Lab 1 : Probability Theory

1. Sampling from uniform distribution
2. Sampling from Gaussian distribution
3. Sampling from categorical distribution through uniform distribution
4. Central limit theoram
5. Law of large number
6. Area and circumference of a circle using sampling
7. Fun Problem

There are missing fields in the code that you need to fill to get the results but note that you can write you own code to obtain the results

1.Sampling from uniform distribution

a) Generate N points from a uniform distribution range from [0 1]

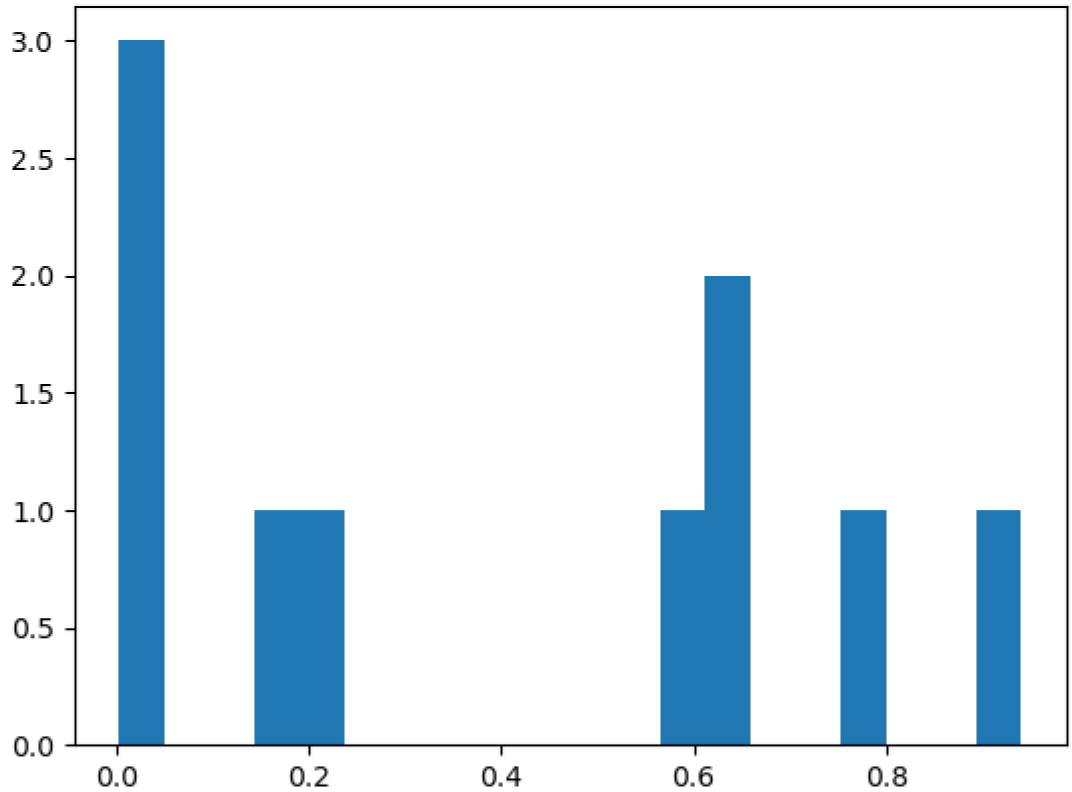
In [1]:

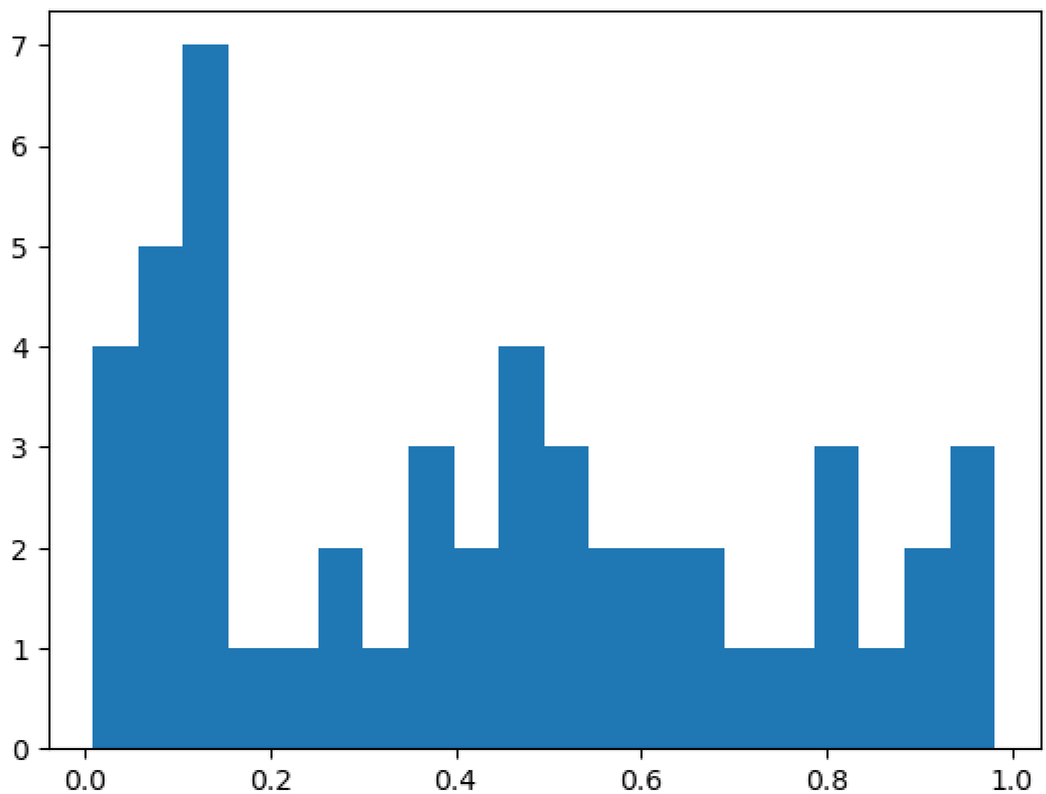
```
import numpy as np
import matplotlib.pyplot as plt

N =  10#  Number of points (Example = 10)
X =  np.random.uniform(0,1,N)#  Generate N points from a uniform distribution range from 
```

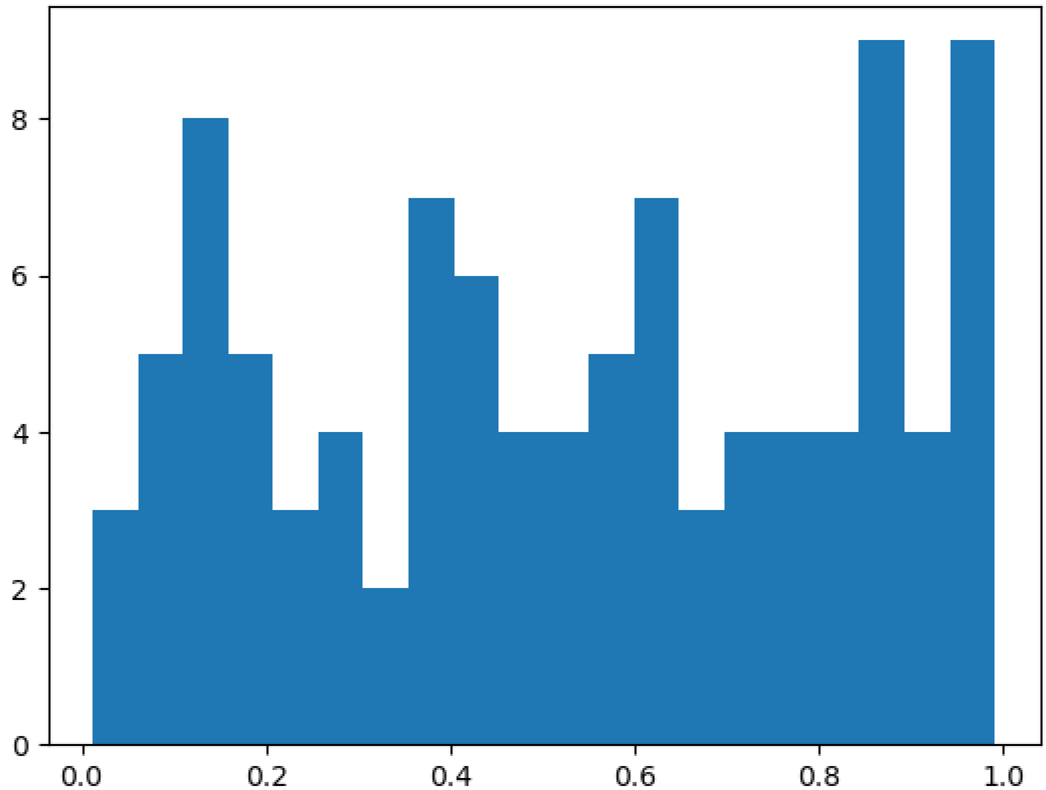
b) Show with respect to no. of sample, how the sampled distribution converges to parent distribution.

