28]: values, counts
```

```
[28]: (array(['H', 'T'], dtype='<U1'), array([5, 5]))
```

- Now we wanna count the number of head and tails produced in 10 flips
- We can do it manually but if the data is large in thousands or millions we won't be able to, which is why we are gonna use a “ **.unique** “ function of numpy to take the result of the flip and the number of times it appeared and put it in a variable. Basically two different array with different values and counts
- This values and counts have tuple of what values we have like here we have “ HEADS “ and “ TAILS “ , and the counts tell us how many such values we have

- 

```
[ ]:
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline

import numpy as np

import seaborn as sns
sns.set(color_codes=True)
sns.set_style("white")     # See more styling options here: https://seaborn.pydata.org/tutorial/

# np.random.seed(0)   # random numbers and seed

# generate a 'flip'
def flip(num = 1):
    flips = []

    for i in range(num):
        num = np.random.uniform(low=0.0, high=1.0)
        if num > 0.5:
            flips.append('H')             # should be doing yield here if you know 'generators'
        else:
            flips.append('T')
    return flips

# Flip
flips = flip(10)
```

- Now we have put the above all code in a separate block/cell
- 

```
            flips.append('T')
    return flips

# Flip
flips = flip(10)
values, counts = np.unique(flips, return_counts=True)

# print values/stats
# print(flip())
print(flips)
print(counts)
```

```
['H', 'T', 'H', 'H', 'H', 'T', 'T', 'H', 'H', 'H']
[7 3]
```

Probability of Flips

- There is a problem here called as " HEISENBUGS "
- The problem is if we have an error with success of lets say 5 heads 5 tails and the next time we run it and it produces a success of lets say 7 heads and 3 tails we won't be able to find the bug in [5 ,5] because it produces a different output every time
- Because it is by chance and we cannot identify the problem because we don't know if it will appear next time by chance here in the small data set we can find the pattern but in

lets say millions of data we won't be able to find the chance which had problems

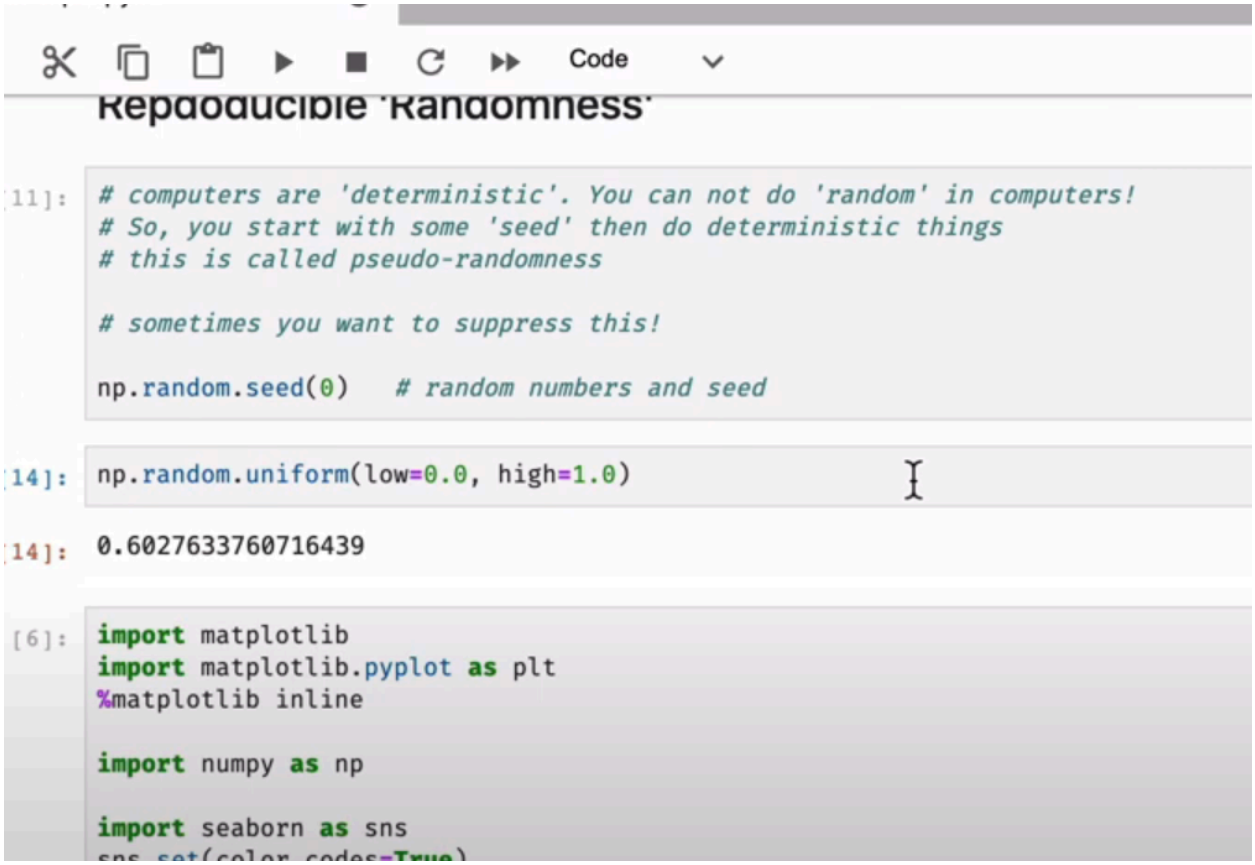
- We want reproducible randomness , so that i know the next time i produce something random , and i can predict it
- RANDOMNESS :
    - A computer cannot generate randomness because it does not have the capability to generate randomness , because it can only execute specific instructions
    - The computer will listen to noise around it and catch a frequency (from real world surrounding of the computer) the computer will perform different operations on it , and get a random number from it this is called PSEUDO RANDOMNESS
    - The first number that the computer gets is called a **SEED** . also the rest generated numbers won't be distributed , it will have one peak
    - We are gonna take the seed and then make a random number out of it because it has come from the real world

