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

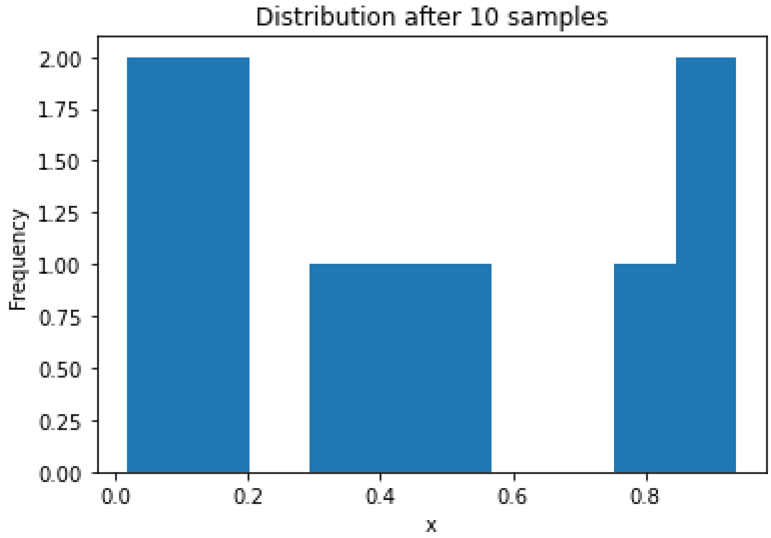
a) Generate N points from a uniform distribution range from [0 1]

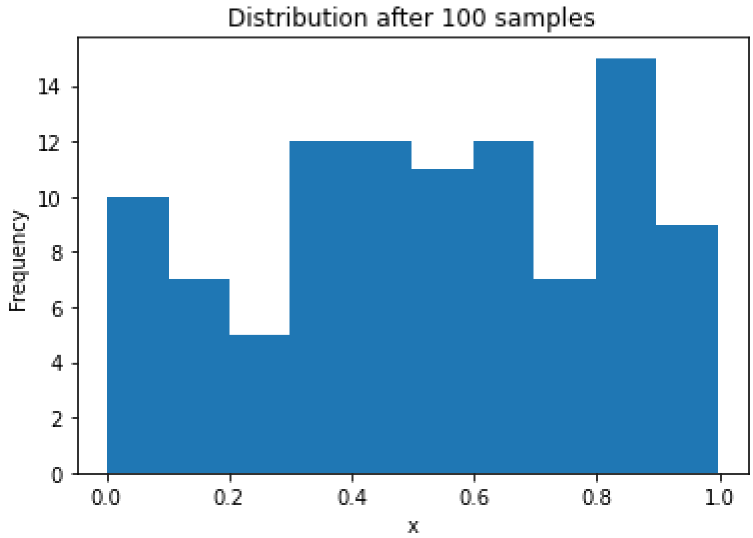
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt

N =  10 # Number of samples
X = np.random.uniform(size=N)

