Chapter 2:

4- Random Number Generators using Python codes

a) In Linear congruential generator (in Page 34) explain the formula and elaborate the sensitivity of the random variable generator with respect to the parameters. In other words, do you have a better set of random variables for into higher a, c, and m? Set the parameters to (a =3, c= 4, m=5, and x=3) and generate the first 100 pseudorandom values in Python. Explain the role of each Python function you use.

In [ ]:
```
import numpy as np

a,c,m,x =  3,4,5,3

for i in range(100):
    x=np.mod((a*x+c),m)
    print(x)
```

b) Explain the Learmonth-Lewis generator method and its parameters (similar to Problem 5 above). Set the parameters equal to a = 60, c = 0, and m = 2^(31) -1, and generate the first uniform random variables in the range of [0, 1] in Python. Explain the commands and the role of each function you use.

In [ ]:
```
import numpy as np

a,c,m,x=60,0,2**(31)-1,0.1

for i in range(100):
    x=np.mod((a*x+c),m)
    u=x/m
    print(u)
```

c) Explain about Lagged Fibonacci generator method and provide a list of benefits over other methods. Explain what m is in this method and how it helps to improve the generator. Implement a simple example of additive LFG in Python using the following parameters: x0 = x1 = 1 and m = 2^(32).

In [ ]:
```
import numpy as np

x0,x1=1,1
m=2**64
for i in range(100):
    x=np.mod((x0+x1),m)
    x0=x1
    x1=x
    print(x)
```

5- Using Python function for random number generators do the following.

a) Generate 20 Pseudorandom numbers between 0 and 1.

In [ ]:
```
import random

for i in range(20):
    print('%05.4f'%random.random(),end=' ')
```

b) Do part (a) using random.seed() function to generate the same list of random variables.

In [ ]:
```
import random
random.seed(1)
for i in range(20):
    print('%05.4f'%random.random(),end=' ')
```

c) Make a random number generator using a “for-loop” that generates integer random numbers 'n' number of times

In [ ]:
```
import random
n=int(input())
for i in range(n):
    print(random.randint(0,100))
```

d) Use a random generator to select three “same” sets of random countries from the following list. {Canada, USA, Italy, China, India, South Africa, Spain, Brazil, Mexico}

In [ ]:
```
import random

contries=['Canada', 'USA', 'Italy', 'China', 'India', 'South Africa',
n=3
s=random.sample(contries,n)
for i in range(n):
    print(s)
```

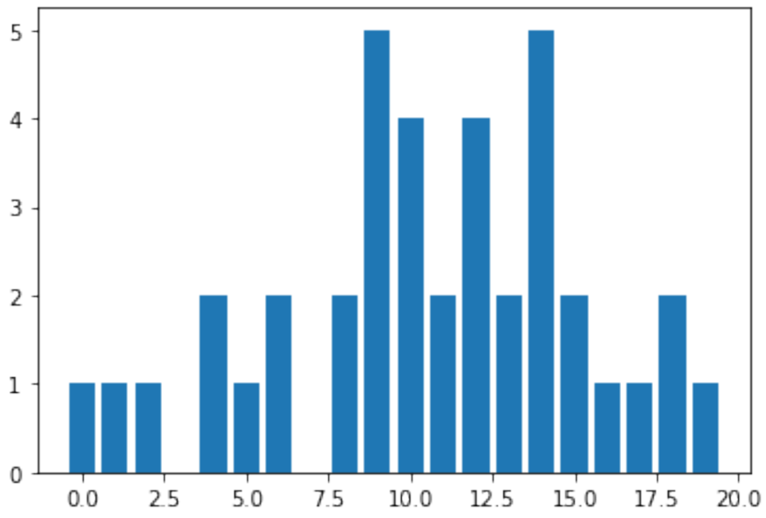
6- Testing Uniformity of Final Exam Grades. A random sample of final examination grades for a college course follows.

55 85 72 99 48 71 88 70 59 98 80 74 93 85 74 82 90 71 83 60 95 77 84 73 63 72 95 79 51 85 76 81 78 65 75 87 86 70 80 64

In the book we had test of Uniformity. Use the above dataset and methods in page 45- 47to determine if these values are uniformly distributed. Setup your Null and Alternative hypothesis carefully.