```
[ ]: # computers are 'deterministic'. You can not do 'random' in computers!
     # So, you start with some 'seed' then do deterministic things
     # this is called pseudo-randomness

     # sometimes you want to suppress this!

     np.random.seed(0)   # random numbers and seed
```

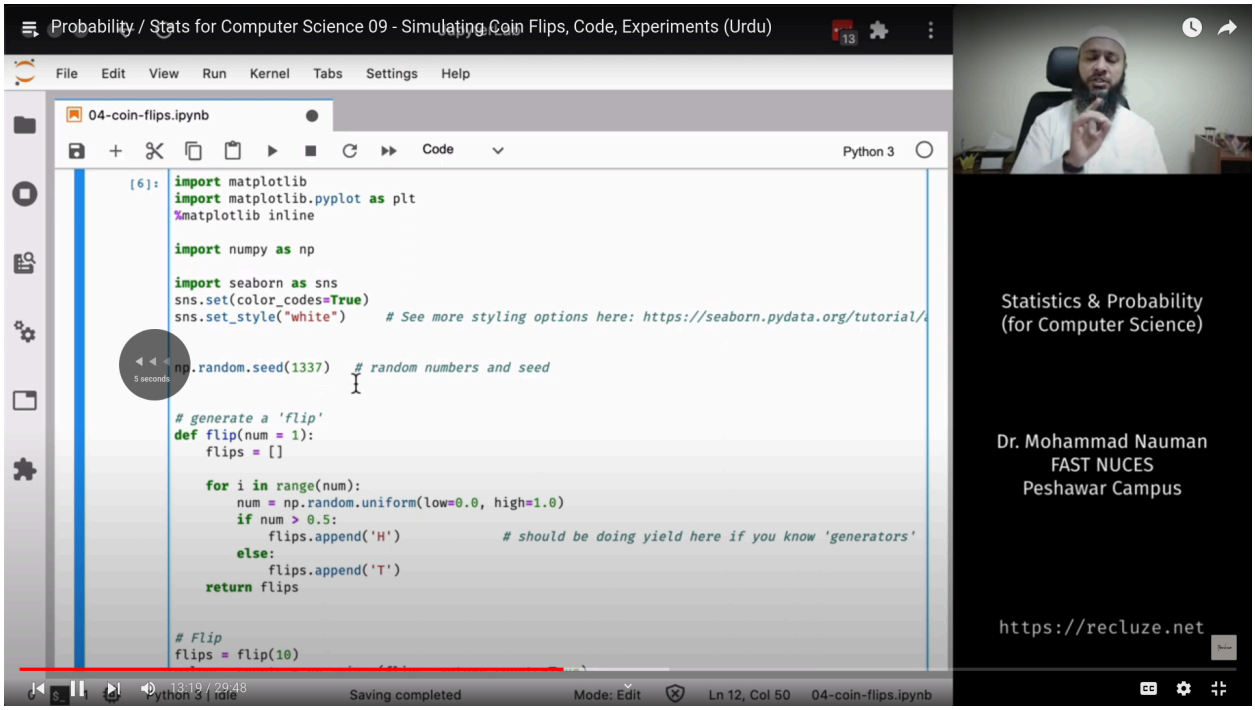
```
[6]: import matplotlib
     import matplotlib.pyplot as plt
     %matplotlib inline

     import numpy as np

     import seaborn as sns
     sns.set(color_codes=True)
     sns.set_style("white")     # See more styling options here: https://seaborn.pyd