print(f"{N} points from a uniform distibution from [0,1]: {X}")
```

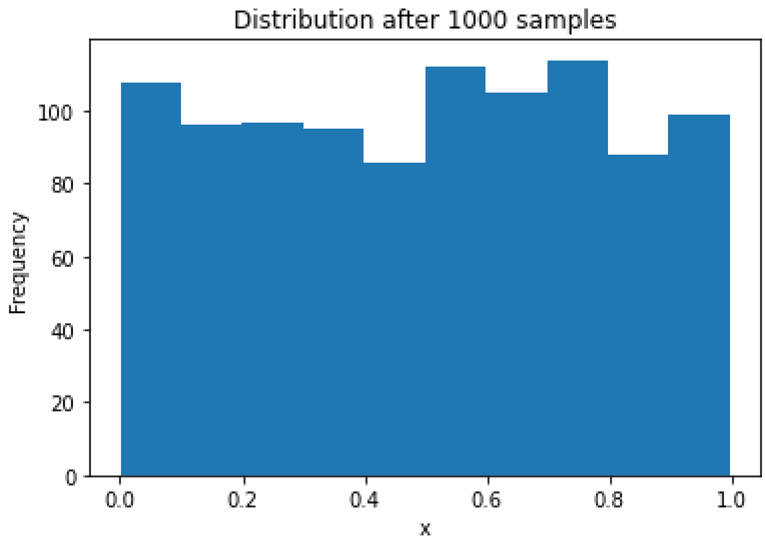
```
10 points from a uniform distibution from [0,1]: [0.02332336 0.99577499
0.28805031 0.24280312 0.59825093 0.42719487
 0.79249859 0.21551282 0.19162123 0.50432217]
```

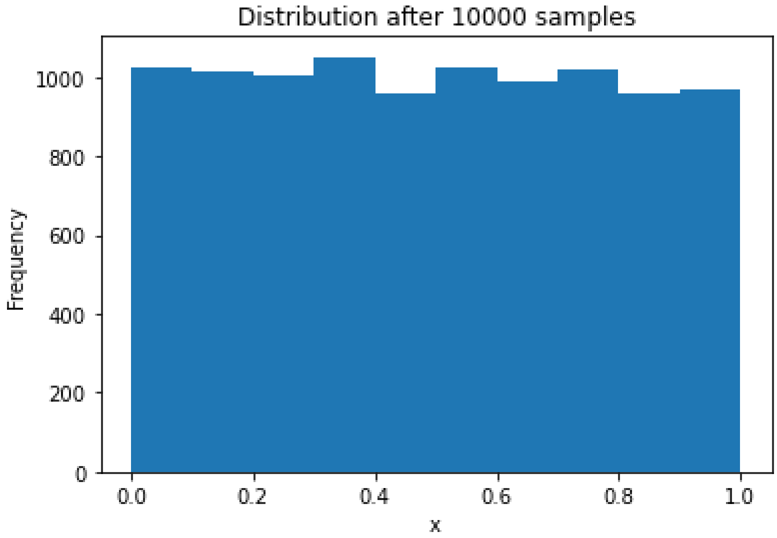
b) Show with respect to no. of sample, how the sampled distribution converges to parent distribution.

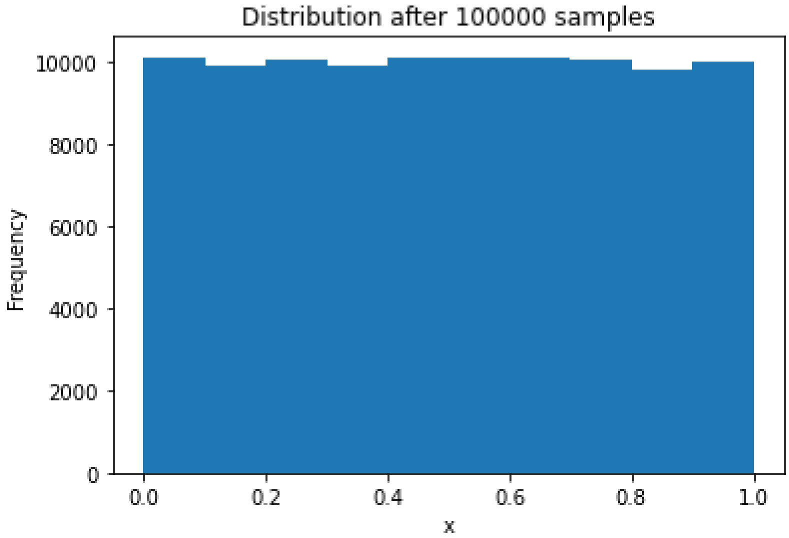
In [ ]:
```
arr = np.array([10, 100, 1000, 10000, 100000])

for i in arr:
  x = np.random.uniform(size=i)

  # Draw distribution for i samples
  n, bins, patches = plt.hist(x, 10)
  plt.xlabel("x")
  plt.ylabel("Frequency")
  plt.title(f"Distribution after {i} samples")
  plt.show()
```


Distribution after 10 samples
Frequency
x
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0.0
0.2
0.4
0.6
0.8


Distribution after 100 samples
Frequency
x
14
12
10
8
6
4
2
0
0.0
0.2
0.4
0.6
0.8
1.0

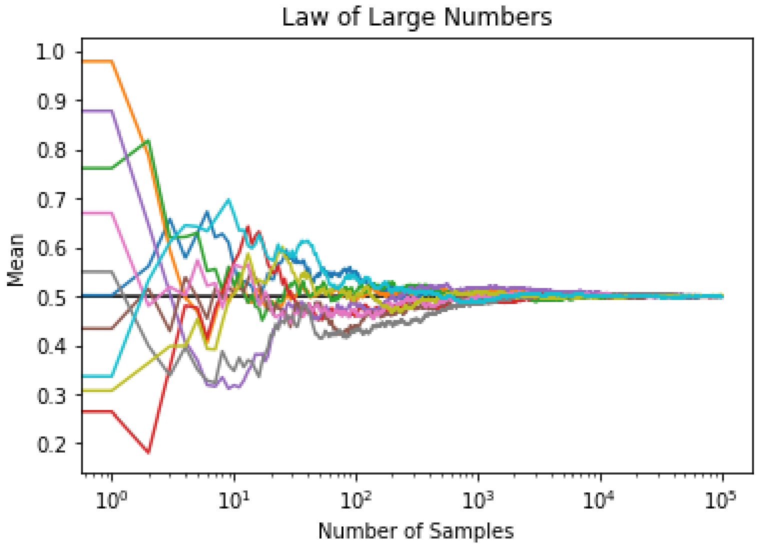

Distribution after 1000 samples
Frequency
x
100
80
60
40
20
0
0.0
0.2
0.4
0.6
0.8
1.0





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 [ ]:
```
N = 100000 # Number of samples
k = 10 # Number of iterations