In [2]:

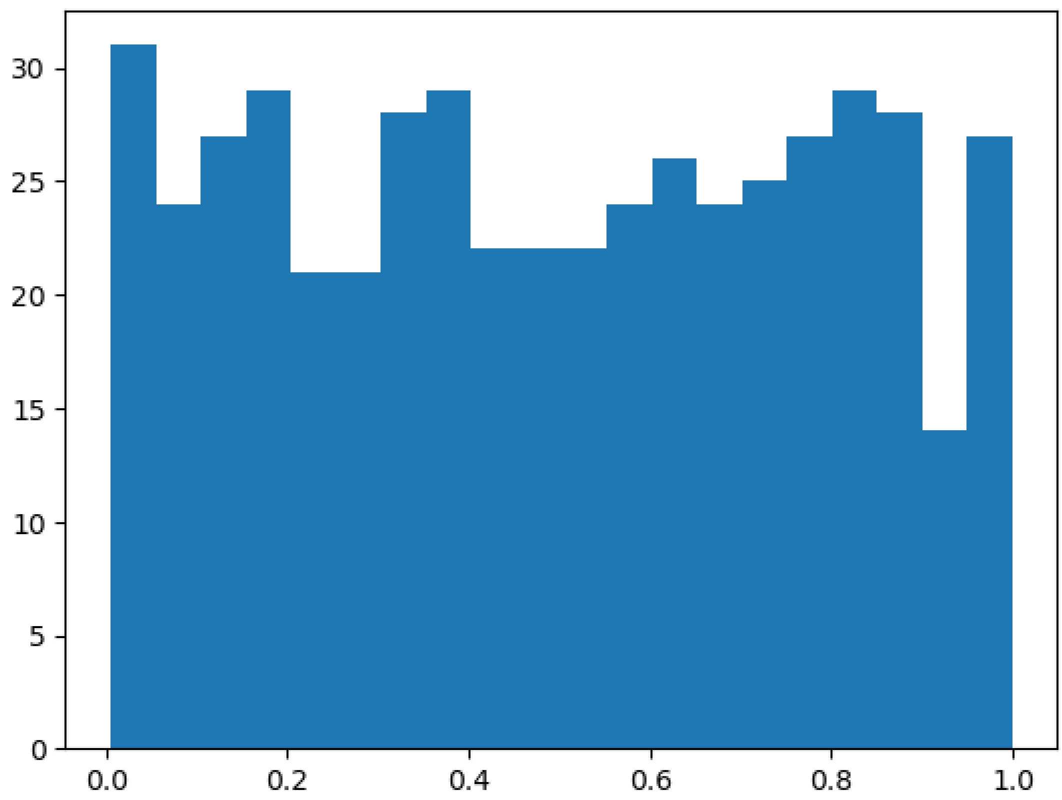
```
arr =  np.array([10,50,100,500,1000,5000,10000,50000,100000])# Create a numpy array of di

for i in arr:
    x =  np.random.uniform(0,1,i)# Generate i points from a uniform distribution range fro
    plt.hist(x,bins=20)
    plt.show()
  # write the code to plot the histogram of the samples for all values in arr # Ref : http
```