     # np.random.seed(0)   # random numbers and seed


     # generate a 'flip'
     def flip(num = 1):
         flips = []
```

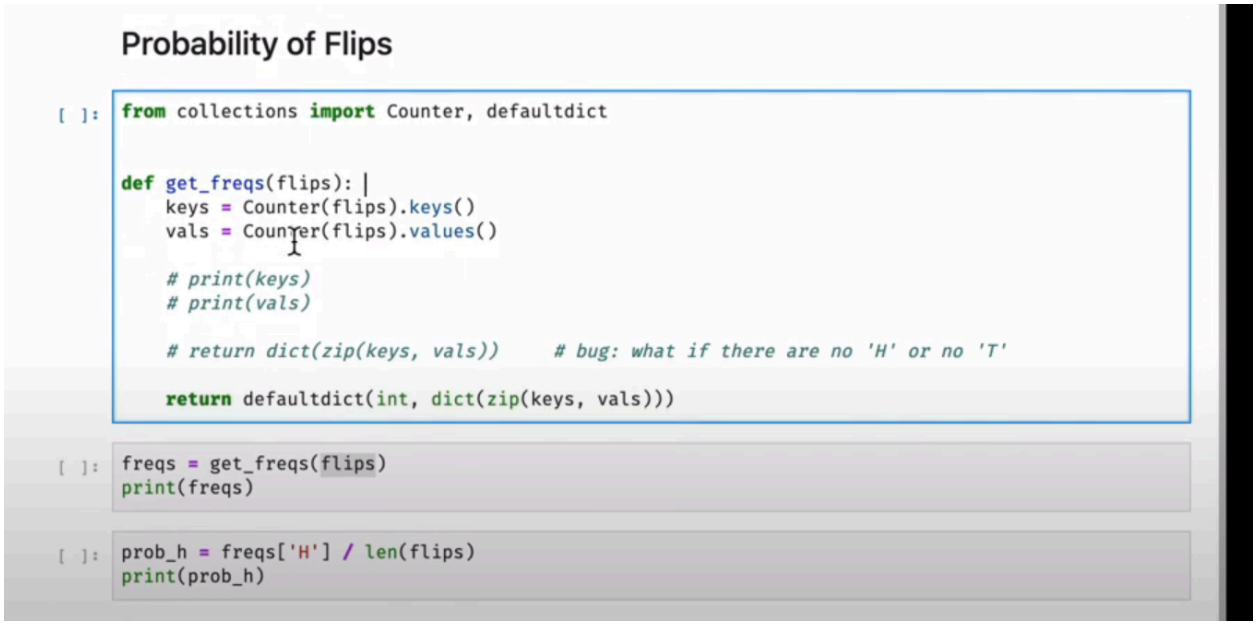
- We are not gonna take random number seed from real world because it will always gives us some random new seed instead we put the seed to 0

- “ np.random.seed(0)” will generate a number and make it a seed everytime it generates a number gives it to us and makes it as a seed so if you dont change the code it will be reproducible in the same order it produced the outputs



- It produces the random number and saves its order so it will be generate numbers in the same pattern but if we make the seed 0 then it will generate the number again from starting in the same pattern

- 

- Always put seed 0 or anynumber so that you have reproducible outputs
- 

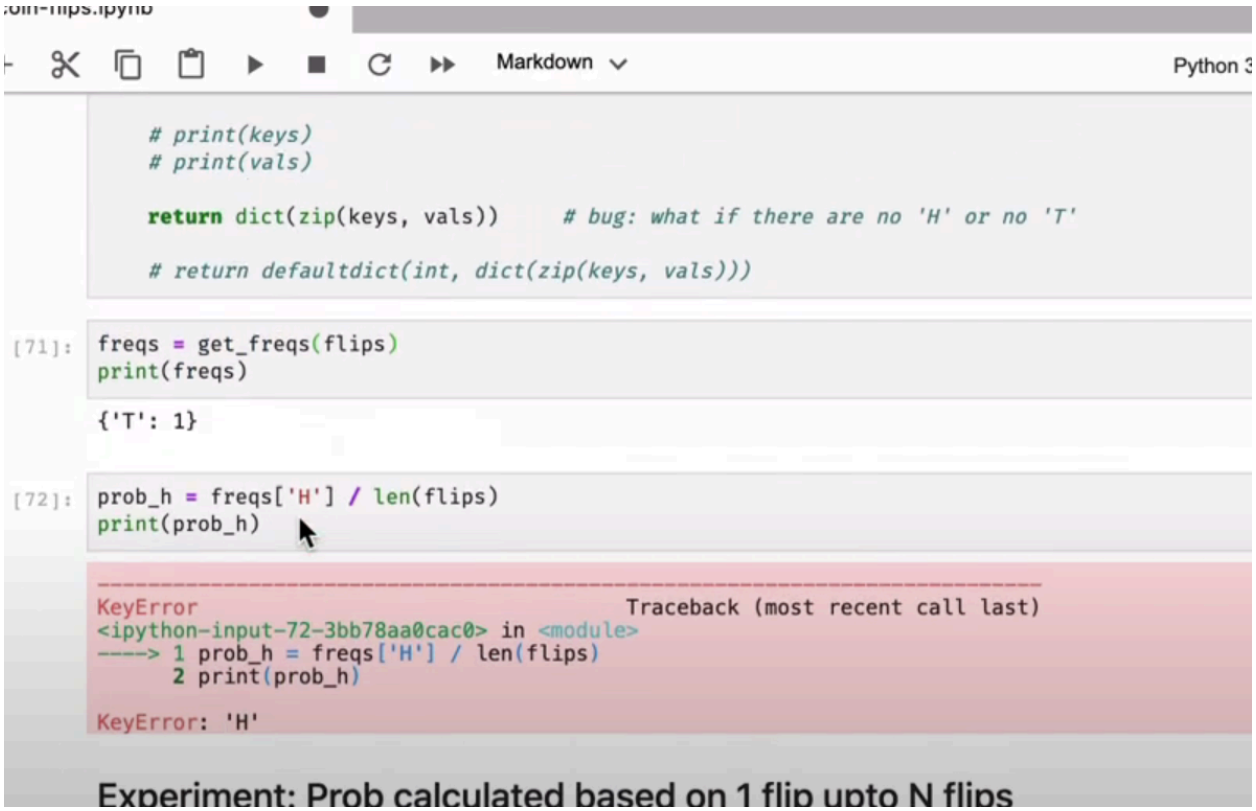
- Making the key value pair of the values and its frequency
- We don't use e “ dict(zip(keys,vals)) “ because there is an error in it , what if there is no tails in the data it will generate error when we coun tails frequency (basic python according to sir) so if there is no tails it will give key error

- Alternate we use “ defaultDict “