m = 0.5 # mean of uniform distribution

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

# Do j iterations
for j in range(k):

  i = np.arange(1, N+1, 1)
  x = np.random.uniform(size=N)
  mean_sampled = np.cumsum(x)/(i) # Find means for various number of sample

  plt.semilogx(mean_sampled)

  plt.title("Law of Large Numbers")
  plt.xlabel("Number of Samples")
  plt.ylabel("Mean")
plt.show()
```

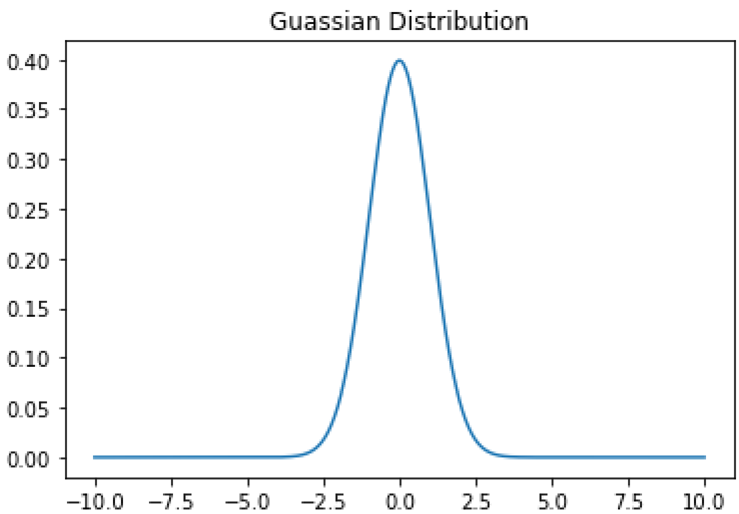


2. Sampling from Gaussian Distribution

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

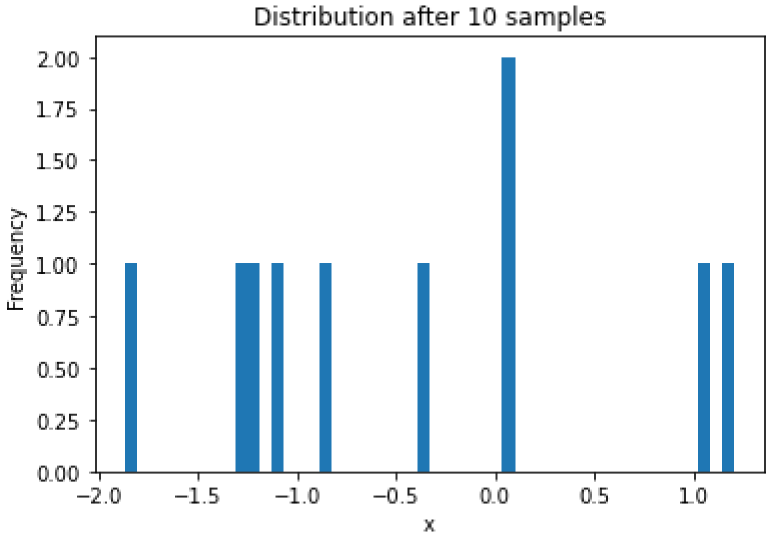
In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math

X = np.linspace(-10, 10, 1000)

# Define mean and variance
mean = 0
variance = 1
std = variance**0.5

# Calculate value of pdf for each value of X
gauss_distribution = 1/std/(2*math.pi)**0.5*math.e**(-1/2*((X-mean)/std)**2)

plt.title("Guassian Distribution")
plt.plot(X, gauss_distribution)
```

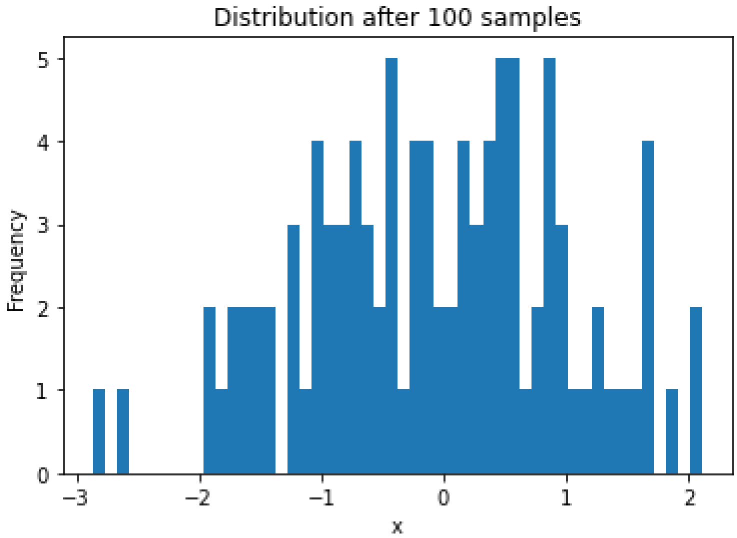
Out[ ]:
```
[<matplotlib.lines.Line2D at 0x7f2272e59540>]
```

