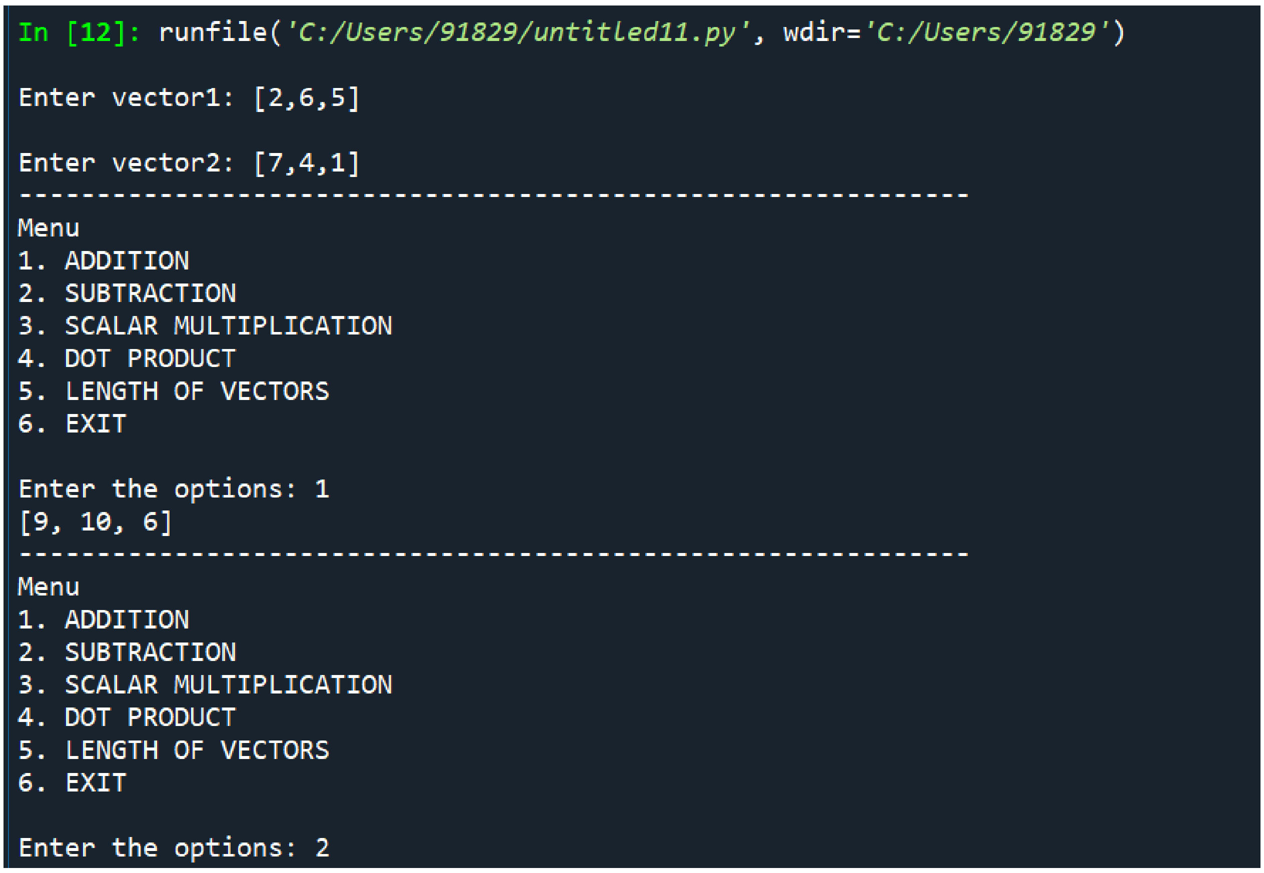
1. **Write a menu-driven program to perform Addition, Subtraction, Scalar Multiplication, Dot Product and Length of vectors.**

```
from typing import List
Vector=List[float]
def add(v:Vector,w:Vector) -> Vector:
   assert len(v)==len(w)
   return [v_i+w_i for v_i,w_i in zip(v,w)]
def subtract(v:Vector,w:Vector) -> Vector:
   assert len(v)==len(w)
   return [vi-wi for vi,wi in zip(v,w)]
def scalar_multiply(c:float,v:Vector) ->Vector:
   return [c*vi for vi in v]
def dot_product(v:Vector,w:Vector) ->  Vector:
   assert len(v)==len(w)
   return sum(vi*wi for vi,wi in zip(v,w))
def main():
   isExit=False
   mat1=eval(input("Enter vector1: "))
   mat2=eval(input("Enter vector2: "))
   while not isExit:
      print('-----------------------------------------------------------')
      print('Menu')
      print('1. ADDITION')
      print('2. SUBTRACTION')
      print('3. SCALAR MULTIPLICATION')
      print('4. DOT PRODUCT')
      print('5. LENGTH OF VECTORS')
      print('6. EXIT')
      user_input=int(input('Enter the options: '))
      if(user_input==1):
         print(add(mat1,mat2))
      elif user_input==2:
         print(subtract(mat1,mat2))
      elif user_input==3:
         print("Enter the scalar: ")
         scalar=int(input())
         print(scalar_multiply(scalar, mat1))
      elif user_input==4:
         print(dot_product(mat1, mat2))
      elif user_input==5:
         print('Length of Vector1 ',len(mat1))
         print('Length of Vector2 ',len(mat2))
      elif user_input==6:
         print("You logged out")
         isExit= not isExit
      else:
         print("Invalid Input")
         continue
if __name__=='__main__':
    main()
```