```
[60]: from collections import Counter, defaultdict

      def get_freqs(flips):
          keys = Counter(flips).keys()
          vals = Counter(flips).values()

          # print(keys)
          # print(vals)

          # return dict(zip(keys, vals))     # bug: what if there are no 'H' or no 'T'

          return defaultdict(int, dict(zip(keys, vals)))

[61]: freqs = get_freqs(flips)
      print(freqs)

      defaultdict(<class 'int'>, {'T': 7, 'H': 3})

[ ]:  prob_h = freqs['H'] / len(flips)
      print(prob_h)
```

16:37 / 29:48

- So now we use default dict , it will return the value of the parameter in this case it is int so it will return 0 if no tails or heads is produced



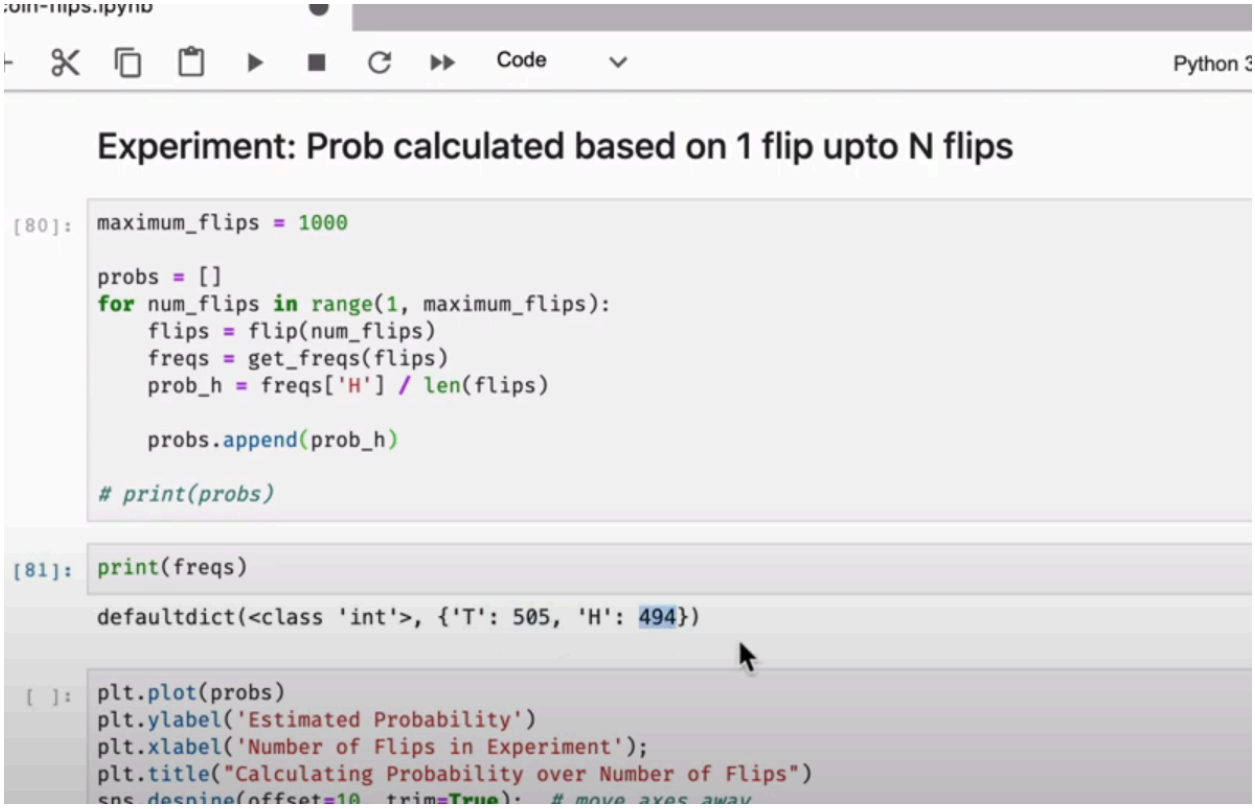
```
Markdown                                                    Python 3