3.0
2.5
2.0
1.5
1.0
0.5
0.0
0.0
0.2
0.4
0.6
0.8

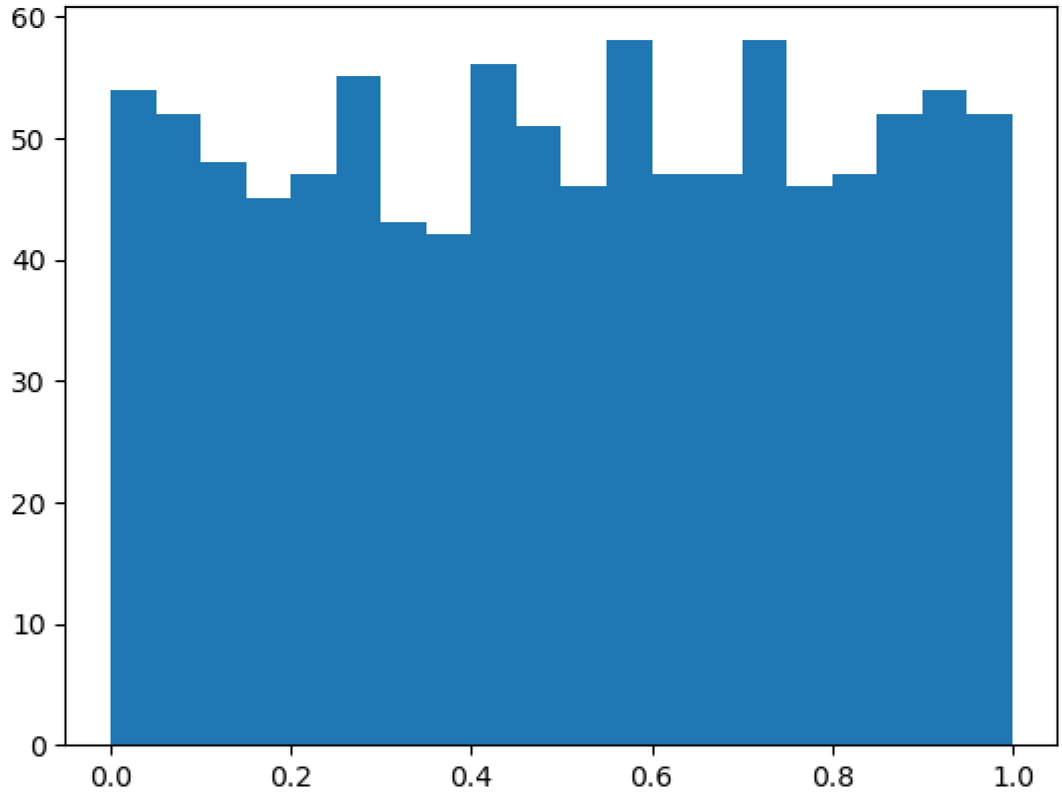

7
6
5
4
3
2
1
0
0.0
0.2
0.4
0.6
0.8
1.0

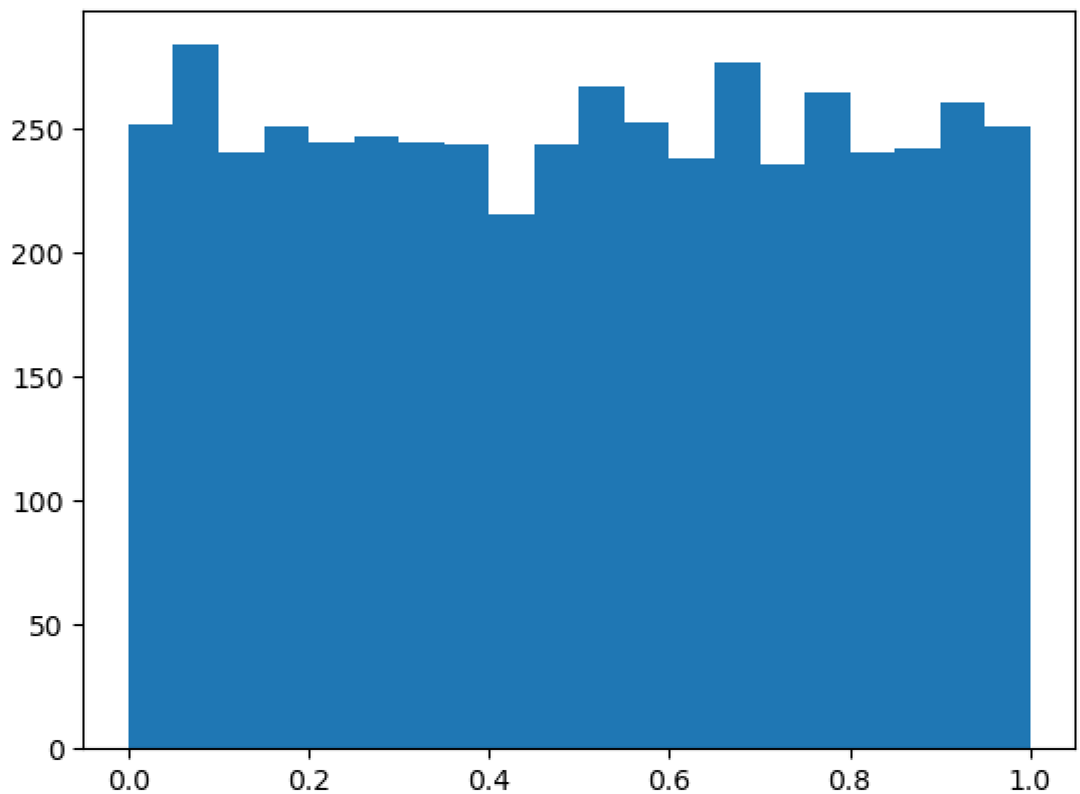


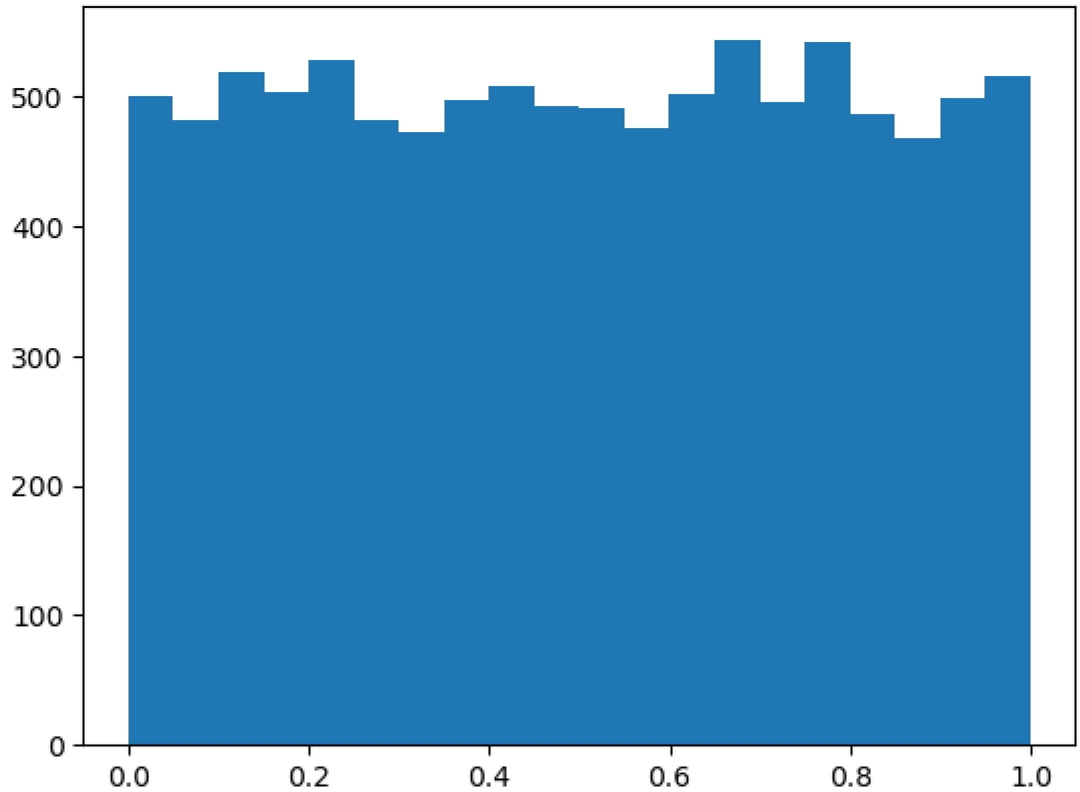
0
2
4
6
8
0.0
0.2
0.4
0.6
0.8
1.0

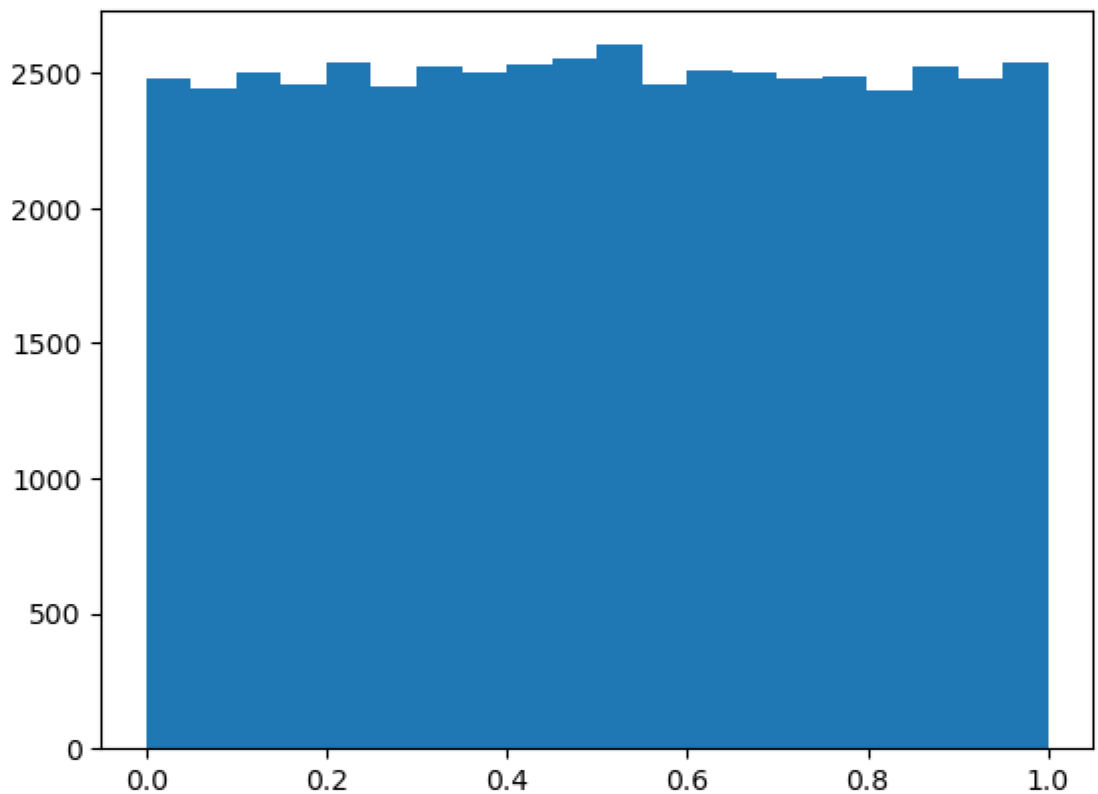


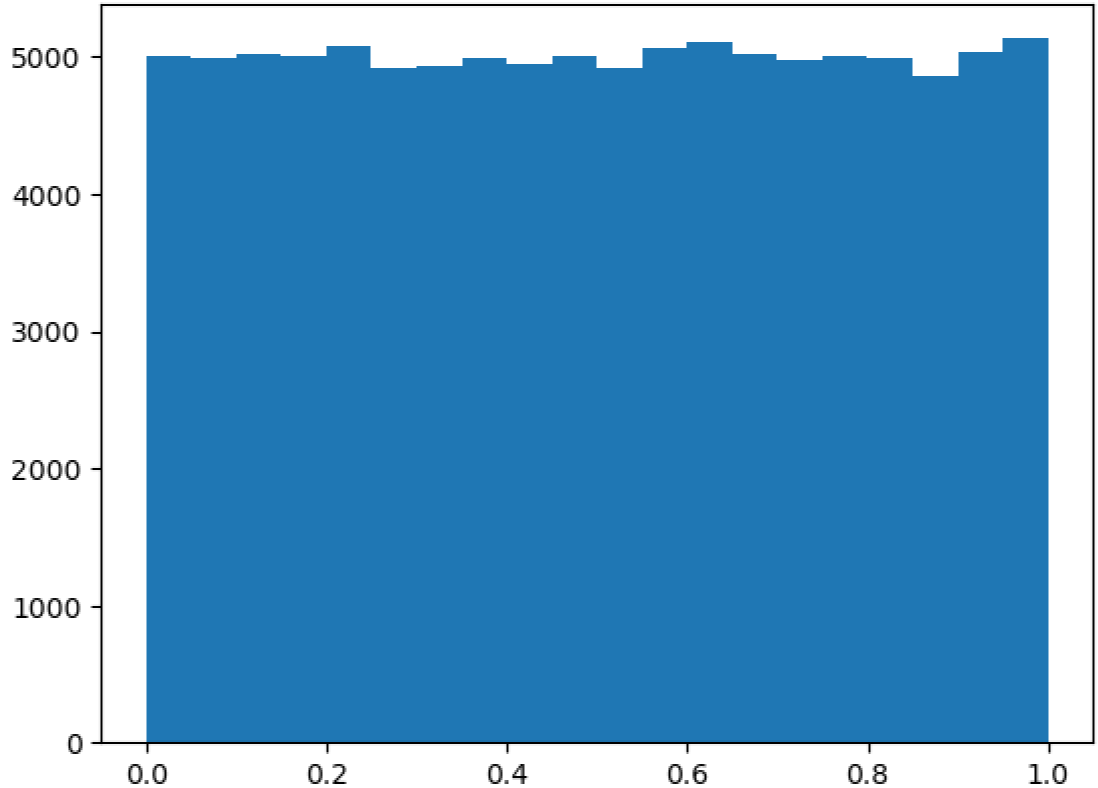
0
5
10
15
20
25
30
0.0
0.2
0.4
0.6
0.8
1.0


0
10
20
30
40
50
60
0.0
0.2
0.4
0.6
0.8
1.0


0
50
100
150
200
250
0.0
0.2
0.4
0.6
0.8
1.0


0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0

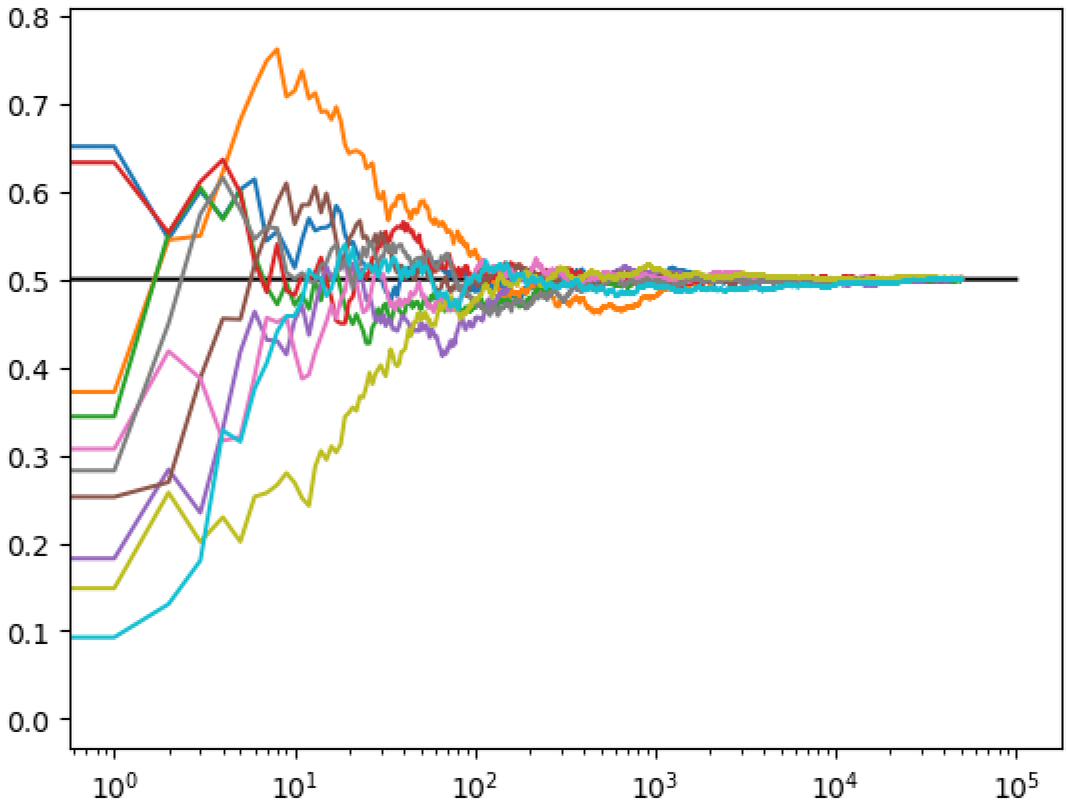

0
500
1000
1500
2000
2500
0.0
0.2
0.4
0.6
0.8
1.0



c) Law of large numbers: $average(x_{sampled}) = \bar{x}$, where x is a uniform random variable of range [0,1], thus $\bar{x} = \int_0^1 xf(x)dx = 0.5$

