4/3/20

Dominic Zecchino

Part 1: Class Practice

This section of the homework assignment was structured around creating a class with callable functions that allow the user to determine the area and perimeter of a rectangle. Values were initialized using __init__ as a function with controllable parameters including a length attribute and a width attribute. The only reason that this initialization works, however, is due to an extra parameter, an object to be more specific, called self. The self object allows the user to manipulate what they want specifically without creating a massive amount of variables to get the same task completed. This is done by using, for example, self.length_attribute and self.width_attribute. This self object was used to control lengths and widths during the first section of the assignment. Additionally, the code represents the ability to call a specific function within a class. One example seen in the written code for part one is in the main function, more specifically, myRec = Rectangular(10,20) followed by print(myRec.area()). The snippet of code just mentioned sets a label (myRec) equal to the class at specific parameter values (Rectangular(10,20)).

The second task within the first part of the assignment involved numpy. It was asked that two numpy arrays of size 10 be defined with length and width. These arrays served the sole purpose of testing the accuracy of the Rectangular class. Because both arrays are of size 10, the application of those arrays into the class yield 10 output values for both area and perimeter.

Part 2: Display Time

Part two was intuitively rather simple. The idea was to take two times in hours, minutes, and seconds, and output the total time. The issues arose when realizing that once the second or

minute counter reached a value greater than or equal to 60, the value had to be scrapped and the value in the position in front of it had to be increased by an extra value. This means that, for example, 65 seconds would have to be rewritten as 1 minute 5 seconds. A method named addTime was created to accomplish this very task, once again using the self object. Inside of the addTime method another function was called, called makeProper. The method makeProper took the values that were greater than or equal to 60, and rewrote the time. In my code specifically, I created two different ways to add times together correctly, making sure to rewrite them with makeProper. A method called displayTime was then created to print that new time. For the last portion of this section another method was created called DisplayMinute.The purpose of this method was to take a given time, and output its equivalent value completely in seconds.

Part 3:

Part 3 was definitely the most complex part out of the three, and undoubtedly the most extensive. Initialized values were first created including seed, multiplier, increment, modulus, and once again the object: self. A linear congruential algorithm was used in this portion to create a random number generator. Additionally, methods were created to get the seed, set the seed, find the next random number, and output a sequence of randomly generated numbers. Really, though, this is just pseudo-randomness. True randomness is impossible to replicate in code so this outputs a seemingly random sequence of numbers that can all be linked to a single formula. An inherited class, called SCG, was then created that allowed it to utilize all of the code from the LGC class without rewriting a large portion of the code. A multitude of values were tested for this to make sure it was working properly.

Following that section, the assignment led us toward bringing the files together. A Point.py file was created with a new class that had the purpose of determining distances from the center. That calculation would eventually be utilized in comparing distribution ratios. The last file created was MCTest.py. This file was utilized to bring the different components of the homework together. The Generator.py file and the Point.py file were first imported. Because the values the generator was generating was in the millions, a scaling function was created to bring every value between -1 and 1. These new values were put into a new list. This process was done for both the x and y lists. Those values were then turned into coordinates using both the RectangularCoordinate class and a makePoint method. The final spread of the values were then able to be tested against the theoretical distribution. The final outcome of my written code had a distribution ratio difference of less than 9.7*(10^-5), which is unbelievably close to the theoretical.

I pledge my honor that I have abided by the Stevens Honor System.
Dominic Zecchino