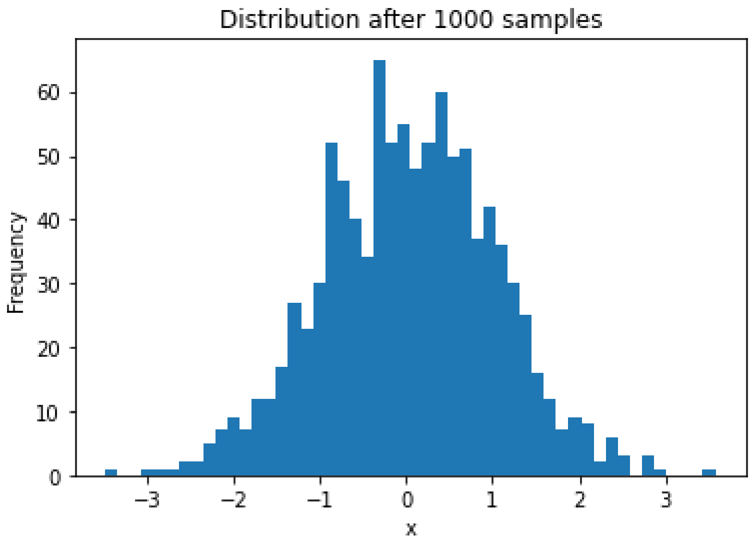


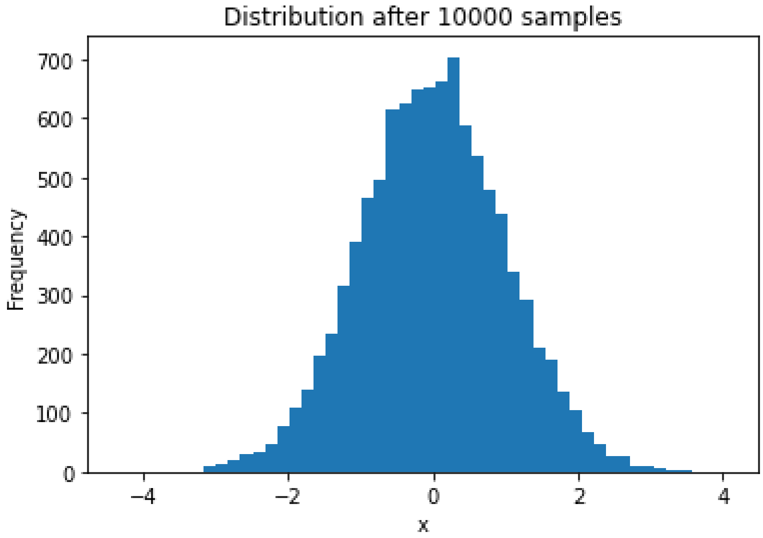
b) Sample from a univariate Gaussian distribution, observe the shape by changing the no. of sample drawn.

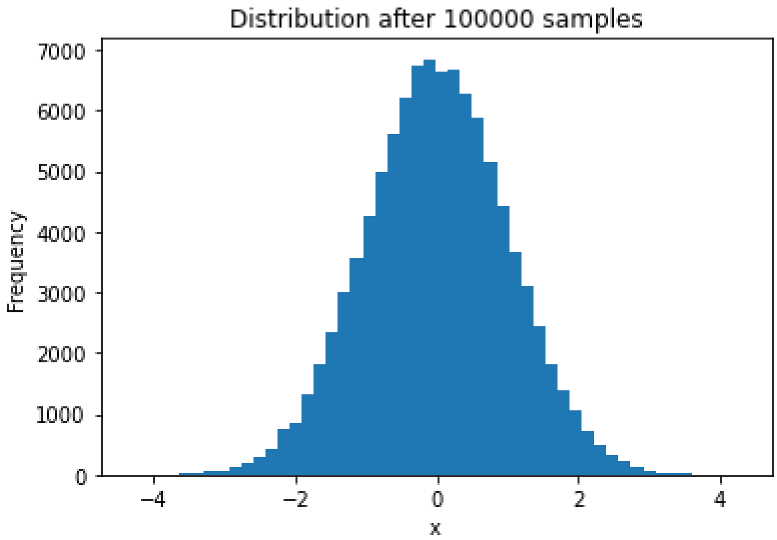
In [ ]:
```
arr = np.array([10, 100, 1000, 10000, 100000])

for i in arr:
  x_sampled = np.random.normal(0, 1, i) # Generate i samples from univariat

  # Plot the distribution
  n, bins, patches = plt.hist(x_sampled, 50)
  plt.title(f"Distribution after {i} samples")
  plt.xlabel("x")
  plt.ylabel("Frequency")
  plt.show()
```