In [7]:
```
import numpy as np
data=np.array([55,85,72,99,48,71,88,70,59,98,80,74,93,85,74,82,90,71,8

data=(data-np.min(data))/(np.max(data)-np.min(data)) #adjust array int
N = 40
s = 20
Ns = N / s
S = np.arange(0, 1, 0.05)
counts = np.empty(S.shape, dtype=int)
V = 0
print(f's={s}')
print(f'S={S}')
for i in range(0, 20):
    counts[i] = len(np.where((data >= S[i]) & (data < S[i] + 0.05))[0]
    V = V + (counts[i] - Ns) ** 2 / Ns

print("R = ", counts)
print("V = ", V)
alpha=30.14 #ci-square number for df 19
if V < alpha:  # null hypothesis: data is uniformed
    print("The null hypothesis can be rejected")
else:
    print("The null hypothesis cannot be rejected")

import matplotlib.pyplot as plt

Ypos = np.arange(len(counts))

plt.bar(Ypos, counts)
plt.show()
```

```
s=20
S=[0.   0.05 0.1  0.15 0.2  0.25 0.3  0.35 0.4  0.45 0.5  0.55 0.6  0
.65
 0.7  0.75 0.8  0.85 0.9  0.95]
R =  [1 1 1 0 2 1 2 0 2 5 4 2 4 2 5 2 1 1 2 1]
V =  20.5
The null hypothesis can be rejected
```



Chapter 3:

7- According to the above definition, solve the following two problems.

a) Toss two fair dice. Let A be the event that the sum of the two dice is 9. Let B be the event that the first die is a 4. Are A and B independent?

```
In [ ]: # Total possibility: 36
        # Event A: [4,5] [5,4] [3,6] [6,3]
        # P(A) = 4/36 = 1/9
        # Event B: [4,1] [4,2] [4,3] [4,4] [4,5] [4,6]
        # P(B) = 6/36 = 1/6
        # Event A & B: [4,5]
        # P(A & B) = 1/36
        # P(A)*P(B) = 1/54 != P(A & B) = 1/36
        # A and B is not independent
```

b) Toss two fair dice. Suppose A is the event that the sum of the two dice equals 7, and B is the event that the first die shows a 4. Are A and B independent?

In [ ]:
```
# Total possibility: 36
# Event A: [1,6] [6,1] [2,5] [5,2] [3,4] [4,3]
# P(A) = 6/36 = 1/6
# Event B: [4,1] [4,2] [4,3] [4,4] [4,5] [4,6]
# P(B) = 6/36 = 1/6
# Event A & B: [4,3]
# P(A & B) = 1/36
# P(A)*P(B) = 1/36 == P(A & B) = 1/36
# A and B is independent
```

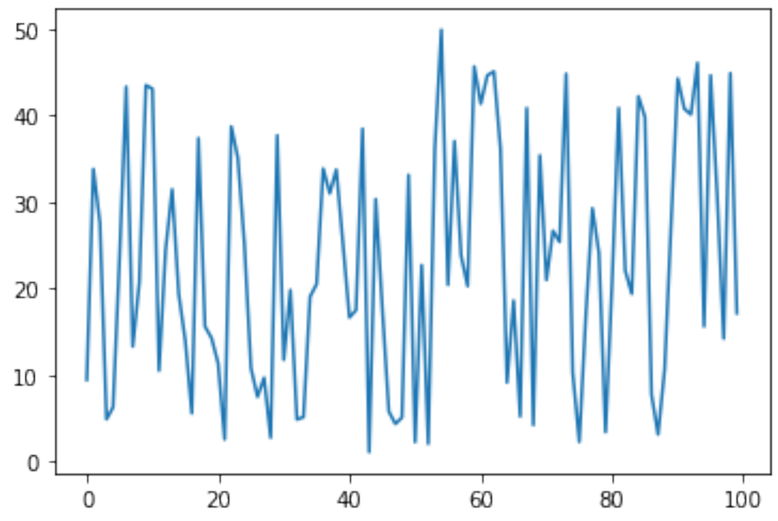
8- Using Python functions, generate a uniform distribution of random numbers contained in interval [0, 50] and do the following.

