Problem 1.1:

I defined the Rectangular Class:

```
###Q1.1

## Creating a Rectangular Class
class Rectangular:

## Creating a constructor
    def __init__(self,length,width):
        self.length=length
        self.width=width
## Creating an area method to return the area of rectangular
    def area(self):

        return self.length*self.width
## Creating a perimeter method to return the perimeter of rectangular
    def perimeter(self):

        return 2*self.length+2*self.width
```

I tested the program and printed the results below

```
# Test the Rectangular class
myRec = Rectangular(10,20)

print(myRec.area())


print(myRec.perimeter())

200
60
```

Problem 1.2

I used numpy to create length and width arrays:

```
###Q1.2

import numpy as np

# Define length and width
length=np.array([1,3,5,7,9,11,13,15,17,19])
width=np.array([2,4,6,8,10,12,14,16,18,20])
```

I tested the program and printed results:

```
# Use lenght and width as a argument in Rectangular function
test_rect=Rectangular(length,width)

# Print out the results to see test the code
print(test_rect.area())

print(test_rect.perimeter())
```

```
[  2  12  30  56  90 132 182 240 306 380]
[ 6 14 22 30 38 46 54 62 70 78]
```

Problem 2:

I defined class time and method addtime here:

```
class Time:

    ## Creating a constructer

    def __init__(self,hours,minutes,seconds):

        self.hours=hours
        self.minutes=minutes
        self.seconds=seconds

        self.timehour=self.hours
        self.timemin=self.minutes
        self.timesec=self.seconds


    ## Adding time to the time given in constructer
    def addTime(self,h,m,s):

        self.h=h
        self.m=m
        self.s=s

        totsec=self.seconds+self.s

        if totsec<60:
            self.timesec=totsec
            sec_excess=0
        if totsec>=60:
            self.timesec=totsec-60
            sec_excess=1

        totmin=self.minutes+self.m+sec_excess

        if totmin<60:
            self.timemin=totmin
            min_excess=0
        if totmin>=60:
            self.timemin=totmin-60
            min_excess=1

        self.timehour=self.hours+self.h+min_excess

    #### Test the class
        print(self.timehour,self.timemin,self.timesec)
```

Here I defined two other methods displaytime and displaysecond.

Displaytime method displays time in in order of hours, minutes,seconds and displaysecond displays the total seconds of the time given

```
def displayTime(self):

    print(self.hours,"hours",self.minutes,"minutes",self.seconds,"seconds")

### Displaying total seconds of the time given
def DisplaySecond(self):

    display_second= (self.timehour*3600)+(self.timemin*60)+self.timesec

    return display_second
```

I tested the program and printed the results

```
###Testing the program
if __name__=="__main__":

    time=Time(1,2,0)

    time.displayTime()
#time.addTime(1,10,10)

    time.addTime(1,38,27)

    time.DisplaySecond()

1 hours 2 minutes 0 seconds
2 40 27
```

Problem 3.1

I defined the LCG class:

setseeds help us to set the seeds

getseeds help us to get the seeds

init_generator initialize the function

formula includes LCG formula

```
class LCG:
    ###Creating a constructor
    def __init__(self,seed,multiplier,increment,modulus):

        self.seed=seed
        self.multiplier=multiplier
        self.increment=increment
        self.modulus=modulus


    ### Creating setseed method
    def setseed(self, set):

        self.set=set
        return self.set
    ### Creating getseeds method
    def getseeds(self):

        return self.seed


    def initgen(self):

        value_zero=self.seed

    ### To create a uniform distributed sequence,  you only need to divide the  sequence above by M '''

        num1=value_zero /self.modulus
        return num1

    def formula(self,X):

        return (X*self.multiplier+self.increment)%self.modulus

    def nextrand(self):

        value_zero =self.seed
        value_one=self.formula(value_zero)

        num2= value_one/self.modulus
        return value_one
```

```
def seqrandom(self,leng):

    seq_list= list()

    value_zero=self.seed
    num1=self.initgen()

    seq_list.append(num1)

    if leng<1:

        return "Incorrect Length "
    if leng==1:

        return seq_list

    else:

        for input in range(1,leng):

            value_i=self.formula(value_zero)

            NUMi= value_i/self.modulus

            seq_list.append(NUMi)

            value_zero=value_i
    return seq_list
```

Here I tested the code and printed out results

```
if __name__=="__main__":

    zort1=LCG(0,1103515245,12345,2**32)


    zort1.setseed(1)

    print(zort1.getseeds())

    print(zort1.nextrand())

    print(zort1.initgen())


    print(zort1.seqrandom(2))
```

```
0
12345
0.0
[0.0, 2.8742942959070206e-06]
```

Here defined the class SCG, and I inherited it, since they are similar

```
class SCG(LCG):

    def initgen(self):
        if self.seed%4 is not 2:

            # print Error
            print("Error: Mod should be 2")

            return LCG.initgen(self)


    def formula(self,X):

        return ((( self.multiplier*((( self.multiplier*(X+1))+self.increment)%self.modulus))+self.increment)%self.modulus)%self.mod
```

Here tested the code and printed out results

```
if __name__=="__main__":



    zort2=SCG(3,1103515245,12345,2**32)

    print(zort2.getseeds())

    zort1.setseed(6)

    print(zort2.initgen())

    print(zort2.nextrand())

    print(zort2.seqrandom(5))
```

```
3
Error: Mod should be 2
6.984919309616089e-10
3731259426
Error: Mod should be 2
[6.984919309616089e-10, 0.8687515337951481, 0.3169204501900822, 0.7586911860853434, 0.5668
142472859472]
```

Problem 3.2

```
import math

##Define the class
class pointcord:

    x,y=0,0


    def __init__(self, xpoint,ypoint):

        self.xpoint=xpoint

        self.ypoint=ypoint



    def distance(self):

        dist=math.sqrt((self.ypoint-0)**2 + (self.xpoint-0)**2)

        return dist
```

Here I tested the program and printed out the result:

```
## Test the program
if __name__=="__main__":
    test_point= pointcord(0.9,0.6)
    print(test_point.distance())

1.0816653826391966
```

Problem 3.3

Imported modules and packages firstly

```
import time
from generator import LCG
from generator import SCG
from point import pointcord


time_start=time.time()
```

Here testing LCG method:

```
LCG_test= LCG(2,1103515245,12345,2**32)

LCG_X=LCG_test.seqrandom(2000)

LCG_test= LCG(5,1103515245,12345,2**32)

LCG_Y=LCG_test.seqrandom(2000)


value=0

for i in LCG_X:

    for j in LCG_Y:

        dot=pointcord(i,j)

        if dot.distance()<=1:

            value=value+1

ratio=value/1000000

print(ratio)

ratio_difference= abs(ratio- 0.78539816339)

print(ratio_difference)
```

Tested SCG method:

```
SCG_test= LCG(2,1103515245,12345,2**32)
SCG_X=LCG_test.seqrandom(2000)
SCG_test= LCG(5,1103515245,12345,2**32)
SCG_Y=LCG_test.seqrandom(2000)

value=0
for i in SCG_X:
    for j in SCG_Y:
        dot=pointcord(i,j)
        if dot.distance()<=1:
            value=value+1
ratio=value/1000000
print(ratio)
ratio_difference= abs(ratio- 0.78539816339)
print(ratio_difference)


print("Total seconds spent to get result is",time.time()-time_start)
```

It took totally 7.25 seconds to get the result for two estimates

```
print("Total seconds spent to get result is",time.time()-time_start)

3.129253
2.34385483661
3.16164
2.3762418366099998
Total seconds spent to get result is 7.259793281555176
```