A L E X M A L A V E

Assignment: Homework #3

**Problem 1:**
*mcintersection.py*

My solution in *mcintersection.py* contains two functions, **monteCarlo** and **main**, which attempt to find the area bounded by two unit circles with centers offset by a given distance. **monteCarlo** takes two parameters (*dist* [distance], and *numDarts* [number of darts]), and checks if the given distance between unit circles is small enough for them to overlap. If so, two variables are created: **leftPoint** and **rightPoint**. **rightPoint** refers to the rightmost point of the leftmost circle, and **leftPoint** refers to the leftmost point of the rightmost circle. To calculate **leftPoint**, it finds the coordinates of the leftmost point of the leftmost circle, which is -1 (cos of pi is -1) – the distance given (the offset). Next, it *adds* double the distance given, because both circles are offset by that distance at the same time. For **rightPoint**, it finds the coordinates of the rightmost point of the rightmost circle, which is 1 (cos of 0 is 1) + the distance given (the offset). Next, it *subtracts* double the distance given, because both circles are offset by that distance at the same time, as I mentioned before.

Then, it uses the *random* module in order to randomly choose a real number between 0 and 1. For the **x** variable, since the range is originally 0 to 1, multiplying it by the **rightPoint***2 and then subtracting **rightPoint** would end up giving me a range of **-rightPoint** to **+rightPoint**, which is the range I need in order to "throw" darts at the correct region. The function then assigns **y** to a range of -1 to 1, which covers the height of the region bound by the two unit circles. It uses the formula of a circle ***sqrt((x^2)+(y^2)) = r^2*** and plugs the **x** and **y** variables in to check if the formula is true for both circles, which have been offset by +- the given distance. If it is true, 1 is added to an accumulator variable (the number of darts in the region).

The distance between **rightPoint** and **leftPoint** gives me the smallest width for the rectangle I use in order to calculate the area of the region between the circles (the height is 2). The equation for the Monte Carlo approximation method is: (# of darts in the region/total darts)*area of rectangle (which is 2 [the height]* [**rightPoint – leftPoint**].

I use **main** to print out what **monteCarlo** returns, which is the estimation of the area of the region bounded by the two unit circles (if the given distances lets them overlap). The file expects two parameters (*distance* and *the number of darts*), and the code should be run as: *python3 mcintersection.py <distance> <number of darts>*

**Problem 2:**
*startree.py*

My solution in *startree.py* contains one function, **main**, which creates a tree like shape on the ouput of the terminal with asterisks. The function uses a variable **n**, which is provided by the user and denotes the amount of rows the tree will have, an empty string **s**, an accumulator variable that begins at 0, and a **spaces** variable that begins at **n** – 1 (the last row doesn't need a space).

The function iterates through code once per row given by the user. The function uses concatenation in order to redefine the variable **s**. The accumulator is used to increase the amount of times that asterisks are repeated depending on the row the loop is on. The variable **spaces** is used to determine the amount of spaces needed to make a nice tree shape. The code should be run as follows: *python3 startree.py <number of desired rows>*

**Problem 3:**
*guessnumber.py*

My solution in *guessnumber.py* contains one function, **main**, which plays a guessing game with the user. The user is required to enter an integer between 0 & 9, and they get three guesses. At the beginning, the function creates a variable **rnum** that contains a random integer between 0 and 9 (obtained using the *random* module). The function then initiates an iteration that asks the user for their guess three times. The user must guess three times, regardless of whether they guess correctly or not. If their number is too high compared to **rnum**, the function will print "Too big;" if their number is too low, it will print "Too small;" if their number is right on, it will print "Perfect!"

The function also utilizes a boolean variable **hasWon** that is originally false and only made true if the user guesses correctly. At the end of the loop, if **hasWon** is true the function will print "You win!" Otherwise, the function will print "You lose!"