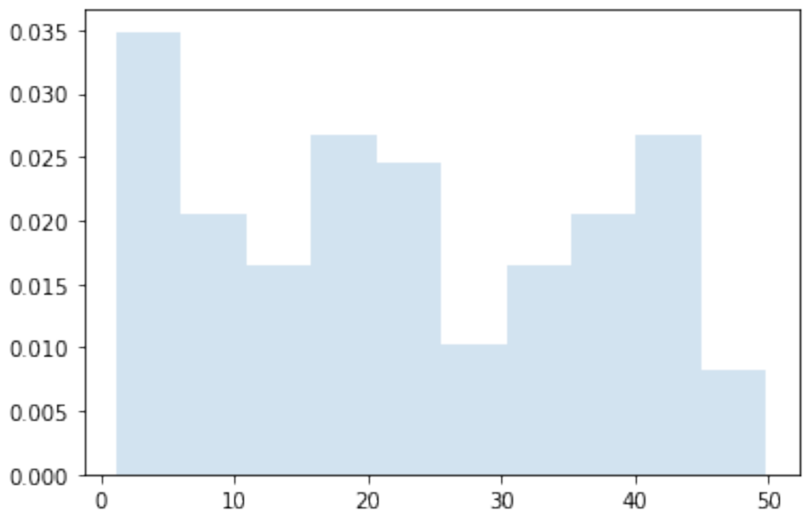
• Draw a diagram in which we report the values of the 50 random numbers that we have generated.

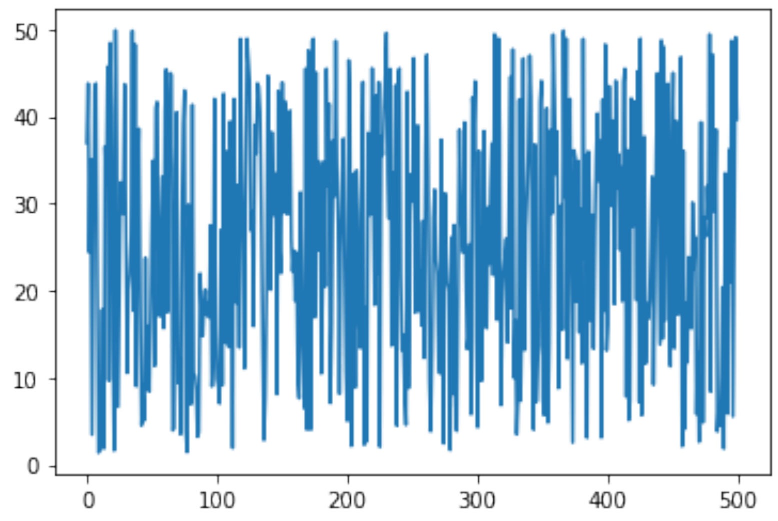
• Draw a graph of the probability density function.

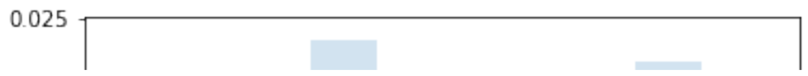
• Repeat the same graph by replicating 500 samples.