In [3]:

```
N =  50000# Number of points (>10000)
k =  10# set a value for number of runs

## Below code plots the semilog scaled on x-axis where all the samples are equal to the me
m = 0.5 # mean of uniform distribution
m = np.tile(m,x.shape)
plt.semilogx(m,color='k') # Ref : https://matplotlib.org/stable/api/_as_gen/matplotlib.pyp

for j in range(k):

    i =  np.arange(1,N+1)#Generate a list of numbers from (1,N) # Ref : https://numpy.org/
    x =  np.random.uniform(0,1,N)#  Generate N points from a uniform distribution range fr
    mean_sampled = np.cumsum(x)/(i) # Ref : https://numpy.org/doc/stable/reference/generat
    plt.semilogx(mean_sampled)
plt.show()
  ## Write code to plot semilog scaled on x-axis of mean_sampled, follow the above code or
```

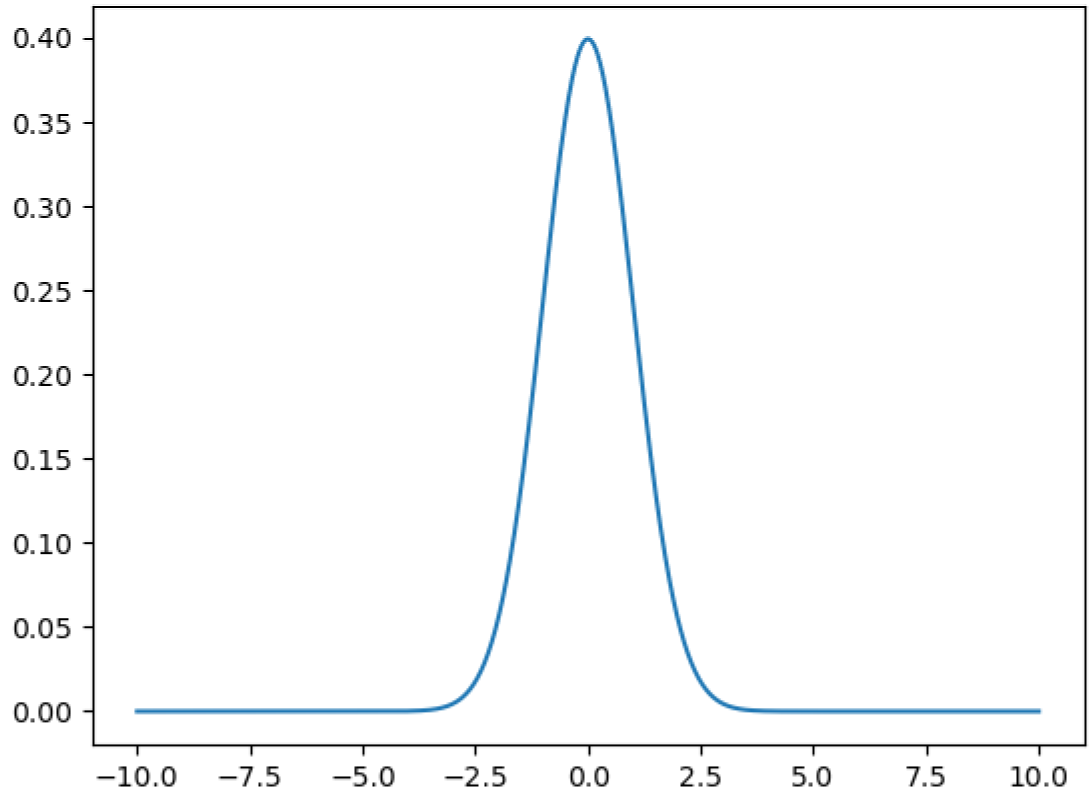


2. Sampling from Gaussian Distribution

a) Draw univariate Gaussian distribution (mean 0 and unit variance)

In [4]:

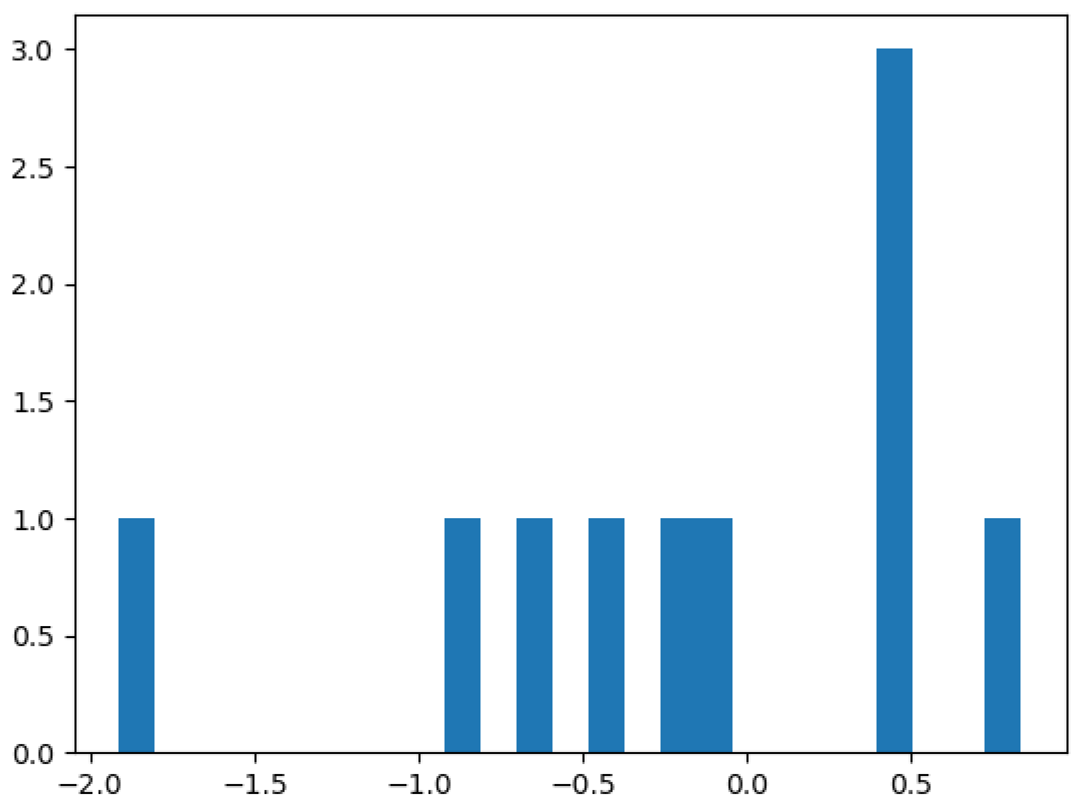
```
import numpy as np
import matplotlib.pyplot as plt

X = np.linspace(-10,10,1000) # Generate 1000 points from -10 to 10 # Ref : https://numpy.c

# Define mean and variance
mean =0
variance=1

gauss_distribution =  np.sqrt(1/(2*np.pi*variance))*np.exp(-0.5*((X-mean)**2)/variance)# I

## Write code to plot the above distribution # Ref : https://matplotlib.org/stable/api/_as
plt.plot(X,gauss_distribution)
plt.show()
```