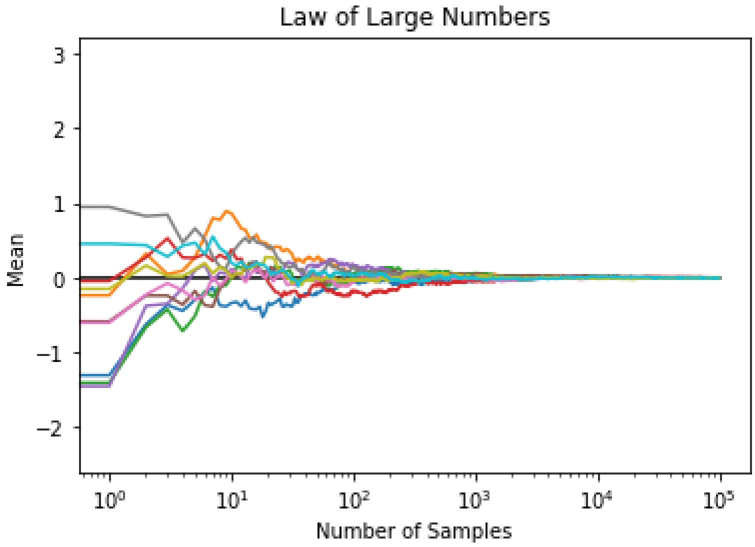






c) Law of large number

In [ ]:
```
N = 100000 #  Number of points
k = 10 # Number of iterations

# Plot mean
m = np.tile(mean, N)
plt.semilogx(m,color='k')

for j in range(k):

  i = np.arange(1, N+1, 1)
  x = np.random.normal(0, 1, N)
  mean_sampled = np.cumsum(x) / i

  plt.semilogx(mean_sampled)

plt.title("Law of Large Numbers")
plt.xlabel("Number of Samples")
plt.ylabel("Mean")
plt.show()
```



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 :

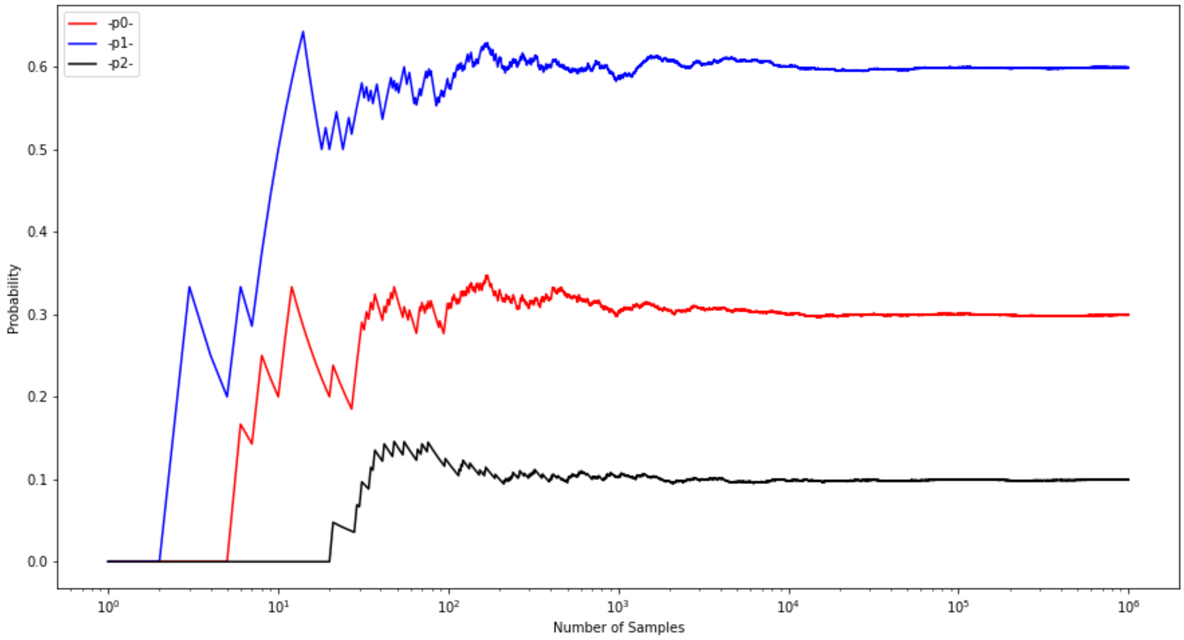
1. $p_0 :< prob_0$
2. $p_1 :< prob_1$
3. $p_2 :< prob_2$