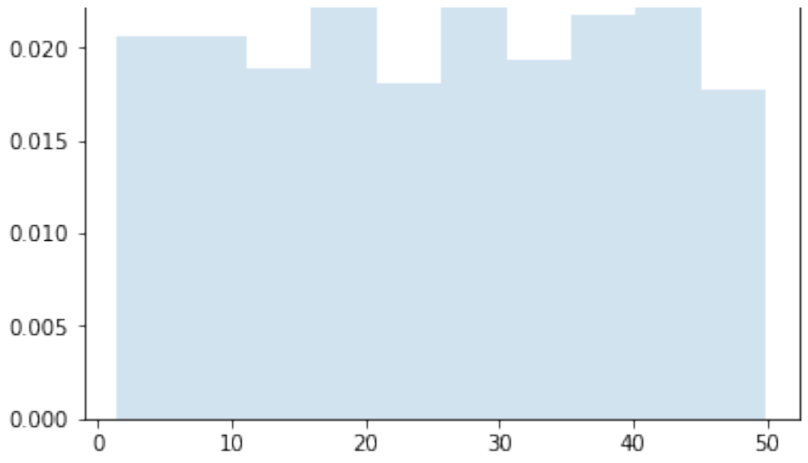
In [8]:
```
import numpy as np
import matplotlib.pyplot as plt

a,b,N=1,50,100

x1=np.random.uniform(a,b,N)

plt.plot(x1)
plt.show()

#probability density function

plt.figure()
plt.hist(x1,
         density=True,
         histtype='stepfilled',
         alpha=0.2)
plt.show()

#500 samples
N=500
x2=np.random.uniform(a,b,N)
plt.plot(x2)
plt.show()
plt.figure()
plt.hist(x2,
         density=True,
         histtype='stepfilled',
             alpha=0.2)
plt.show()
```











9- Using Python functions and libraries, evaluate the probability density function of a binomial distribution X as the probability of success with N = 500, n = 25, and p = 0.1.

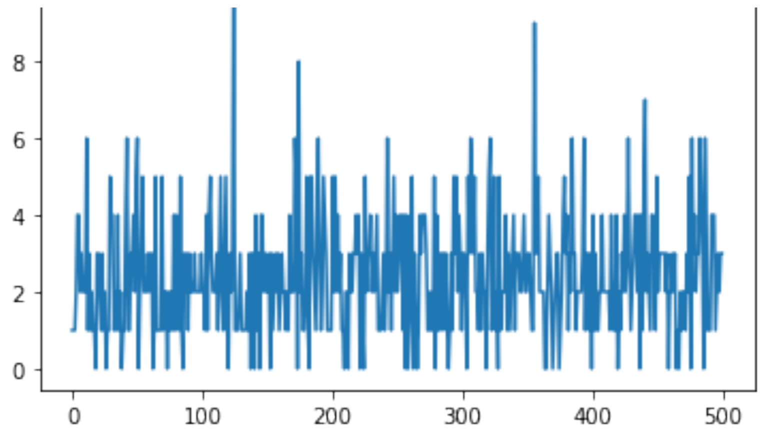
• Generate the probability distribution

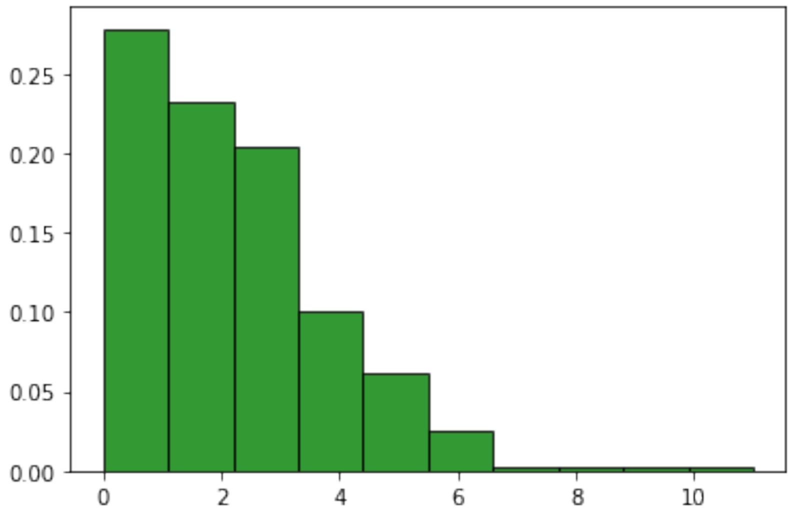
• Plot the data generated.

• Plot the histogram of the return values.