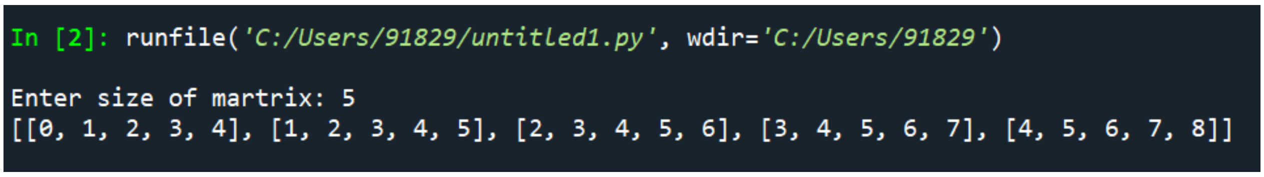
Name:



```
In [12]: runfile('C:/Users/91829/untitled11.py', wdir='C:/Users/91829')

Enter vector1: [2,6,5]

Enter vector2: [7,4,1]
-------------------------------------------------------------
Menu
1. ADDITION
2. SUBTRACTION
3. SCALAR MULTIPLICATION
4. DOT PRODUCT
5. LENGTH OF VECTORS
6. EXIT

Enter the options: 1
[9, 10, 6]
-------------------------------------------------------------
Menu
1. ADDITION
2. SUBTRACTION
3. SCALAR MULTIPLICATION
4. DOT PRODUCT
5. LENGTH OF VECTORS
6. EXIT

Enter the options: 2
```

2. Write a program that takes the order of the matrix and creates a matrix in the following manner: The (ij)th entry of the matrix should be the sum of i and j. Eg: The 0th row and 0th column should have the value (0+0) i.e. 0 and the 0th row and first column should have value (0+1) i.e. 1 and so on.

```
from typing import List
from typing import Callable
Matrix=List[List[float]]
def make_matrix(num_rows:int,num_cols:int,entry_func:Callable[[int,int],int]):
   return [[entry_fn(i,j) for j in range(num_cols)] for i in range(num_rows)]
def getValue(n:int) -> Matrix:
   return make_matrix(n,n,lambda i,j : i+j)
n=int(input("Enter size of martrix: "))
print(getValue(n))
```

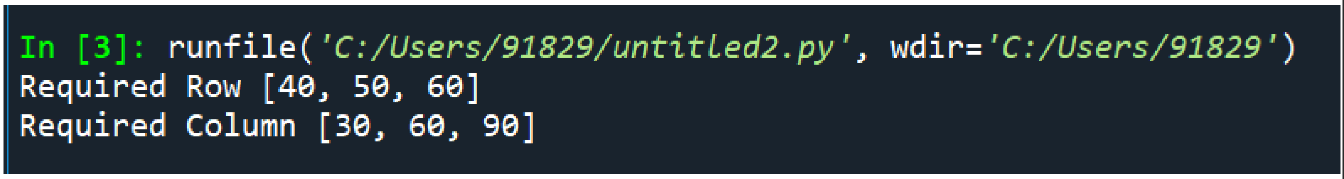


```
In [2]: runfile('C:/Users/91829/untitled1.py', wdir='C:/Users/91829')

Enter size of martrix: 5
[[0, 1, 2, 3, 4], [1, 2, 3, 4, 5], [2, 3, 4, 5, 6], [3, 4, 5, 6, 7], [4, 5, 6, 7, 8]]
```