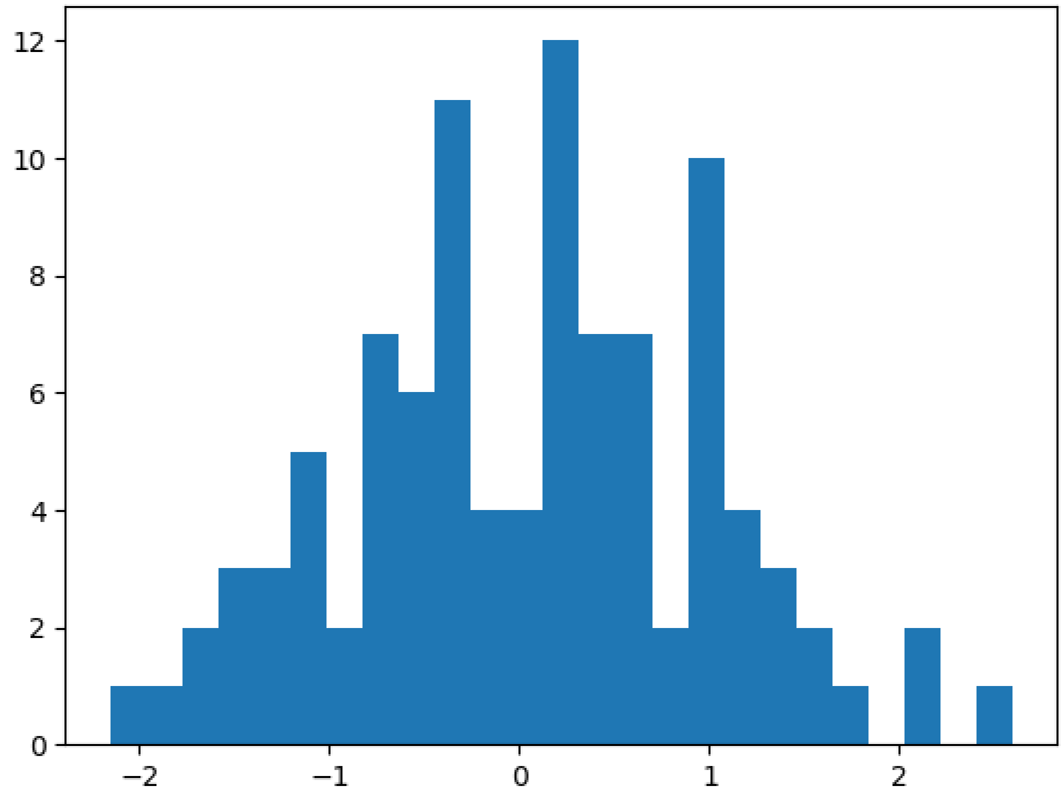


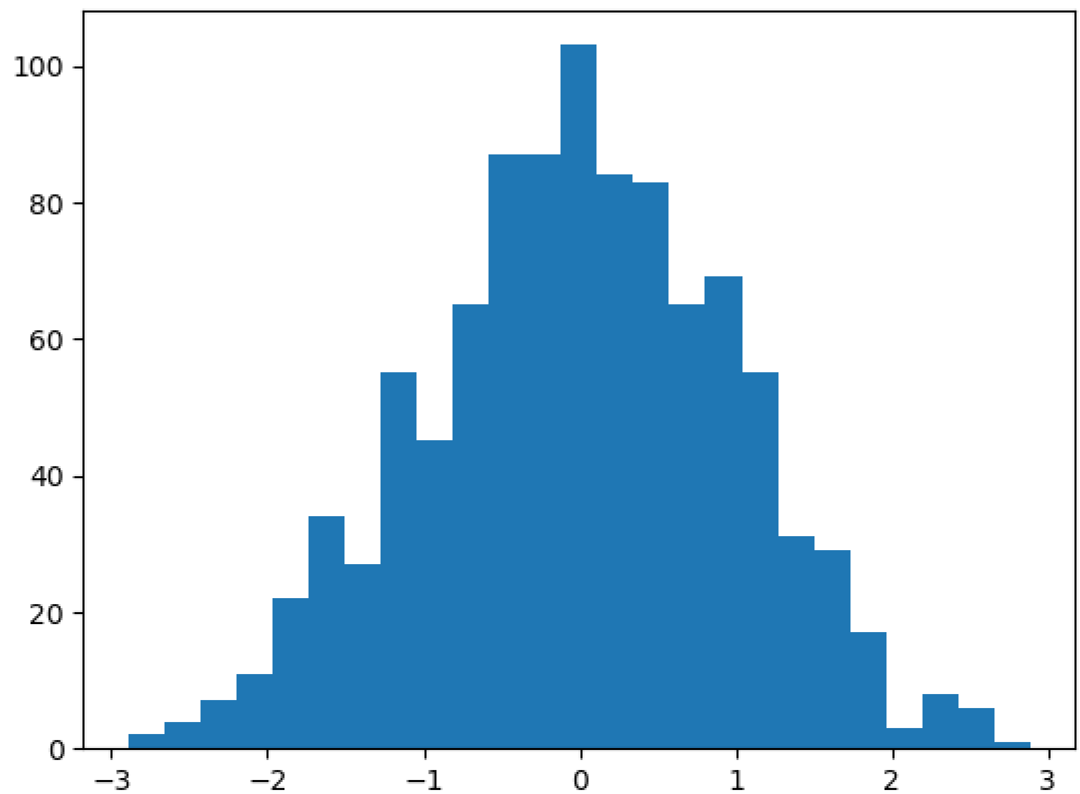
b) Sample from a univariate Gaussian distribution, observe the shape by changing the no. of sample drawn.

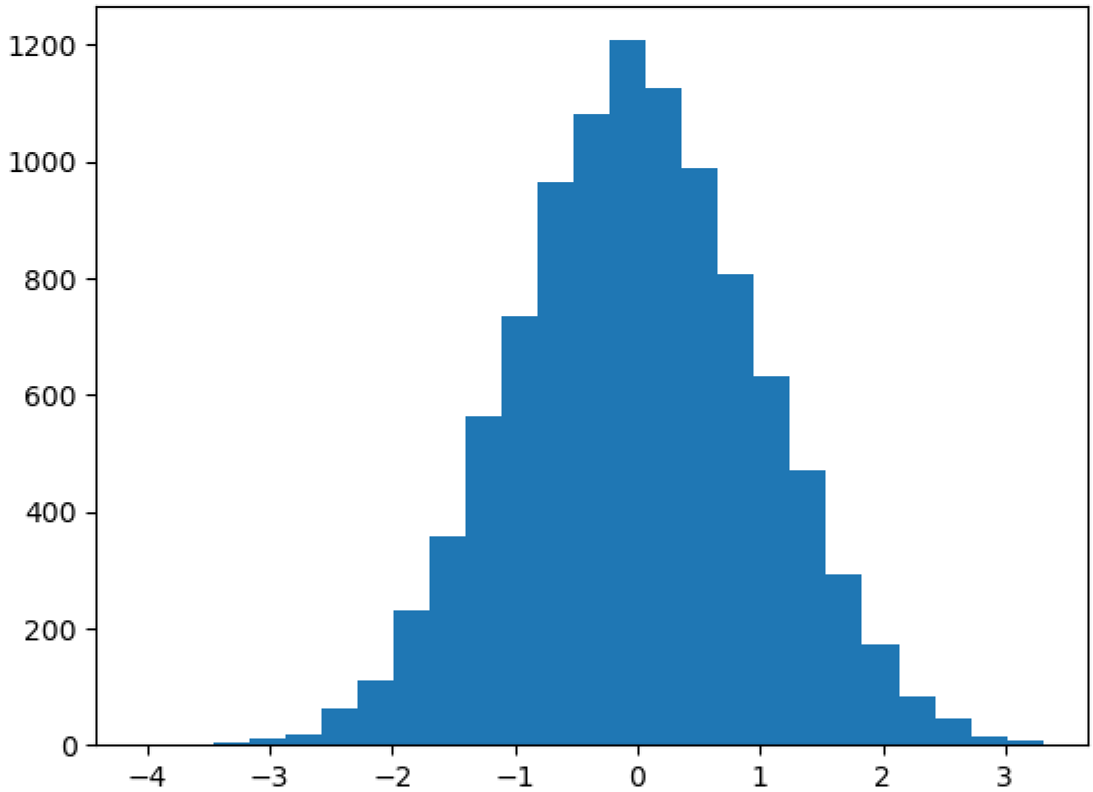
In [5]:

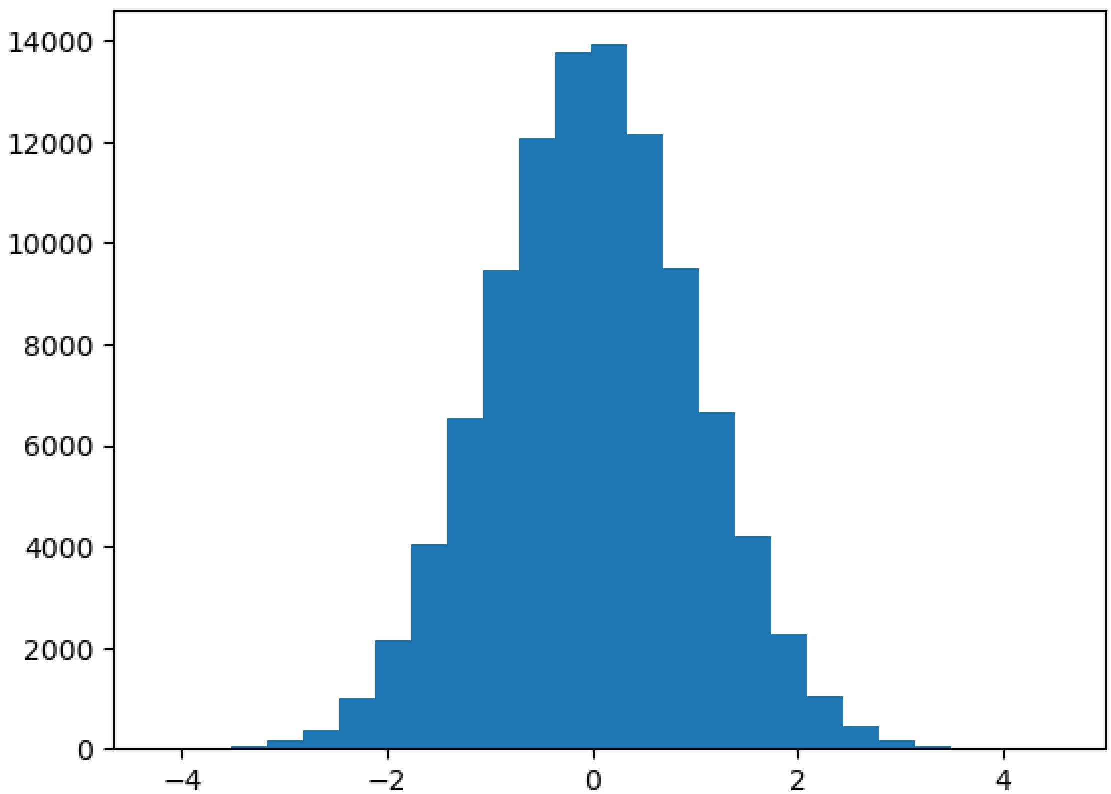
```
arr =  np.array([10,100,1000,10000,100000])# Create a numpy array of differnt values of nc

for i in arr:
    x_sampled =  np.random.normal(0,1,i)# Generate i samples from univariate gaussian dist
    plt.hist(x_sampled,bins=25)
    plt.show()
  # write the code to plot the histogram of the samples for all values in arr
```