In [ ]:

```
n = 1000000 # Number of points
y = np.random.uniform(size=n)
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
p1 = np.cumsum(y<prob1)/x
p2 = np.cumsum(y<prob2)/x


# Plot
plt.figure(figsize=(15, 8))
plt.xlabel("Number of Samples")
plt.ylabel("Probability")
plt.semilogx(x, p0,color='r')
plt.semilogx(x, p1,color='b')
plt.semilogx(x, p2,color='k')
plt.legend(['-p0-','-p1-','-p2-'])
```

Out[ ]:

```
<matplotlib.legend.Legend at 0x7f227314cb50>
```

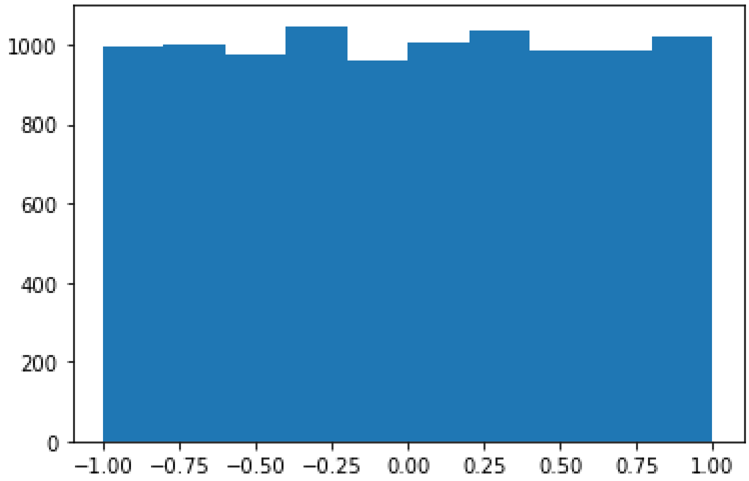


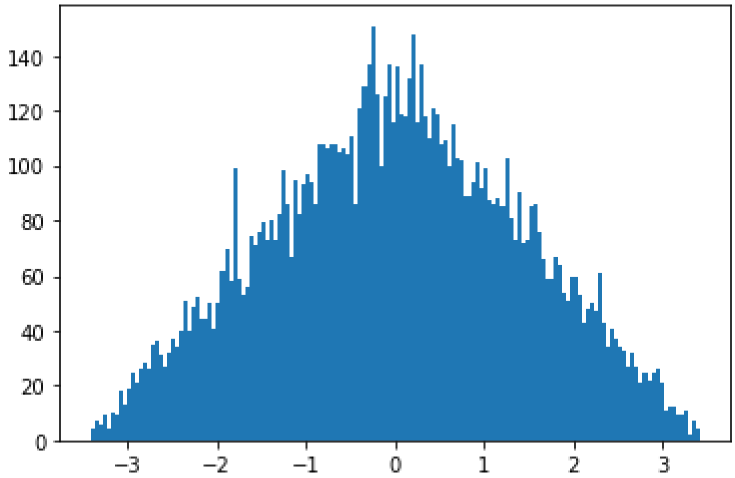
4. Central limit theorem

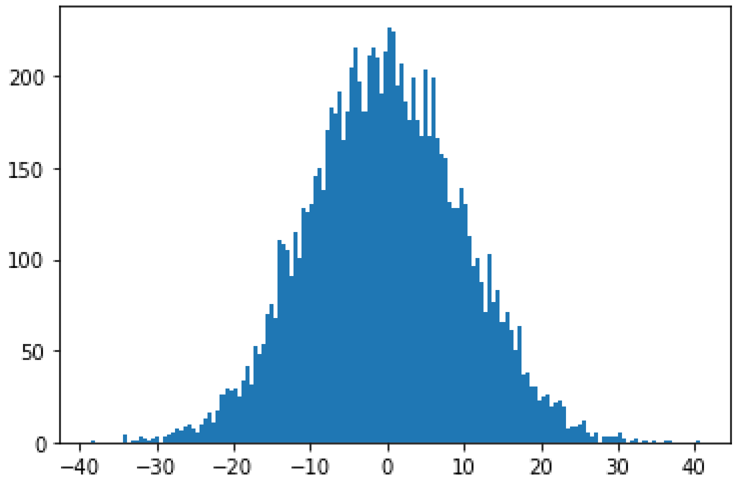
a) Sample from a uniform distribution (-1,1), some 10000 no. of samples 1000 times (u1,u2,....,u1000). show addition of iid rendom variables converges to a Gaussian distribution as number of variables tends to infinity.