Name:

3. Write two functions that extract the rows and columns of a matrix A.

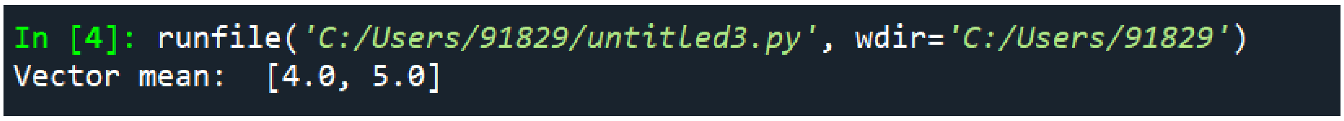
```
from typing import List
matrix=List[List[float]]
Vector=List[float]
def get_row(A:matrix,i:int) -> Vector:
    assert i<=len(A),"Index length out of matrix length"
    return A[i]
def get_Col(A:matrix,i:int) -> Vector:
    return [a[i] for a in A]
print('Required Row',get_row([[10,20,30],[40,50,60],[70,80,90]], 1))
print('Required Column',get_Col([[10,20,30],[40,50,60],[70,80,90]], 2))
```



```
In [3]: runfile('C:/Users/91829/untitled2.py', wdir='C:/Users/91829')
Required Row [40, 50, 60]
Required Column [30, 60, 90]
```

4. Write a function to compute the component-wise mean of a list of vectors. Assert the condition that the vectors must be of same length.

```
from typing import List
Vector=List[float]
def scalar_multiply(c:float,v:Vector) ->Vector:
   return [c*vi for vi in v]
def vector_sum(vectors:List[Vector]) -> Vector:
   assert vectors,"""No vector provided"""
   num_ele=len(vectors[0])
   assert all(len(v)==num_ele for v in vectors),"""different size"""
   return [sum(vector[i] for vector in vectors)
         for i in range(num_ele)]
def vector_mean(vectors:List[Vector]) -> Vector:
   n=len(vectors)
   return scalar_multiply(1/n, vector_sum(vectors))

print('Vector mean: ',vector_mean([[2,3],[4,5],[6,7]]))
```

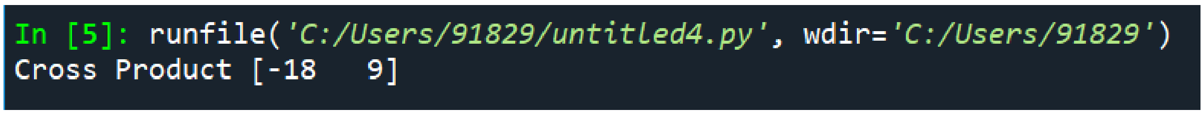


```
In [4]: runfile('C:/Users/91829/untitled3.py', wdir='C:/Users/91829')
Vector mean:  [4.0, 5.0]
```

Name:

5. Use the numpy module to create two arrays of same size and perform cross product operation on them using the built in function cross present in the numpy module.