• Find probability of having 5 successes (that is p (X = 5))

In [10]:
```
import matplotlib.pyplot as plt
N,n,p=500,25,0.1
P1=np.random.binomial(n,p,N)

plt.plot(P1)
plt.show()

plt.figure()
plt.hist(P1,
         density=True,
         alpha=0.8,
         histtype='bar',
         color='green',
         ec='black')
plt.show()
c=0
for i in P1:
    if i ==5:
        c+=1
p=c/len(P1)
print(p)
```





```
0.068
```

10- Using Python functions and libraries, generate a normal probability distribution Y with mu= 25, sigma = 3.

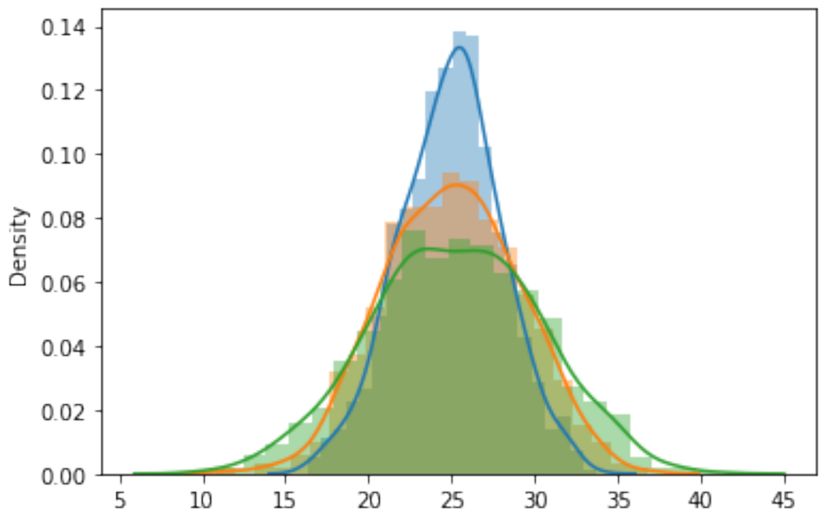
- Generate the probability distribution
- Plot the data generated. Generate the same plot by changing Sigma = 4 and 5.
- Plot the histogram of the return values.
- Find P (Y > 20)
- Find P (Y = 22.5)

In [21]:
```python
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

mu = 25
sigma =3
```