    # print(keys)
    # print(vals)

    return dict(zip(keys, vals))     # bug: what if there are no 'H' or no 'T'

    # return defaultdict(int, dict(zip(keys, vals)))

[71]: freqs = get_freqs(flips)
      print(freqs)

      {'T': 1}

[72]: prob_h = freqs['H'] / len(flips)
      print(prob_h)

      ---------------------------------------------------------------------------
      KeyError                                  Traceback (most recent call last)
      <ipython-input-72-3bb78aa0cac0> in <module>
      ----> 1 prob_h = freqs['H'] / len(flips)
            2 print(prob_h)

      KeyError: 'H'
```

- Experiment: Prob calculated based on 1 flip upto N flips

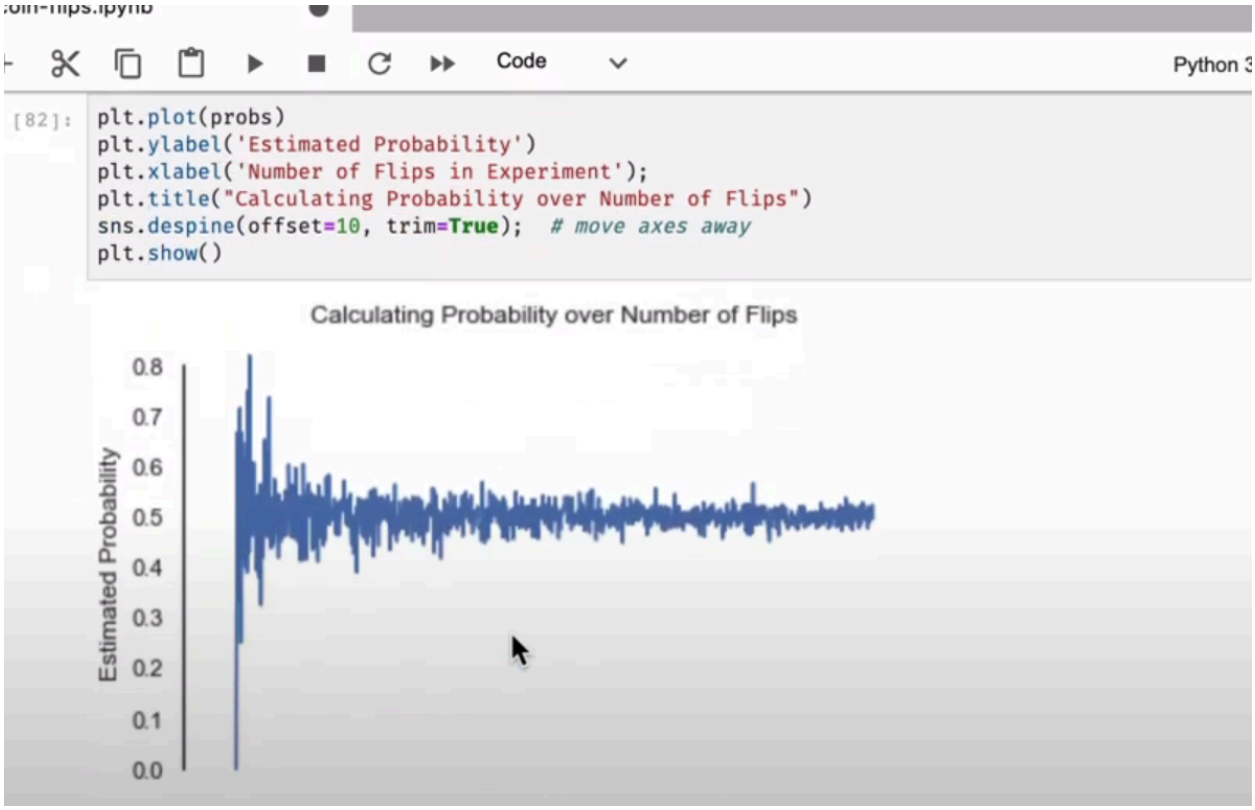
- As you can see above it gives error

```
[62]: flips = ['H']

