Generating random numbers from scratch

In [1]:

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

In [2]:

```
def generate_seed():
    """
     seed: previously value number generated by the generator
     output: returned by the current system clock as the seed
    """

    import time
    t = time.perf_counter() # current system clock
    return int(10**9*float(str(t-int(t))[0:]))
```

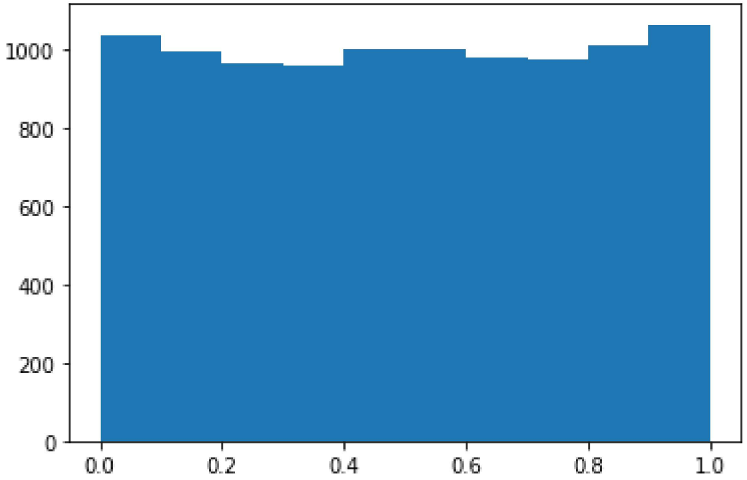
In [3]:

```
def pseudo_uniform(mult=16807,
                   mod=(2**31)-1,
                   seed=123456789,
                   size=1):
    """
    a pseudo random genarator with multipliers and modulus
    mult: Lehmer random number generator parameter (7**5)
    """
    u = np.zeros(size)
    x = (seed*mult+1)%mod
    u[0] = x/mod

    for i in range(1,size):
        x = (x*mult+1)%mod
        u[i] = x/mod
    return u
```

In [4]:

```
ls = pseudo_uniform(size=10000, seed=3)
plt.hist(ls)
plt.show()
```



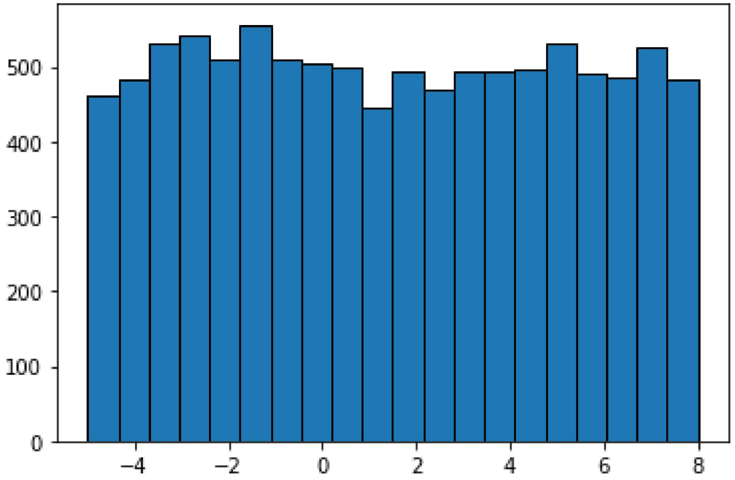
In [5]:

```
def uniform(low=0, high=1, seed=123456789, size=1):
    """
    return uniformly generated random number between low and high boundaries
    """

    return low + (high-low)*pseudo_uniform(seed=seed, size=size)
```

In [6]:

```python
ls = uniform(low=-5, high=8, size=10000)
plt.hist(ls, bins=20, edgecolor='k')
plt.show()
```



In [7]:
```python
for i in range(5):
    print(uniform(low=-5, high=7, size=10, seed=generate_seed()))
```

```
[ 4.91561602  2.75840994 -1.40408263 -0.4167627  -2.53076206 -0.5178853
  1.90180808  1.68834944  2.08903948  4.48656981]
[ 1.79837236  3.24430683  5.06490531  3.86359348 -2.5844439  -2.74862073
 -2.06858969  3.21302248 -4.73121184  0.52252602]
[ 6.95681041  1.11257845 -2.89397816  2.90902086 -2.08634831  4.74397474
 -2.01653645  2.07190798  4.55738747  6.01127194]
[-2.26074333  1.68682326  0.43858382 -2.72180213  4.6716758   6.85511124
 -4.14543053  5.74912886 -4.39126833  1.95312498]
[ 0.39970548  3.8500265  -2.60461658  6.20911846 -1.34599688  3.83045125
  4.39419784 -0.7168568   5.78770168 -4.09788391]
```

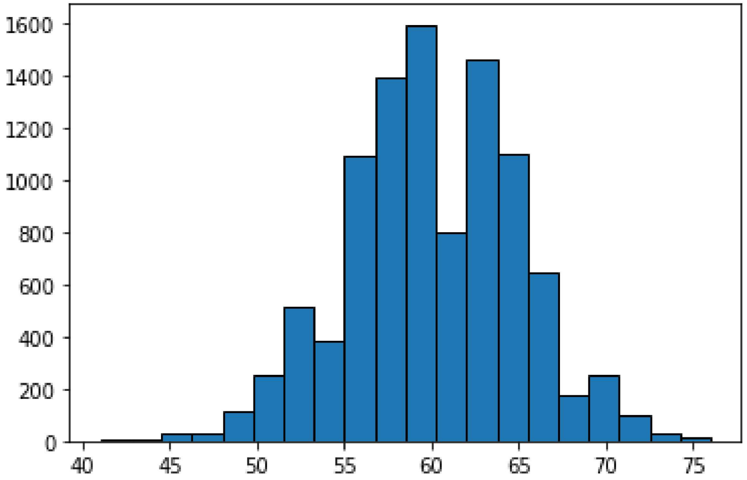
Visualization using statistical methods

Binomial distribution

In [8]:
```python
def binomial_distribution(n=100, p=0.5, size=1):
    """
    binomial distribution is the discrete probability distribution of a specific number of successes in a sequence o
    each asking a yes-no question, and each with its own boolean-valued outcome:
    success with probability p or failure with probability q = 1 − p.
    """

    b = []
    for _ in range(size):
        u = uniform(size=n, seed=generate_seed()) # Binomial random variate generator from the Uniform generator
        y = (u<=p).astype(int)
        b.append(np.sum(y))
    return b
```

In [9]:
```
ls = binomial_distribution(p=0.6,size=10000)
plt.hist(ls, bins=20, edgecolor='k')
plt.show()
```



Poission distribution

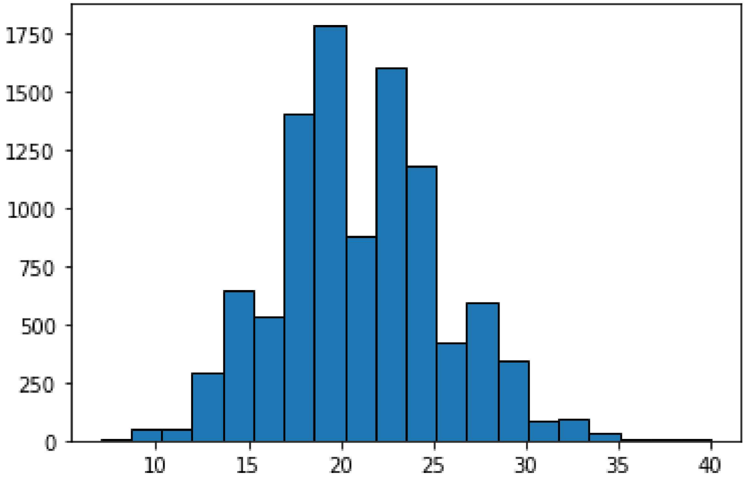
In [10]:

```python
def poisson_distribution(alpha, size=1):
    """
    poisson distribution is a discrete probability distribution that expresses the probability of a given number of
    in a fixed interval of time or space if these events occur with a known constant mean rate and independently of
    """

    poisson = []
    for k in range(size):
        u = uniform(size=5*alpha, seed=generate_seed()) # poission random variate generator from the Uniform generat
        x,p,i = 0,1,0
        while p>=np.exp(-alpha):
            p = u[i]*p
            x+=1
            i+=1
        poisson.append(x)
    return poisson
```

In [11]:

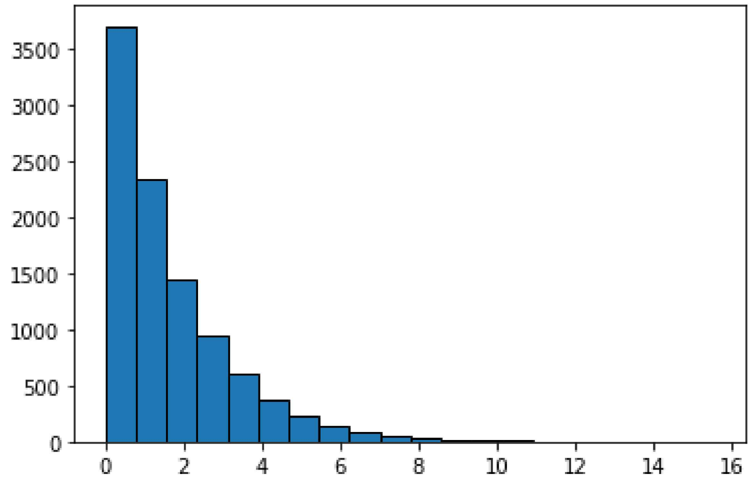
```python
ls = poisson_distribution(20, 10000)
plt.hist(ls, bins=20, edgecolor='k')
plt.show()
```



Exponential distribution

In [12]:
```python
def expo_distribution(lambda_val, size=1):
    u = uniform(size=size, seed=generate_seed())
    return -(1/lambda_val)*(np.log(1-u))
```

In [13]:
```python
ls = expo_distribution(0.6, 10000)
plt.hist(ls, bins=20, edgecolor='k')
plt.show()
```



Predicting the value of pi using generated number

In [14]:
```python
def pi_estimator(samples):
    points_inside_circle= 0
    total_num_points = 0
    X,Y = pseudo_uniform(size=2*samples).reshape(2,-1)

    for x,y in zip(X,Y):
        distance = x**2 + y**2
        if distance <= 1:
            points_inside_circle +=1
        total_num_points += 1
    return 4*points_inside_circle/total_num_points
```

In [15]:

```
pi_pred = pi_estimator(10**7)
print('Predicted value of pi is', pi_pred)
```

```
Predicted value of pi is 3.1418544
```

In [16]:

```
pi_test = np.pi
error = abs(pi_test-pi_pred)/pi_test
print(error)
```

```
8.331647004201937e-05
```

In [ ]:

```
```