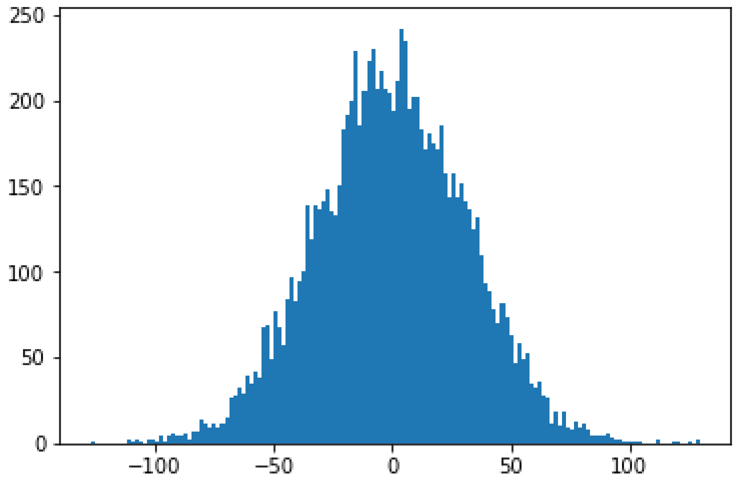
In [ ]:
```
x = np.random.uniform(-1,1,(10000,1000)) # Generate 1000 diferent uniform d

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)

# addition of 100 random variables
tmp3=np.sum(x[:,0:100],axis=1)/(np.std(x[:,0:100]))
plt.figure()
plt.hist(tmp3,150)

# addition of 1000 random variables
tmp3=np.sum(x[:,0:1000],axis=1)/(np.std(x[:,0:1000]))
plt.figure()
plt.hist(tmp3,150)
```

Out[ ]:
```
(array([  1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,   1.,
          2.,   1.,   0.,   2.,   2.,   1.,   5.,   1.,   5.,   6.,   4.,
          4.,   6.,   2.,   7.,   7.,  14.,  11.,   9.,  12.,   9.,  11.,
         15.,  27.,  28.,  32.,  29.,  40.,  35.,  42.,  38.,  67.,  69.,
         49.,  77.,  68.,  57.,  84.,  97.,  83.,  95., 100., 139., 119.,
        139., 136., 141., 148., 135., 133., 150., 183., 191., 200., 229.,
        185., 205., 223., 230., 207., 217., 207., 204., 194., 211., 242.,
        235., 195., 202., 202., 183., 171., 181., 175., 171., 186., 157.,
        144., 158., 144., 152., 141., 137., 125., 132., 110.,  93.,  89.,
         78.,  70.,  82.,  73.,  63.,  47.,  58.,  49.,  52.,  35.,  33.,
         36.,  28.,  27.,  12.,  19.,  10.,  18.,   9.,   8.,  13.,   9.,
         12.,   8.,   5.,   4.,   5.,   4.,   6.,   3.,   2.,   2.,   1.,
          1.,   1.,   1.,   1.,   0.,   0.,   0.,   2.,   0.,   0.,   0.,
          1.,   1.,   0.,   0.,   1.,   0.,   2.]),
 array([-127.3028793 , -125.59039162, -123.87790394, -122.16541627,
        -120.45292859, -118.74044091, -117.02795323, -115.31546555,
        -113.60297788, -111.8904902 , -110.17800252, -108.46551484,
        -106.75302716, -105.04053949, -103.32805181, -101.61556413,
         -99.90307645,  -98.19058878,  -96.4781011 ,  -94.76561342,
         -93.05312574,  -91.34063806,  -89.62815039,  -87.91566271,
         -86.20317503,  -84.49068735,  -82.77819967,  -81.065712  ,
         -79.35322432,  -77.64073664,  -75.92824896,  -74.21576129,
         -72.50327361,  -70.79078593,  -69.07829825,  -67.36581057,
         -65.6533229 ,  -63.94083522,  -62.22834754,  -60.51585986,
         -58.80337218,  -57.09088451,  -55.37839683,  -53.66590915,
         -51.95342147,  -50.24093379,  -48.52844612,  -46.81595844,
         -45.10347076,  -43.39098308,  -41.67849541,  -39.96600773,
         -38.25352005,  -36.54103237,  -34.82854469,  -33.11605702,
         -31.40356934,  -29.69108166,  -27.97859398,  -26.2661063 ,
         -24.55361863,  -22.84113095,  -21.12864327,  -19.41615559,
         -17.70366791,  -15.99118024,  -14.27869256,  -12.56620488,
         -10.8537172 ,   -9.14122953,   -7.42874185,   -5.71625417,
          -4.00376649,   -2.29127881,   -0.57879114,    1.13369654,
           2.84618422,    4.5586719 ,    6.27115958,    7.98364725,
           9.69613493,   11.40862261,   13.12111029,   14.83359796,
          16.54608564,   18.25857332,   19.971061  ,   21.68354868,
          23.39603635,   25.10852403,   26.82101171,   28.53349939,
          30.24598707,   31.95847474,   33.67096242,   35.3834501 ,
          37.09593778,   38.80842546,   40.52091313,   42.23340081,
          43.94588849,   45.65837617,   47.37086384,   49.08335152,
          50.7958392 ,   52.50832688,   54.22081456,   55.93330223,
          57.64578991,   59.35827759,   61.07076527,   62.78325295,
          64.49574062,   66.2082283 ,   67.92071598,   69.63320366,
          71.34569134,   73.05817901,   74.77066669,   76.48315437,
          78.19564205,   79.90812972,   81.6206174 ,   83.33310508,
          85.04559276,   86.75808044,   88.47056811,   90.18305579,
          91.89554347,   93.60803115,   95.32051883,   97.0330065 ,
          98.74549418,  100.45798186,  102.17046954,  103.88295721,
         105.59544489,  107.30793257,  109.02042025,  110.73290793,
         112.4453956 ,  114.15788328,  115.87037096,  117.58285864,
         119.29534632,  121.00783399,  122.72032167,  124.43280935,
         126.14529703,  127.85778471,  129.57027238]),
 <BarContainer object of 150 artists>)
```