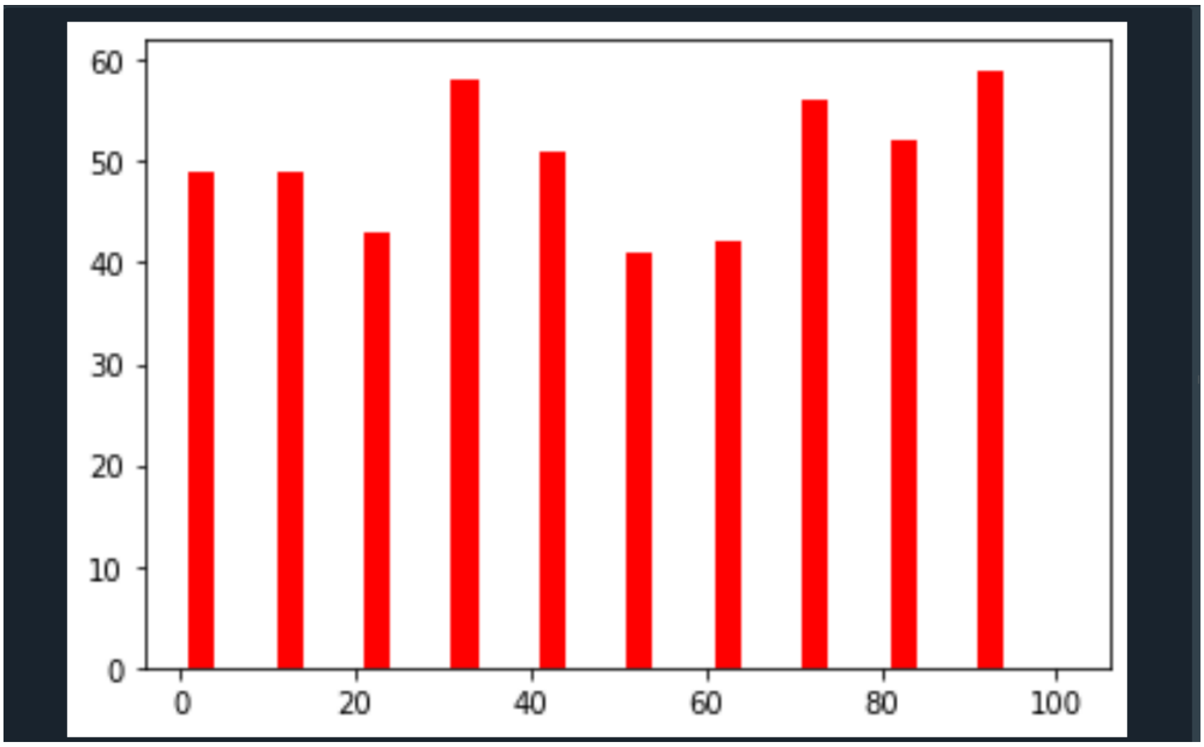
```
import numpy as np
m=np.matrix([[2,4],[9,6]])
n=np.matrix([[7,5],[9,7]])
crs_prod=np.cross(m, n)
print('Cross Product',crs_prod)
```



```
In [5]: runfile('C:/Users/91829/untitled4.py', wdir='C:/Users/91829')
Cross Product [-18   9]
```

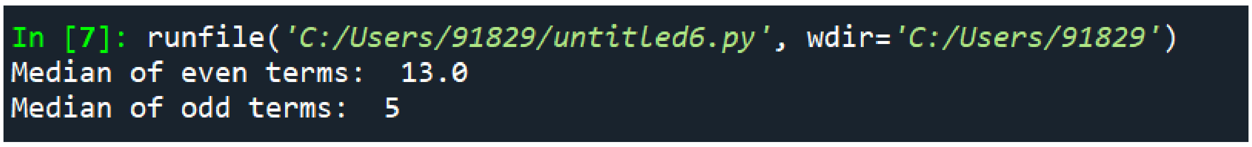
6. Generate a list of 100 random integers between 1 and 100 and plot a histogram of the same.

```
import random
from matplotlib import pyplot as plt
randomInt=[]
for i in range(500):
   randomInt.append(random.randint(1, 101))
plt.hist(randomInt,width=3,color='red')
plt.show()
```



7. Write a program to find median of a given list of integers. Combine both odd and even number of terms.

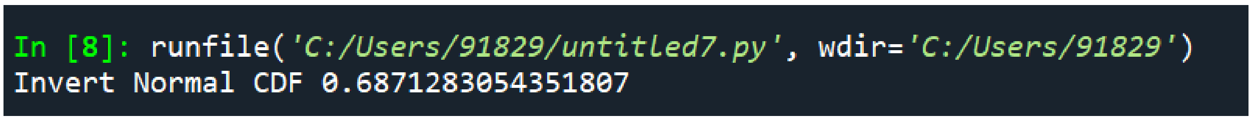
```
from typing import List
Vector=List[float]
def median_odd(xs:List[float]) -> float:
   return sorted(xs)[len(xs)//2]

def median_even(xs:List[float])-> float:
   sorted_xs=sorted(xs)
   hi_mid=len(xs)//2
   return (sorted_xs[hi_mid-1] + sorted_xs[hi_mid])/2

def median(f:List[float]) -> float:
   return median_even(f) if len(f)%2==0 else median_odd(f)
print("Median of even terms: ",median([5,20,15,12,14,10]))
print("Median of odd terms: ",median([2,8,3,5,9]))
```



```
In [7]: runfile('C:/Users/91829/untitled6.py', wdir='C:/Users/91829')
Median of even terms:  13.0
Median of odd terms:  5
```