12
10
8
6
4
2
0
−2
−1
0
1
2


100
80
60
40
20
0
−3
−2
−1
0
1
2
3

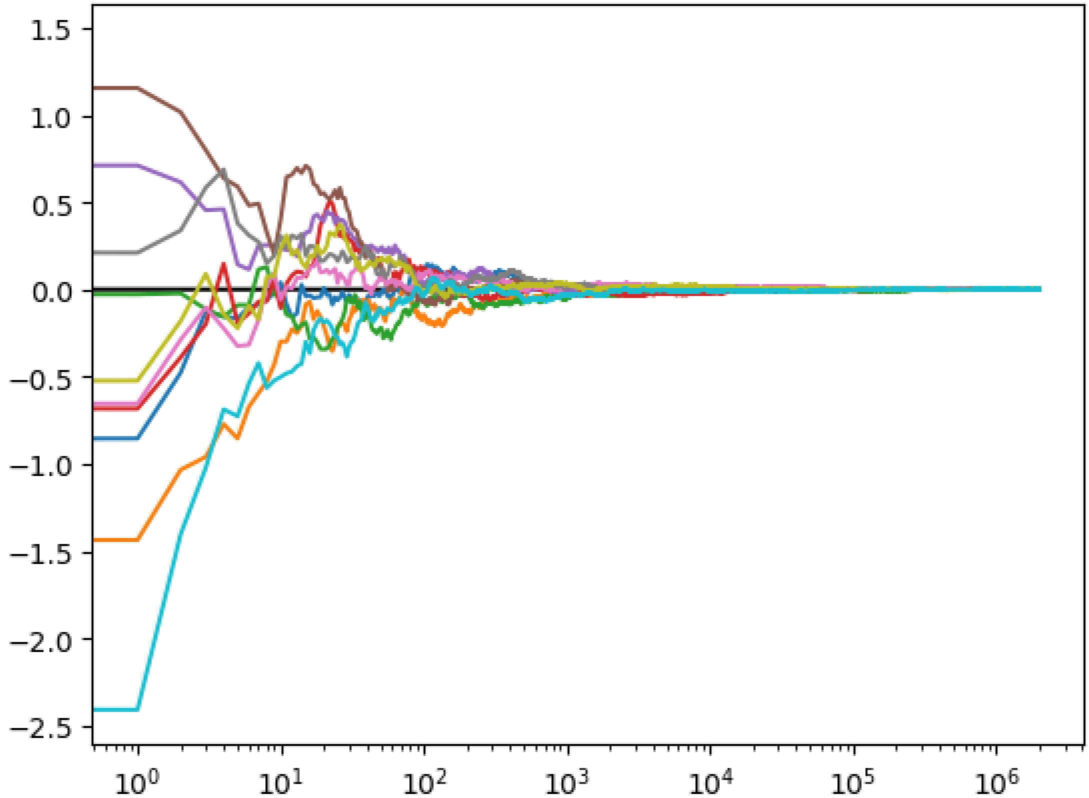




c) Law of large number

In [6]:
```
N =  2000000#  Number of points (>1000000)
k =  10# set a value for number of distributions

## Below code plots the semilog when all the samples are equal to the mean of distributio

m = np.tile(mean,x.shape)
plt.semilogx(m,color='k')

for j in range(k):
```

```
    i =  np.arange(1,N+1)# Generate a list of numbers from (1,N)
    x =  np.random.normal(0,1,N)# Generate N samples from univariate gaussian distributio
    mean_sampled =  np.cumsum(x)/i# insert your code here (Hint : Repeat the same steps as
    plt.semilogx(mean_sampled)
plt.show()
  ## Write code to plot semilog scaled on x axis of mean_sampled, follow the above code o
```



3.Sampling of categorical from uniform

i) Generate n points from uniforms distribution range from [0 1] (Take large n)

ii) Let $prob_0 = 0.3$, $prob_1 = 0.6$ and $prob_2 = 0.1$

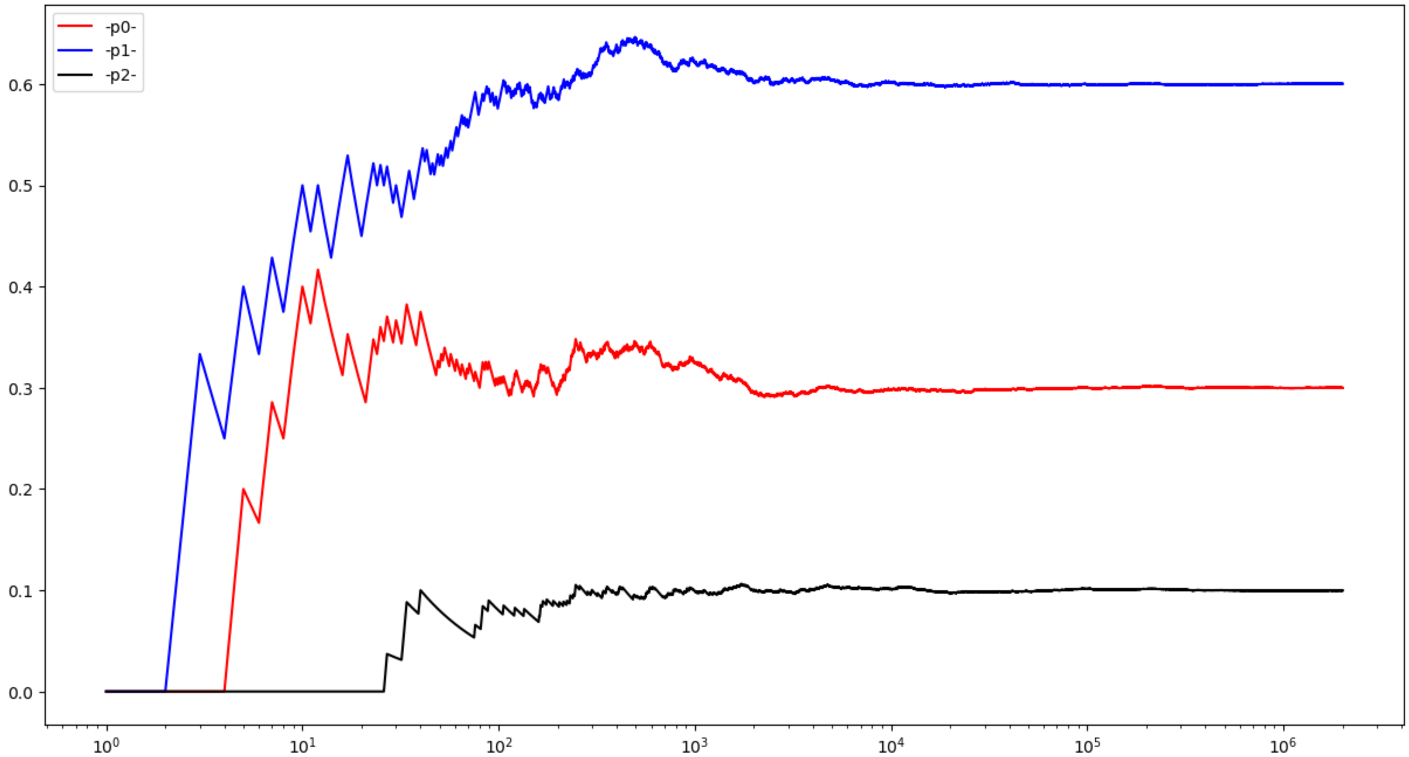
iii) Count the number of occurences and divide by the number of total draws for 3 scenarios :

1. $p_0 :< prob_0$
2. $p_1 :< prob_1$
3. $p_2 :< prob_2$

In [7]:

```
n =  2000000# Number of points (>1000000)
y =  np.random.uniform(0,1,n)# Generate n points from uniform distribution range from [0 1
x = np.arange(1, n+1)
prob0 = 0.3
prob1 = 0.6
prob2 = 0.1

# count number of occurrences and divide by the number of total draws

p0 =  np.cumsum(y<prob0)/x# insert your code here
p1 =  np.cumsum(y<prob1)/x# insert your code here
p2 =  np.cumsum(y<prob2)/x# insert your code here

plt.figure(figsize=(15, 8))
plt.semilogx(x, p0,color='r')
plt.semilogx(x, p1,color='b')
```

```
plt.semilogx(x,p2,color='k')
plt.legend(['-p0-','-p1-','-p2-'])
plt.show()
```