[63]: from collections import Counter, defaultdict


      def get_freqs(flips):
          keys = Counter(flips).keys()
          vals = Counter(flips).values()

          # print(keys)
          # print(vals)

          # return dict(zip(keys, vals))     # bug: what if there are no 'H' or no 'T'

          return defaultdict(int, dict(zip(keys, vals)))

[64]: freqs = get_freqs(flips)
      print(freqs)

      defaultdict(<class 'int'>, {'H': 1})

[65]: prob_h = freqs['H'] / len(flips)
      print(prob_h)

      1.0
```

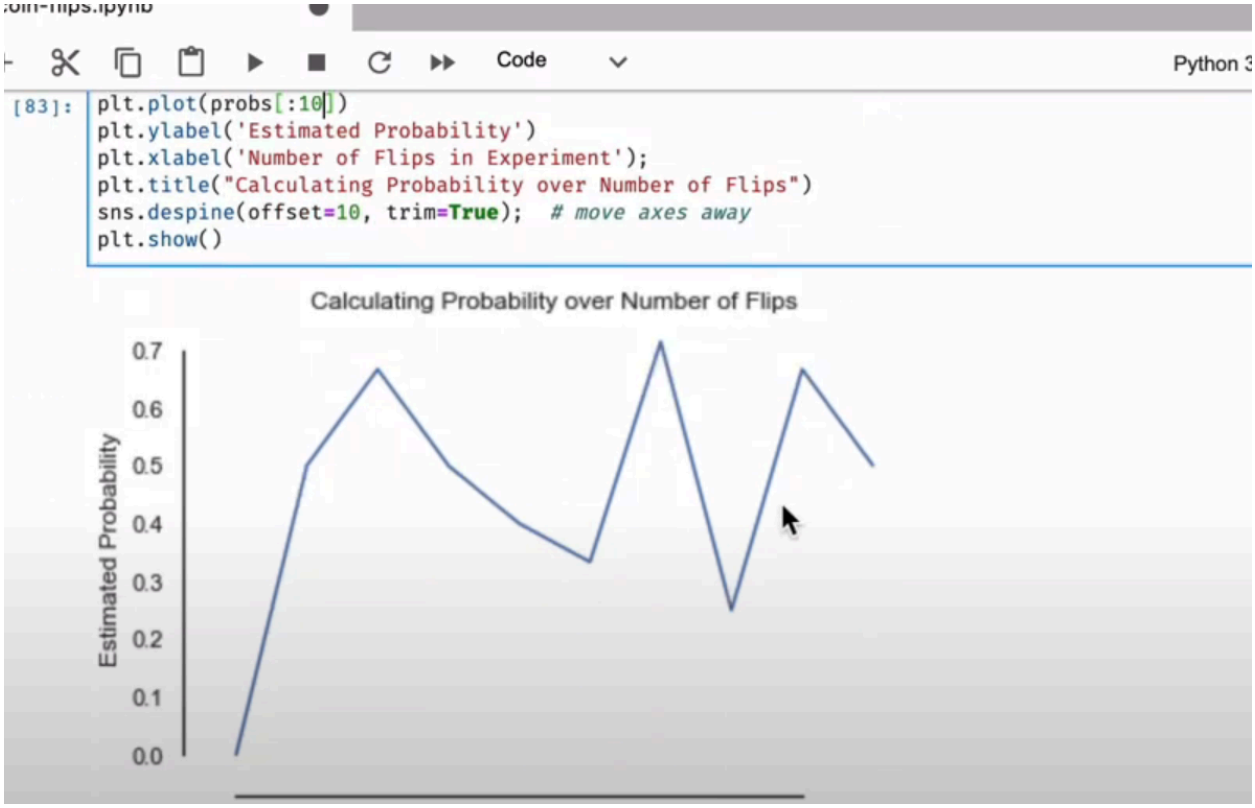
-
- See it works now



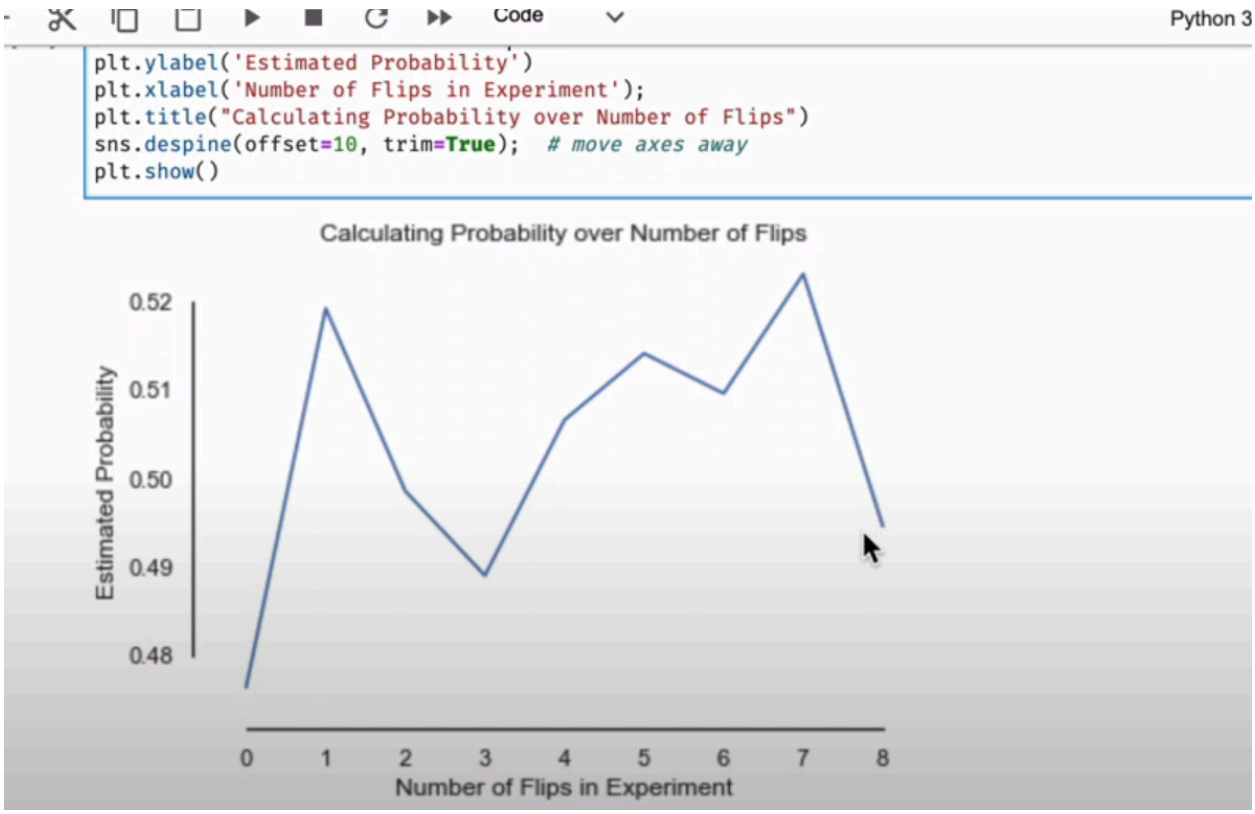
- Now we flip it 1000 times and save its probability and its frequency



- As you can see the graph for it for the frequency

- 

- Plotting values of first 10 probabilities in the above graph

- 

- Above is the plot for last 10 probabilities

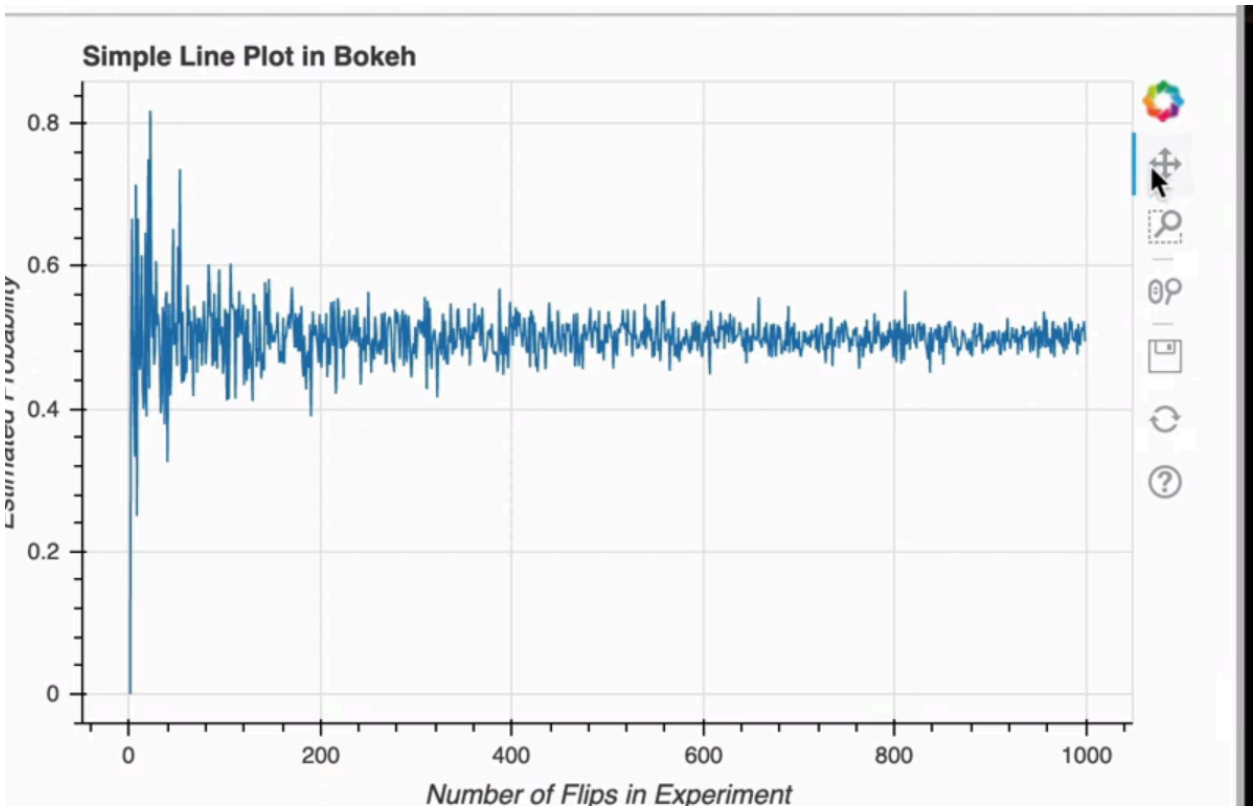
- We can never get to 0.5 in this experiment

- BOKEH FOR INTERACTIVE PLOTS

-

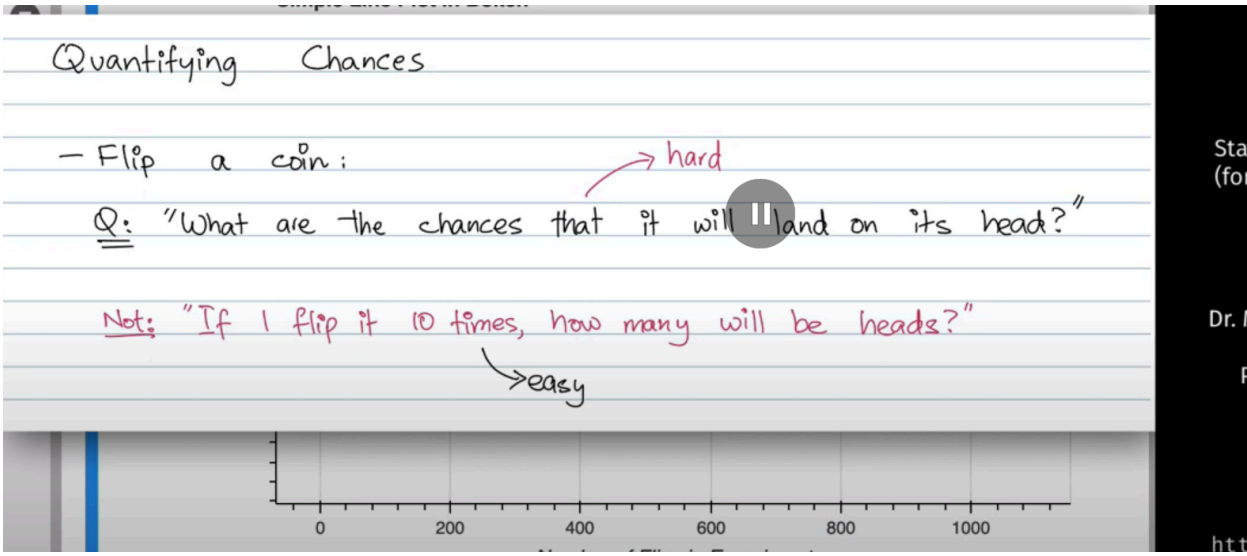
```
[ ]: !pip install bokeh