8. We have defined the function normal cdf. Write a program to invert normal cdf to find the value corresponding to a specified probability.

```
import math
def normal_cdf(x:float,mu:float=0,sigma:float=1) -> float:
   return (1+math.erf((x-mu) / math.sqrt(2) /sigma))/2
#print('Normal CDF',normal_cdf(0.9))
def inverse_normal_cdf(p: float,mu: float = 0,sigma: float = 1,tolerance: float = 0.00001) -> float:
   if mu != 0 or sigma != 1:
      return mu + sigma * inverse_normal_cdf(p, tolerance=tolerance)
   low_z = -50.0
   hi_z = 50.0
   while hi_z - low_z > tolerance:
      mid_z = (low_z + hi_z) / 2
      mid_p = normal_cdf(mid_z)
      if mid_p < p:
         low_z = mid_z
      else:
         hi_z = mid_z
   return mid_z
print('Invert Normal CDF',inverse_normal_cdf(0.754))
```



```
In [8]: runfile('C:/Users/91829/untitled7.py', wdir='C:/Users/91829')
Invert Normal CDF 0.6871283054351807
```

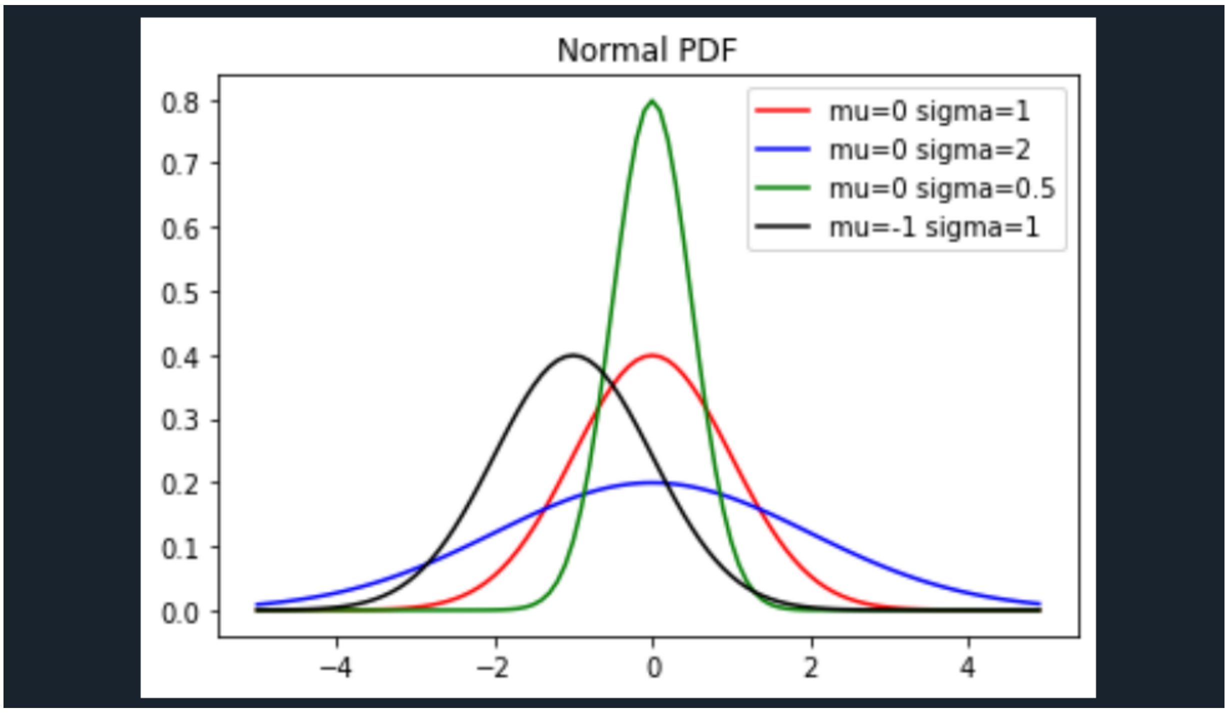
Name:

9. Plot the Normal PDFs using various value of μ and σ as mentioned below:

mean	sigma
0	1
0	2
0	0.5
-1	1

**Use different line styles for each plot and compare the graphs thus obtained.
You can use any range
for x-axis.**

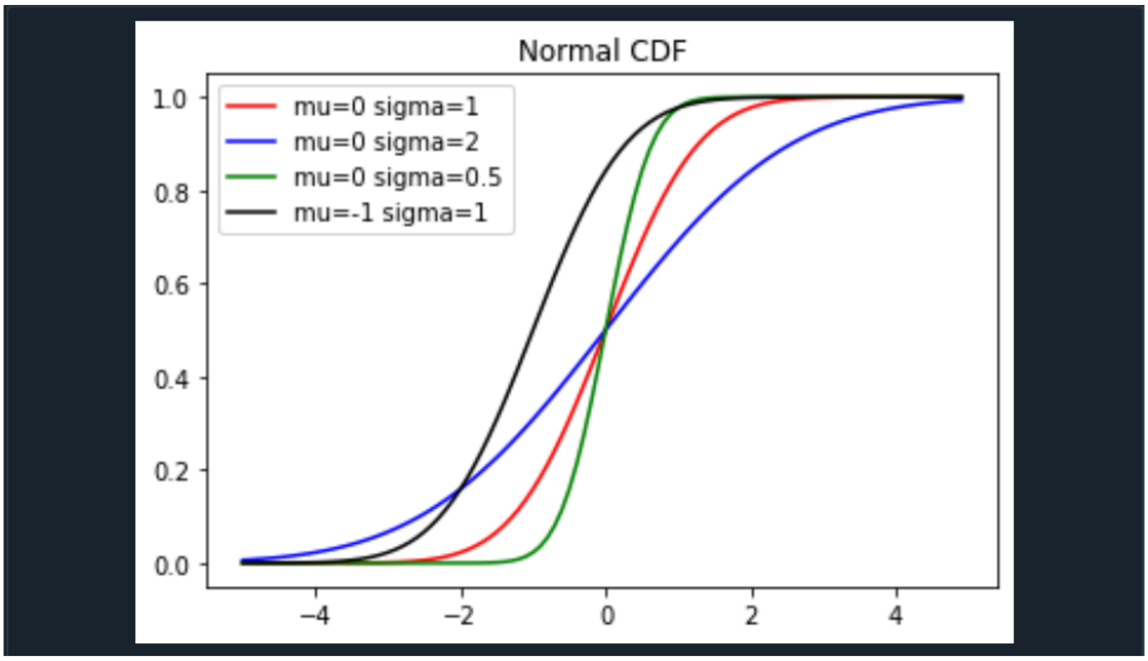
```
import math
from matplotlib import pyplot as plt
SQRT_TWO_PI=math.sqrt(2*math.pi)

def normal_pdf(x:float , mu:float=0,sigma:float=1) -> float:
   return (math.exp(-(x-mu) ** 2/2/sigma**2) / (SQRT_TWO_PI * sigma))

point=[ x /10.0 for x in range(-50,50)]
plt.plot(point , [normal_pdf(x) for x in point] , color='red' , label='mu=0 sigma=1')
plt.plot(point , [normal_pdf(x,0,2) for x in point] , color='blue' , label='mu=0 sigma=2')
plt.plot(point , [normal_pdf(x,0,0.5) for x in point] , color='green' , label='mu=0 sigma=0.5')
plt.plot(point , [normal_pdf(x,-1,1) for x in point] , color='black' , label='mu=-1 sigma=1')
plt.legend()
plt.title("Normal PDF")
plt.show()
```



Name:

10. Do the same as above question for Normal CDFs using the same values of μ and σ.

```
import math
from matplotlib import pyplot as plt
def normal_cdf(x:float,mu:float=0,sigma:float=1) -> float:
   return (1+math.erf((x-mu) / math.sqrt(2) /sigma))/2
point=[ x /10.0 for x in range(-50,50)]
plt.plot(point , [normal_cdf(x) for x in point] , color='red' , label='mu=0 sigma=1')
plt.plot(point , [normal_cdf(x,0,2) for x in point] , color='blue' , label='mu=0 sigma=2')
plt.plot(point , [normal_cdf(x,0,0.5) for x in point] , color='green' , label='mu=0 sigma=0.5')
plt.plot(point , [normal_cdf(x,-1,1) for x in point] , color='black' , label='mu=-1 sigma=1')
plt.legend()
plt.title("Normal CDF")
plt.show()
```



11. What are random variables? Give two examples.

A random variable is a variable that is subject to random variations so that it can take on multiple different values, each with an associated probability.
A typical example of a random variable is the outcome of a coin toss.
Rolling of a dice