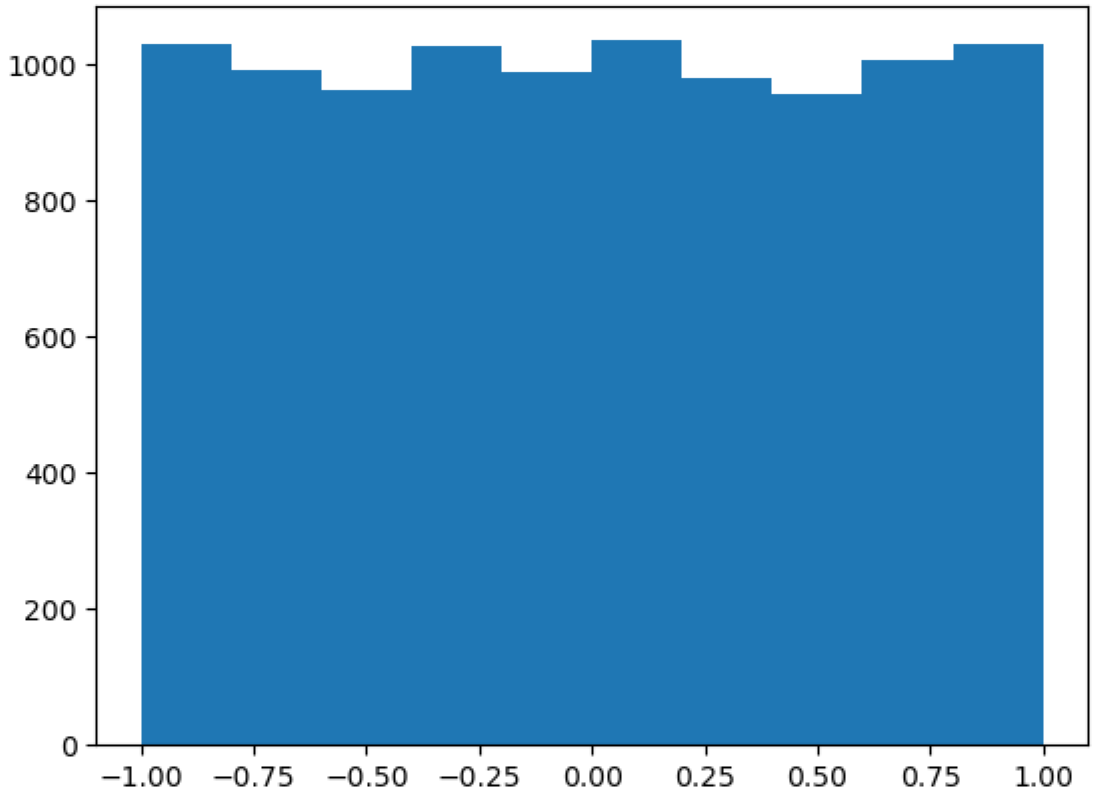


4. Central limit theorem

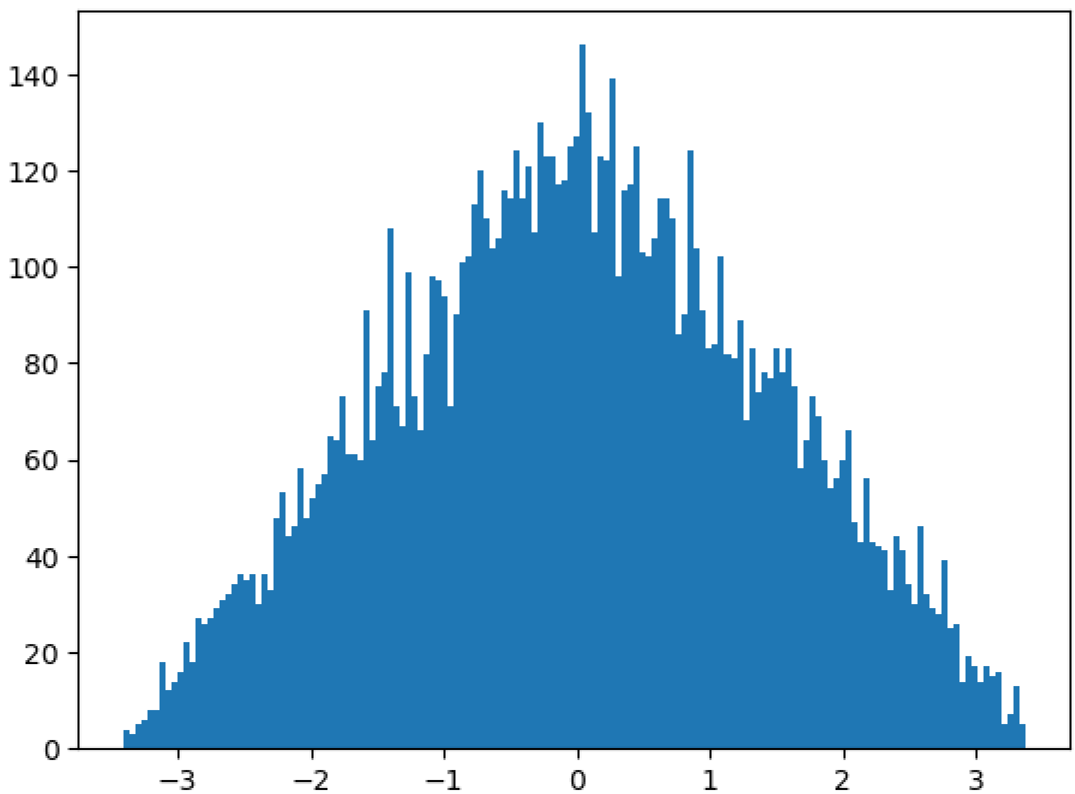
a) Sample from a uniform distribution (-1,1), some 10000 no. of samples 1000 times (u1,u2,....,u1000). show addition of iid rendom variables converges to a Gaussian distribution as number of variables tends to infinity.

In [8]:

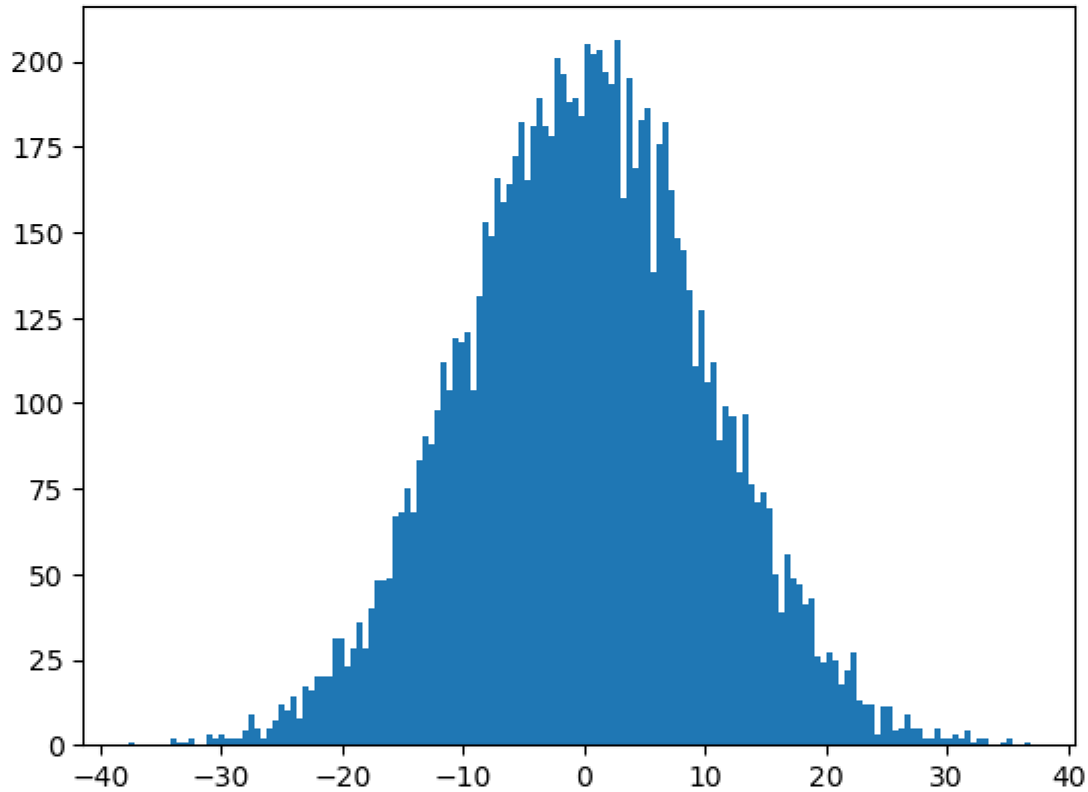
```
x =  np.random.uniform(-1,1,[10000,1000]);# Generate 1000 diferent uniform distributions c

plt.figure()
plt.hist(x[:,0])

# addition of 2 random variables
tmp2=np.sum(x[:,0:2],axis=1)/(np.std(x[:,0:2]));
plt.figure()
plt.hist(tmp2,150)

# Repeat the same for 100 and 1000 random variables

# addition of 100 random variables
# start code here
tmp100=np.sum(x[:,0:100],axis=1)/(np.std(x[:,0:100]));
plt.figure()
plt.hist(tmp100,150)

# addition of 1000 random variables
# start code here
tmp10000=np.sum(x[:,:],axis=1)/(np.std(x[:,:]));
plt.figure()
plt.hist(tmp10000,150)

plt.show()
```

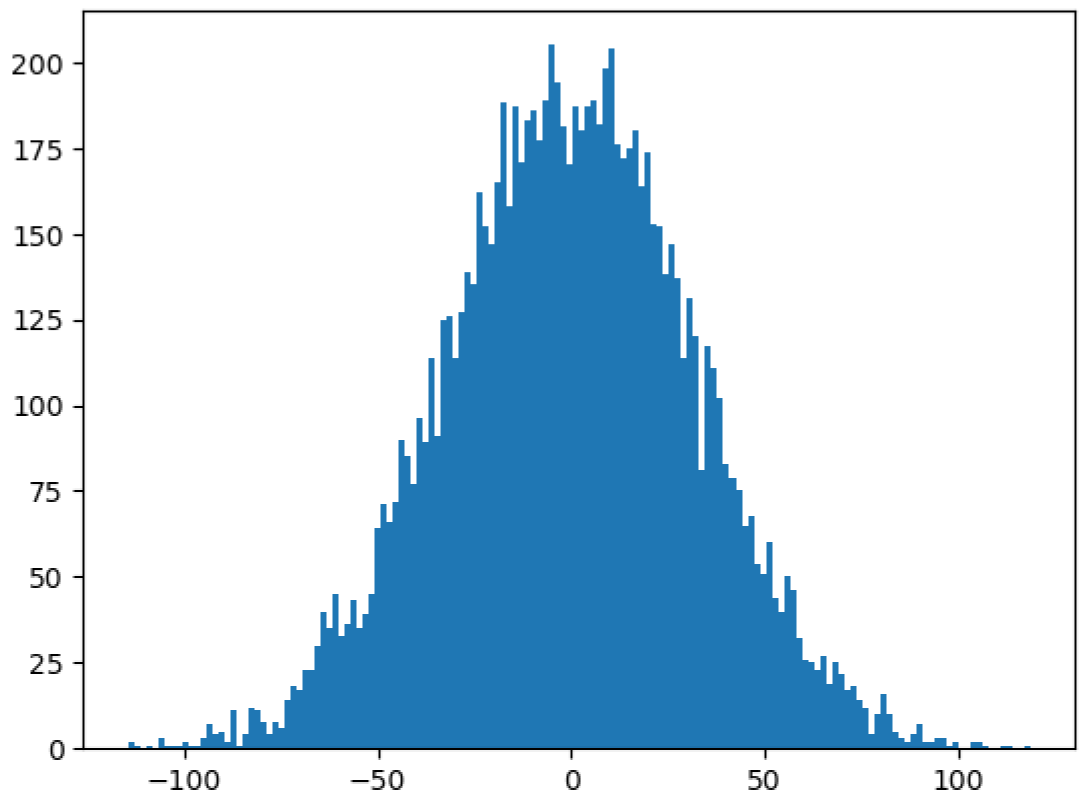


1000
800
600
400
200
0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00



140
120
100
80
60
40
20
0
−3
−2
−1
0
1
2
3





5. Computing π using sampling

a) Generate 2D data from uniform distribution of range -1 to 1 and compute the value of π.