[ ]: from bokeh.io import show, output_notebook
     from bokeh.plotting import figure

     output_notebook()

[ ]: p = figure(title="Simple Line Plot in Bokeh",
                x_axis_label='Number of Flips in Experiment',
                y_axis_label='Estimated Probability',
                plot_width=580, plot_height=380)

[ ]: # Add a line renderer with legend and line thickness
     x = range(1, maximum_flips)
     p.line(x=x, y=probs)

     # Show the results
     show(p)

[ ]:
```

- This is how we print it



- Gives you more options you can zoom , pan and check values at every frequency

- OVERVIEW
    - We were looking for chances
    - We can give different weights of heads and tails
    - The points of flipping it so many times is to understand what is the behaviour of this coin on flips
    - Everytime we do a n-flip , we get a different probability



    - We did this experiment for the second question , on what TH flip we will get heads
    - These are still experiments , this is not necessary they will give us real world information
    - There are some things we cannot experiment more than one time
        - For example
        - Who is going to win the next election
            - We have to predict only one election before it happening
            - We cannot have its probability based on number of experiments
        - What team will win the tournament
            - We cannot predict its probability on different experiments because it will happen once
            - We have to decide on prior / past knowledge
    - Prior / past knowledge is a huge part of machine learning , data science
    - Where do you come up with the initial value of 0.5 ?
        - Do you run experiments for it?
        - Or do you have some kind of a past knowledge

-