Shreyan Wankavala 112634232

Problem 1.1

Source: https://www.geeksforgeeks.org/monte-carlo-integration-in-python/

For this problem, an equation is given for a heart as well as a circle, and they are both overlapping. The question is asking for me to write an algorithm which will find the area of the overlapped segment between the heart and the circle.

For this problem, I will use the Monte Carlo method since it will be much easier filling up specific parts of the heart and circle with randomly generated points when finding the area. I need to find the overlap between the two, so it seems easier to do it with this method.
I can iterate the program multiple times and sum all of my answers, dividing by the number of iterations essentially giving me the average value of the areas. This way, I can guarantee that my estimate is very accurate even though I am generating random points since I will be taking the average of a lot of areas which will hopefully take care of any deviations. The iterations are shown below.

```
number = 0
for i in range(4000):
```

I then put in the bounds for the heart and circle into my program, both of which were located at x = 0 going out to Radical 2.

```
# Upper and Lower bounds for integration. Going from -Rad2 to Rad2
a = -1 * math.sqrt(2)
b = math.sqrt(2)
N = 1000
```

To begin the Monte Carlo process, random values were filled into an array. These will be the dots as demonstrated in the lecture.

```
# fill ar with random values within the interval [a,b]
for i in range (len(ar)):
  ar[i] = random.uniform(a,b)
```

Both equations for the Heart and Disc were put into the program.

```
# Function of the heart as a method
def fHeart(x):
  return math.sqrt(2-(x**2)) + math.sqrt(abs(x))

def fDisc(x):
  return math.sqrt(2-(x**2)) + 1.35
```

Calculations were then done to see where they would overlap. When run, the final output was given.

```
# iterates and sums up values of different functions
# of x
for i in ar:
  integral += fHeart(i)
  discIntegral += fDisc(i)

# we get the answer by the formula derived adobe
ans = (b-a)/float(N)*(integral + discIntegral)
```

```
The value calculated as the overlapping of both figures is 12.343848332303356.
```

The final answer with four degrees of accuracy can be written as 12.3438 as the area that is overlapping between both figures.

At the end, the number of iterations put in to solidify the final answer is what caused the program to take longer to run. This number can be made lower, but the final value returned will not be as accurate when this happens.

Problem 1.2

Source: https://personal.math.ubc.ca/~pwalls/math-python/roots-optimization/bisection/

For this problem, an equation is given for which I must find the roots within the interval [-1, 2]. The graph is already given, so I can use either the bisection method or Newton's method to find the roots since I can eyeball them to start each method.

For this problem, I will use the bisection method. This is because using Newton's method requires finding the derivative of the equation, and that seems like a very complicated task given the equation. Before I start the bisection method, I must first find estimates of all six roots and put them in self contained intervals.

```
#guess 1: -0.8
#range: [-0.9, -0.7]
x1 = bisectionMethod(function,-0.9,-0.7,1000)

#guess 2: 1.25
#range: [1.20, 1.30]
x2 = bisectionMethod(function,1.2,1.3,1000)
#guess 3: 1.4
#range: [1.35, 1.45]
x3 = bisectionMethod(function,1.35,1.45,1000)
#guess 4: 1.7
#range: [1.65, 1.75]
x4 = bisectionMethod(function,1.65,1.75,1000)
#guess 5: 1.8
#range: [1.75,1.85]
x5 = bisectionMethod(function,1.75,1.85,1000)
#guess 6: 1.95
#range: [1.9, 2]
x6 = bisectionMethod(function,1.9,2,1000)
```

Each root has a guess as well as the range that will be put into the bisectionMethod function.

Now for the bisection method, I will be following the formula given in the lecture notes exactly in my code.

Bisection Method

Given initial interval $[a, b]$ such that $f(a)f(b) < 0$
while $(b - a)/2 >$ TOL
    $c = (a + b)/2$
    if $f(c) = 0$, **stop**, **end**
    if $f(a)f(c) < 0$
        $b = c$
    else
        $a = c$
    end
end
The final interval $[a, b]$ contains a root.
The approximate root is $(a + b)/2$.

The method will take in the function, interval, and iterations as parameters.
It will first check if f(a)f(b) < 0. If it does, then the method fails.
The first c value will be the average value of a and b which is our initial guess (since the intervals where made by adding and subtracting one value to the guess)

```
#new range divide by 2
#N iterations which is inputted
for n in range(1,N+1):
    #newRange = c
    newRange = (a1 + b1)/2
    fNew = function(newRange)
```

Next, there will be a series of else if statements which will represent the else statements from the notes. More specifically, if f(a)f(c) < 0, then b = 0, else a = c, else there is no root.

```
#a = c
elif function(b1)*fNew < 0:
    a1 = newRange
    b1 = b1

#if f(c) = 0 then c is the exact root solution
elif fNew == 0:
    print("Found exact solution.")
    return newRange
    #after checking both sides not found
else:
    print("No point found")
    return None
```

After every iteration, the final range will be divided by two one last time and that number will be returned as the root for the inputted interval.

```
#return the final range / 2 as the root after N iterations have passed
return (a1 + b1)/2
```

Finally, I repeated the process six times for each root as shown in the first image. The values of the roots were printed out to be

```
Found exact solution.
The first root is -0.8242859529552095
The second root is 1.2691963654232779
The third root is 1.4142931303740 67
The fourth root is 1.6955940658800746
The fifth root is 1.8067232403094708
The sixth root is 1.9482779886864665
```

Since the question asks for 4 digits of accuracy, they can be simplified.

X_1: -0.8243

X_2: 1.2692

X_3: 1.4143

X_4: 1.6956

X_5: 1.8067

X_6: 1.9483

The program runs very quickly even with 1000 iterations of each method call. The print statements could be condensed, but the time added to print them all separately is negligible so only space is wasted.