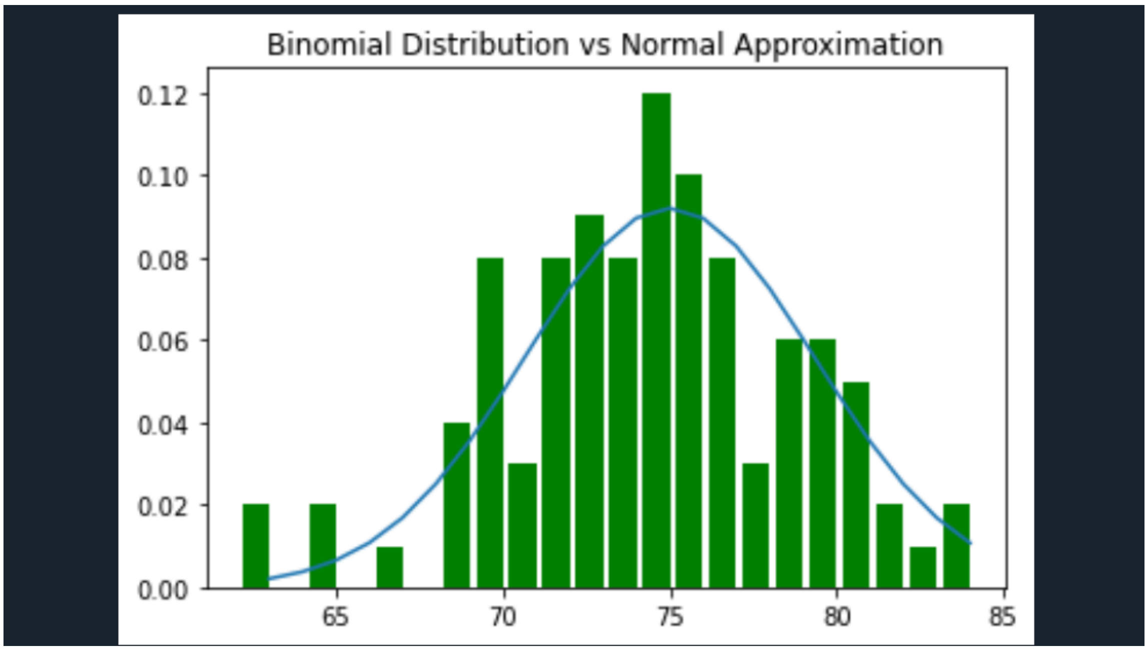
12. What are independent events? Give two examples of the same.

Independent events are those events whose occurrence is not dependent on any other event.
Examples :
You flip a coin and get a head and you flip a second coin and get a tail. The two coins don't influence each other.
The probability of rain today and the probability of my garbage being collected today; The garbage will be collected, rain or shine.

13. Using the Binomial(n, p) distribution plot a histogram to show the actual binomial samples. Use a line chart to show the normal approximation. Plot both in the same graph. Take n=100, p=0.75 and number of points should be 100.

```
from collections import Counter
from matplotlib import pyplot as plt
import random
import math
def normal_cdf(x:float,mu:float=0,sigma:float=1) -> float:
   return (1+math.erf((x-mu) / math.sqrt(2) /sigma))/2
def bernouli_trial(p:float) -> int:
   return 1 if random.random() < p else 0
def binomial(n:int , p:float) -> int:
   return sum(bernouli_trial(p) for _ in range(n))
def binomial_histogram(p:float,n:int,num_points:int) -> None:
   data=[binomial(n,p) for _ in range(num_points)]
   histogram=Counter(data)
   plt.bar([x-0.4 for x in histogram.keys()],
         [v/num_points for v in histogram.values()],
         0.8,
         color='green')
   mu=p*n
   sigma=math.sqrt(n*p*(1-p))

   xs=range(min(data),max(data)+1)
   ys=[normal_cdf(i+0.5,mu,sigma)-normal_cdf(i-0.5,mu,sigma) for i in xs]
   plt.plot(xs,ys)

   plt.title('Binomial Distribution vs Normal Approximation')
   plt.show()
binomial_histogram(0.75, 100, 100)
```

Name:


Binomial Distribution vs Normal Approximation
0.12
0.10
0.08
0.06
0.04
0.02
0.00
65
70
75
80
85

Name: