In [2]:
```
import numpy as np
```

In [334]:
```
#Qsn.1. Random numers using midsquare method
def midsquare(number,n):
    random_list=[]
    for i in range(1,n+1):
        p=number**2
        x=int(p/100)
        y=x%10000
        random_list.append(y)
        number=y
    print(random_list)
```

In [335]:
```
#when seed is 4444, 40 random numbers
midsquare1(4444,40)
```

```
[7491, 1150, 3225, 4006, 480, 2304, 3084, 5110, 1121, 2566, 5843, 1406, 9768, 4
138, 1230, 5129, 3066, 4003, 240, 576, 3317, 24, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0]
```

In [336]:
```
#When seed is 5197, 10 random numbers
midsquare(5197,10)
```

```
[88, 77, 59, 34, 11, 1, 0, 0, 0, 0]
```

In [337]:
```
#I understand from this method is that it's not very good approach to generate ra
#after certain steps
```

In [338]:
```
#Qsn.2.Random numbers using Linear Congruential Generators
def LinConGen(Znot,n,a,m,c):
    random=[]
    for i in range(1,n+1):
        Z=(a*Znot+c)%m
        random.append(Z)
        Znot=Z
    print(random)
```

In [339]:
```
LinConGen(5,20,8,34,2)
```

```
[8, 32, 20, 26, 6, 16, 28, 22, 8, 32, 20, 26, 6, 16, 28, 22, 8, 32, 20, 26]
```

In [340]:
```
#Yes this cycle has full cycle since numbers are repeated
```

In [341]:
```
#Qsn.3.Pseudo Random Number Generators
def PseRanNumGen(x1,x2,n):
    randomprng=[]
    for i in range(1,n+1):
        x3=(3*x2+5*x1)%100
        randomprng.append(x3)
        x1=x2
        x2=x3
    print(randomprng)
```

In [342]:
```
PseRanNumGen(33,46,14)
```

```
[3, 39, 32, 91, 33, 54, 27, 51, 88, 19, 97, 86, 43, 59]
```

In [343]:
```
#Qsn.4. Integration estimation using simulation
#Part a
def integration1(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.exp(np.exp(x))
    print(sum(y)/n)
```

In [344]:
```
integration1(100000)
```

```
6.306841893441574
```

In [345]:
```
#Part b
def integration2(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.exp((-2+4*x)+(-2+4*x)**2)
    print(sum(y)*4/n)
```

In [346]:
```
integration2(100000)
```

```
91.764628576959
```

In [347]:
```
#Part c
def integration3(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.exp(-(-1+1/x)**2)/x**2
    print(sum(y)*2/n)
```

In [348]:
```
integration3(100000)
```

```
1.7732523163441205
```

In [349]:
```
#Pard d
def integration4(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.random.uniform(0,1,n)
    I=np.exp((x+y)**2)
    print(sum(I)/n)
```

In [350]:
```
integration4(100000)
```

```
4.900877738959944
```

In [351]:
```
#Part e
def integration5(n):
        np.random.seed(42)
        x=np.random.uniform(0,1,n)
        y=np.random.uniform(0,1,n)
        I=np.exp(-(1/x-1)*(1+y))*(-(1-x)/x**3)
        print(sum(y)/n)
```

In [352]:
```
integration5(100000)
```

```
0.5013301153148381
```

In [353]:
```
#Qsn.5. Estimating covariance
def covariance(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.exp(x)
    z=x*y
    from statistics import mean
    print(mean(z)-mean(x)*mean(y))
```

In [354]:
```
covariance(100000)
```

```
0.14043338804712746
```

In [ ]:
```
#This approximation is equal to the exact value which is 1.4
```

In [355]:
```
#Qsn.6.Finding Correlation
#Part a
def correlation1(n):
    np.random.seed(42)
    x=np.random.uniform(0,1,n)
    y=np.sqrt(1-x**2)
    z=x*y
    from statistics import variance
    from statistics import mean
    print((mean(z)-mean(x)*mean(y))/(np.sqrt(variance(x))*np.sqrt(variance(y))))
```

In [356]:
```
correlation1(100000)
```

```
-0.921542820077953
```

In [357]:
```
#Part b
def correlation2(n):
    np.random.seed(42)
    x=(np.random.uniform(0,1,n))**2
    y=np.sqrt(1-x**2)
    z=x*y
    from statistics import variance
    from statistics import mean
    print((mean(z)-mean(x)*mean(y))/(np.sqrt(variance(x))*np.sqrt(variance(y))))
```

In [358]:
```
correlation2(100000)
```

```
-0.9128347266524167
```

In [5]:
```
def pi(n):
    i=0
    lst=[]
    x=2*(np.random.uniform(0,1,n))-1
    y=2*(np.random.uniform(0,1,n))-1
    if ((x**2+y**2)<=1):
        i=1
    for k in range(0,n+1):
        lst.append(i)
```

In [6]:
```
pi(10)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-6f43c650319d> in <module>()
----> 1 pi(10)

<ipython-input-5-41d25ab6b633> in pi(n)
      4     x=2*(np.random.uniform(0,1,n))-1
      5     y=2*(np.random.uniform(0,1,n))-1
----> 6     if ((x**2+y**2)<=1):
      7         i=1
      8     for k in range(0,n+1):

ValueError: The truth value of an array with more than one element is ambiguou
s. Use a.any() or a.all()
```

In [ ]:
```
```