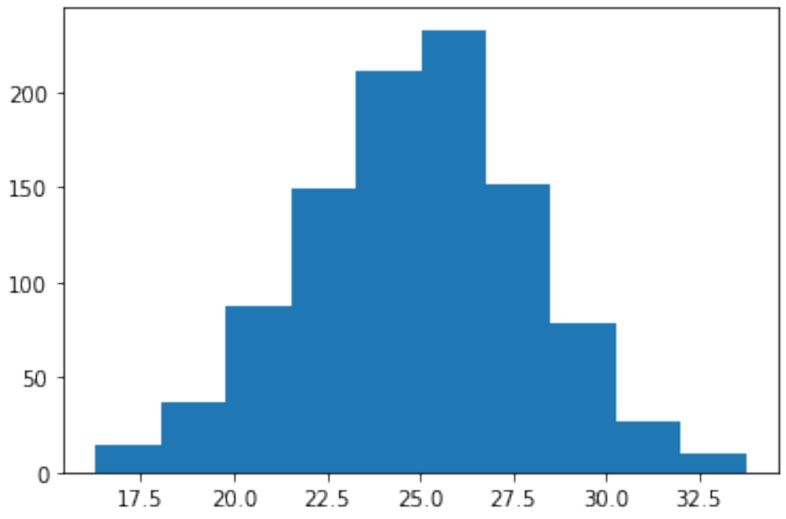
```
P1 = np.random.normal(mu, sigma, 1000)


mu = 25
sigma =4
P2 = np.random.normal(mu, sigma, 1000)

mu = 25
sigma =5
P3 = np.random.normal(mu, sigma, 1000)

Plot1 = sns.distplot(P1)
Plot2 = sns.distplot(P2)
Plot3 = sns.distplot(P3)
plt.figure()
plt.show()
# I didn't Find P (Y > 20) and Find P (Y = 22.5)
plt.hist(P1)
plt.show()

P1_location=np.where(P1 > 20)[0]
print(f'P1_Y>20_POSIBILITY = {len(P1_location)/len(P1)}')

P1_location=np.where(P1 == 22.5)[0]
print(f'P1_Y==22.5_POSIBILITY = {len(P1_location)/len(P1)}')


plt.hist(P2)
plt.show()

P2_location=np.where(P2 > 20)[0]
print(f'P2_Y>20_POSIBILITY = {len(P2_location)/len(P2)}')

P2_location=np.where(P2 == 22.5)[0]
print(f'P2_Y==22.5_POSIBILITY = {len(P2_location)/len(P2)}')

plt.hist(P3)
plt.show()
P3_location=np.where(P3 > 20)[0]
print(f'P3_Y>20_POSIBILITY = {len(P3_location)/len(P3)}')

P3_location=np.where(P3 == 22.5)[0]
print(f'P3_Y==22.5_POSIBILITY = {len(P3_location)/len(P3)}')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

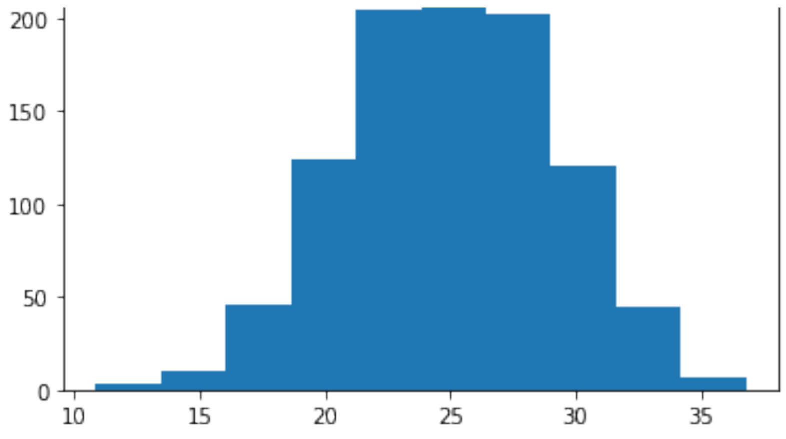
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557:
FutureWarning: `distplot` is a deprecated function and will be remove
d in a future version. Please adapt your code to use either `displot`
(a figure-level function with similar flexibility) or `histplot` (an
axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



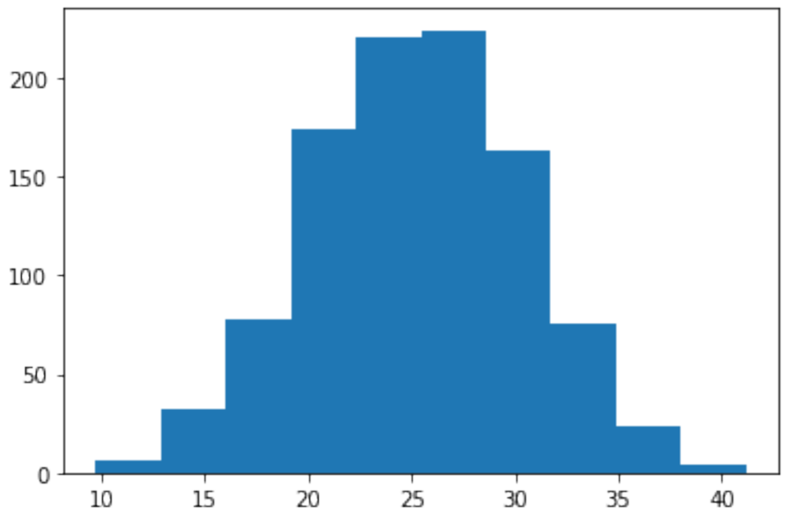
```
<Figure size 432x288 with 0 Axes>
```



```
P1_Y>20_POSIBILITY = 0.948
P1_Y==22.5_POSIBILITY = 0.0
```



```
P2_Y>20_POSIBILITY = 0.888
P2_Y==22.5_POSIBILITY = 0.0
```



```
P3_Y>20_POSIBILITY = 0.845
P3_Y==22.5_POSIBILITY = 0.0
```