# Assignment 3 Report

FE 520

Parth Parab
10444835
pparab@stevens.edu

Question 1 & 2 (Rectangular & Time Class)

---

Code:

*(main.py)*

```
import numpy as np

class Rectangular:
    """ Class Rectangular to calculate area and perimeter """

    def __init__(self, length, width):
        """ initialization of length and width variable """
        self.length, self.width = length, width

    def area(self):
        """ function to return area of rectangle """

        return self.length * self.width

    def perimeter(self):
        """ function to return perimeter of rectangle """

        return (self.length + self.width) * 2

class Time:
    """ Class Time for simple time operations """

    def __init__(self, hours, minutes, seconds):
        """ initialization of hours, minutes and seconds """

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

    def addTime(self, val):
        """ function to add two time objects """

        newHours, newMinutes, newSeconds = self.hours + \
```

```
                val.hours, self.minutes + val.minutes, self.seconds +
val.seconds

            # minutes and seconds correction

            newMinutes, newSeconds = (
                newMinutes + newSeconds // 60), (newSeconds % 60)
            newHours, newMinutes = (newHours + newMinutes // 60), (newMinutes
% 60)

            return Time(newHours, newMinutes, newSeconds)

        def displayTime(self):
            """ display time """

            print(self.hours, "hour(s)", self.minutes,
                  "minute(s)", self.seconds, "second(s)")

        def displaySecond(self):
            """ display time in seconds"""

            print(self.hours * 3600 + self.minutes * 60 + self.seconds,
"seconds")


    if __name__ == "__main__":

        # REACTANGULAR TEST 1
        myRec = Rectangular(10, 20)
        print("REACTANGULAR TEST 1\n")
        print("Area: ", myRec.area())
        print("Perimeter: ", myRec.perimeter())

        # REACTANGULAR TEST 2
        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])

        myRec = Rectangular(length, width)
        print("\nREACTANGULAR TEST 2\n")
```

```
print("Area: ", myRec.area())
print("Perimeter: ", myRec.perimeter())

# TIME TEST 1

t1 = Time(2, 50, 10)
t2 = Time(1, 20, 5)

print("\nTIME TEST 1\n")
res = t1.addTime(t2)
res.displayTime()
res.displaySecond()
```

Output:

```
REACTANGULAR TEST 1

Area:  200
Perimeter:  60

REACTANGULAR TEST 2

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

TIME TEST 1

4 hour(s) 10 minute(s) 15 second(s)
15015 seconds
```

Question 3 (Uniform Distributed Random Number Generator)

---

*generator.py*

Code:

```
class LCG:
    """ Linear Congenital Generator algorithm """

    def __init__(self, seed, multiplier, increment, modules):
        """ initialize variables """

        self.seed, self.multiplier, self.increment, self.modules = seed,
multiplier, increment, modules
        self.initGen()

    def getSeed(self):
        """ get seed value """

        return self.seed

    def setSeed(self, val):
        """ set seed value """

        self.seed = val

    def initGen(self):
        """ initialize the generator (start from the seed) """

        self.start = self.seed

    def rand(self):
        """ generate next random number """

        self.start = (self.multiplier * self.start +
                      self.increment) % self.modules

        return self.start / self.modules
```

```
        # ref: https://stackoverflow.com/questions/19140589/linear-
congruential-generator-in-python

        def rand_array(self, length):
            """ returns a sequence (list) of random number """
            return [self.rand() for val in range(length)]


    class SCG(LCG):

        """ Recursive Congruential Generator """

        def __init__(self, seed, multiplier, increment, modules):

            # seed value check
            if (seed % 4 != 2):
                raise ValueError(
                    "the seed of this generator did not satisfy X to base 0
mod 4 = 2")

            # inherit LCG init
            super().__init__(seed, multiplier, increment, modules)

        def rand(self):
            """ generate next random number """

            self.start = (self.start * (self.start + 1)) % self.modules
            return self.start / self.modules


    if __name__ == "__main__":

        # LCG TEST 1
        r1 = LCG(1, 1103515245, 12345, 2**32)
        print("\nLCG TEST 1\n")
        print(r1.rand_array(3))

        # LCG TEST 2
```

```
r2 = LCG(1, 1140671485, 128201163, 2**24)
print("\nLCG TEST 2\n")
print(r2.rand_array(3))

# SCG TEST 1
r1 = SCG(6, 1103515245, 12345, 2**32)
print("\nSCG TEST 1\n")
print(r1.rand_array(3))

# SCG TEST 2
r2 = SCG(6, 1140671485, 128201163, 2**24)
print("\nSCG TEST 2\n")
print(r2.rand_array(3))
```

Output:

```
LCG TEST 1

[0.25693503906950355, 0.5878706516232342, 0.15432575810700655]

LCG TEST 2

[0.6307034492492676, 0.6911689639091492, 0.0895453691482544]

SCG TEST 1

[9.778887033462524e-09, 4.2049214243888855e-07, 0.0007598293013870716]

SCG TEST 2

[2.5033950805664062e-06, 0.00010764598846435547, 0.19451630115509033]
```

*point.py*

Code:

```
import math

class Point:
    """ Point Class to represent the points in rectangular coordinate """

    def __init__(self, x, y):
        """ initialize x and y coordinates """

        self.x, self.y = x, y

    def distance(self):
        "calculate distance between origin and point."

        # distance calculated using Euclidean formula

        return math.sqrt(self.x ** 2 + self.y ** 2)


if __name__ == "__main__":

    p = Point(4, 3)

    print("x co-ordinate:", p.x)
    print("y co-ordinate:", p.y)
    print("Distance:", p.distance())
```

Output:

```
x co-ordinate: 4
y co-ordinate: 3
Distance: 5.0
```

*MCTest.py*

Code:

```
import generator
import point
import math
import time


def test(gen):
    num = 10 ** 7

    # selecting generator

    if (gen == "LCG"):
        rand = generator.LCG(1, 1103515245, 12345, 2**32)
    elif gen == "SCG":
        rand = generator.SCG(6, 1103515245, 12345, 2**32)

    # generating 2 * 10 ** 7 random numbers
    x, y = rand.rand_array(num), rand.rand_array(num)

    # re-scale points into [-1, 1]
    x, y = [(i-0.5) * 2 for i in x], [(i-0.5) * 2 for i in y]

    # pairing
    points = []

    for i, j in zip(x, y):
        points.append(point.Point(i, j))

    # calculate distance from origin
    count = 0

    for p in points:
        if p.distance() < 1:
```

```
            count += 1

    ratio = count/num

    # actual number
    real = math.pi / 4
    diff = abs(real - ratio)

    print("Estimate Ratio using", gen, ":", ratio)
    print("Theoretical value: ", real)
    print("Difference: ", diff)


if __name__ == "__main__":

    for gen in ("LCG", "SCG"):

        start = time.time()
        test(gen=gen)
        runtime = time.time() - start

        print("Time consumed: ", runtime, "seconds\n")
```

Output:

```
Estimate Ratio using LCG : 0.7854845
Theoretical value:  0.7853981633974483
Difference:  8.633660255175091e-05
Time consumed:  32.988460063934326 seconds

Estimate Ratio using SCG : 0.7855226
Theoretical value:  0.7853981633974483
Difference:  0.00012443660255168076
Time consumed:  29.988184928894043 seconds
```