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


200
150
100
50
0
−40 −30 −20 −10 0 10 20 30 40


250
200
150
100
50
0
−100 −50 0 50 100

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

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

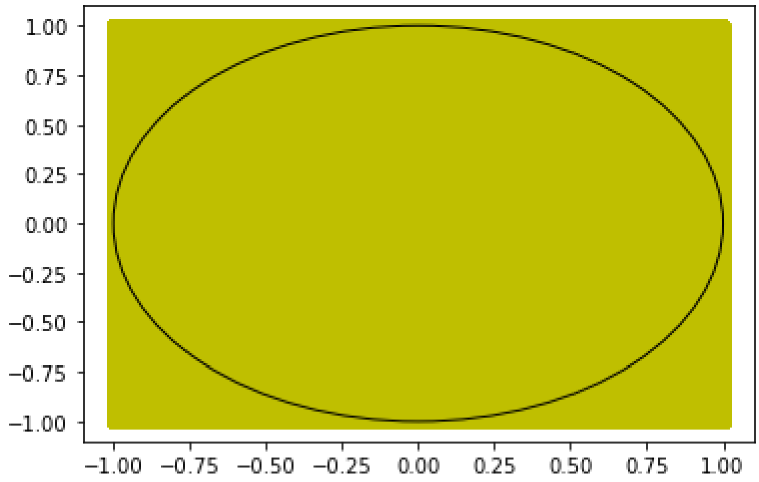
c) Area of a circle can be written as:

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

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

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

radius = 1

n = 10000000 # set the value of n
x = np.random.uniform(-1, 1, (n,2)) # Generate n samples of 2D data from un

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]**2+x[:,1]**2)<=1)

circle1 = plt.Circle((0, 0), 1,fc='None',ec='k')
ax.add_artist(circle1) # plotting circle of radius 1 with centre at (0,0)

# calculate pi value using x_cr and radius
pi = 4 * x_cr / n

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

```
computed value of pi= 3.1419684
```



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 [ ]:

```
"""
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, 2 + 1, nsim) # Sample nsim integers from di

  return answer
```

In [ ]:

```
"""
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, 2 + 1, nsim)  # Sample nsim integers from d

  return answer
```

In [ ]:

```
"""
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
    from both prizedoors and guesses

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

def goat_door(prizedoors, guesses):

  def get_goat_door(prizedoor, guess):
    # Find door which is not prizedoor not guess
    arr = [0,1,2]
    arr.remove(prizedoor)
    if guess in arr:
      arr.remove(guess)

    return np.random.choice(arr)

  answer = [get_goat_door(prizedoor, guess) for (prizedoor, guess) in zip (p

  return answer
```

In [ ]:

```
"""
```

```
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 goat

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

def switch_guess(guesses, goat_doors):
  # Choose door other than guess and goat_door
  return [3]*len(guesses) - guesses - goat_doors
```

In [ ]:
```
"""
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):
  # Find percentage of times guess is same as prizedoor
  answer = 100 * (guesses == prizedoors).mean()

  return answer
```

In [ ]:
```
import numpy as np

nsim = 1000000 # Number of simulations

## case 1 : Keep guesses
```

```
guesses = simulate_guess(nsim)
prizedoors = simulate_prizedoor(nsim)
print(f"Win percentage when original door is kept: {win_percentage(guesses,

## case 2 : Switch
guesses = simulate_guess(nsim)
prizedoors = simulate_prizedoor(nsim)
goatdoor = goat_door(prizedoors, guesses)
guesses = switch_guess(guesses, goatdoor)
print(f"Win percentage when door is switched: {win_percentage(guesses, prize
```

```
Win percentage when original door is kept: 33.2802
Win percentage when door is switched: 66.6283
```