b) Equation of circle

$$x^2 + y^2 = 1$$

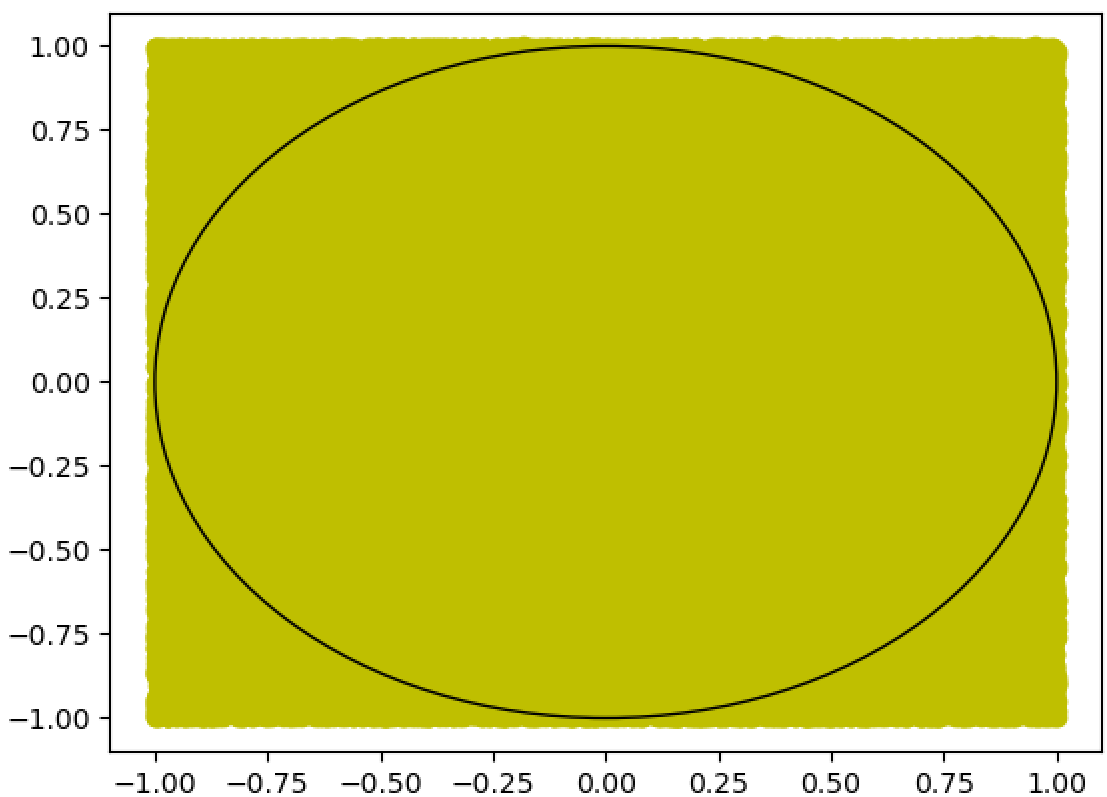
c) Area of a circle can be written as:

$$\frac{No\ of\ points\ (x^2+y^2<=1)}{Total\ no.\ generated\ points} = \frac{\pi r^2}{(2r)^2}$$

where r is the radius of the circle and $2r$ is the length of the vertices of square.

In [9]:

```
import numpy as np
import matplotlib.pyplot as plt
fig = plt.gcf()
ax = fig.gca()

radius = 1

n = 100000 # set the value of n (select large n for better results)
x = np.random.uniform(-1,1,[n,2]) # Generate n samples of 2D data from uniform distributic

ax.scatter(x[:,0],x[:,1],color='y') # Scatter plot of x

# find the number points present inside the circle

x_cr = np.sum((x[:,0]*x[:,0]+x[:,1]*x[:,1])<=1) # insert your code here

circle1 = plt.Circle((0, 0), 1,fc='None',ec='k')
ax.add_artist(circle1) # plotting circle of radius 1 with centre at (0,0)
plt.show()
pi =  4*x_cr/n# calculate pi value using x_cr and radius

print('computed value of pi=',pi)
```




```
computed value of pi= 3.14736
```

6. Monty Hall problem

Here's a fun and perhaps surprising statistical riddle, and a good way to get some practice writing python functions

In a gameshow, contestants try to guess which of 3 closed doors contain a cash prize (goats are behind the other two doors). Of course, the odds of choosing the correct door are 1 in 3. As a twist, the host of the show occasionally opens a door after a contestant makes his or her choice. This door is always one of the two the contestant did not pick, and is also always one of the goat doors (note that it is always possible to do this, since there are two goat doors). At this point, the contestant has the option of keeping his or her original choice, or swtiching to the other unopened door. The question is: is there any benefit to switching doors? The answer surprises many people who haven't heard the question before.

Follow the function descriptions given below and put all the functions together at the end to calculate the percentage of winning cash prize in both the cases (keeping the original door and switching doors)

Note : You can write your own functions, the below ones are given for reference, the goal is to calculate the win percentage

Try this fun problem and if you find it hard, you can refer to the solution here

In [10]:
```
"""
Function
--------
simulate_prizedoor

Generate a random array of 0s, 1s, and 2s, representing
hiding a prize between door 0, door 1, and door 2

Parameters
----------
nsim : int
    The number of simulations to run

Returns
-------
sims : array
    Random array of 0s, 1s, and 2s

Example
-------
>>> print simulate_prizedoor(3)
array([0, 0, 2])
"""
def simulate_prizedoor(nsim):

    answer = np.random.randint(0,3,nsim) # write your code here

    return answer
```

In [11]:
```
"""
Function
--------
simulate_guess

Return any strategy for guessing which door a prize is behind. This
could be a random strategy, one that always guesses 2, whatever.

Parameters
----------
nsim : int
    The number of simulations to generate guesses for

Returns
```

```
    -------
    guesses : array
        An array of guesses. Each guess is a 0, 1, or 2

    Example
    -------
    >>> print simulate_guess(5)
    array([0, 0, 0, 0, 0])
    """
    #your code here

    def simulate_guess(nsim):

        answer = np.random.randint(0,3,nsim)# write your code here

        return answer
```

In [12]:

```
    """
    Function
    --------
    goat_door

    Simulate the opening of a "goat door" that doesn't contain the prize,
    and is different from the contestants guess

    Parameters
    ----------
    prizedoors : array
        The door that the prize is behind in each simulation
    guesses : array
        THe door that the contestant guessed in each simulation

    Returns
    -------
    goats : array
        The goat door that is opened for each simulation. Each item is 0, 1, or 2, and is diff
        from both prizedoors and guesses

    Examples
    --------
    >>> print goat_door(np.array([0, 1, 2]), np.array([1, 1, 1]))
    >>> array([2, 2, 0])
    """
    # write your code here # Define a function and return the required array
    def goat_door(prizedoors,guesses):
        goats=(prizedoors!=guesses)*(3-prizedoors-guesses)+(prizedoors==guesses)*(prizedoors==
        return goats
```

In [13]:

```
    """
    Function
    --------
    switch_guess

    The strategy that always switches a guess after the goat door is opened

    Parameters
    ----------
    guesses : array
         Array of original guesses, for each simulation
    goatdoors : array
         Array of revealed goat doors for each simulation

    Returns
```

```
    -------
    The new door after switching. Should be different from both guesses and goatdoors

    Examples
    --------
    >>> print switch_guess(np.array([0, 1, 2]), np.array([1, 2, 1]))
    >>> array([2, 0, 0])
    """
    # write your code here # Define a function and return the required array
    def switch_guess(guesses,goatdoors):
        return (3-guesses-goatdoors)
```

In [14]:

```
"""
Function
--------
win_percentage

Calculate the percent of times that a simulation of guesses is correct

Parameters
-----------
guesses : array
    Guesses for each simulation
prizedoors : array
    Location of prize for each simulation

Returns
--------
percentage : number between 0 and 100
    The win percentage

Examples
---------
>>> print win_percentage(np.array([0, 1, 2]), np.array([0, 0, 0]))
33.333
"""

def win_percentage(guesses, prizedoors):

    answer = 100 * (guesses == prizedoors).mean()

    return answer
```

In [15]:

```
## Put all the functions together here

nsim = 100000# Number of simulations
prizes=simulate_guess(nsim)
guesses=simulate_guess(nsim)
goats=goat_door(prizes,guesses)
switch=switch_guess(guesses,goats)
## case 1 : Keep guesses
# write your code here (print the win percentage when keeping original door)
print("Win percentage for sticking:",win_percentage(guesses,prizes))
## case 2 : switch
# write your code here (print the win percentage when switching doors)
print("Win percentage for switching:",win_percentage(switch,prizes))
```

```
Win percentage for sticking: 33.129
Win percentage for switching: 66.87100000000001
```