CSCI 1133, Spring 2018 - Final Project

Clarifications, 5/1:

1. To submit your final project, please push your file to the master branch of your usual GitHub repository under the name yourx500_FinalProject.py, where "yourx500" is replaced with your x500 id. For example, if your x500 is abcde123, you would push your final project file abcde123_FinalProject.py to the master branch of the CSCI 1133 repository repo-abcde123. You do not need to create a folder to submit this project, but please follow the capitalization and punctuation conventions given in the example above so that our grading scripts can find your submission.
2. In your plots, please use the x-coordinates -300, -190, -80, 30, 140, and 250 to represent the six years of data, with -300 being associated with the year 2010 and 250 being associated with the year 2015.
3. Please write the name of the County and/or State directly on the turtle screen in your `display()` methods, rather than setting the window title to the name.
4. As a clarification, the indication that your main program “should not be in a main function” means that your main method code should be at the bottom of your program, outside of any function.

Clarifications, 5/2:

5. When you write the class methods which have been named below, you must use the provided names and parameters. It is unnecessary to add more parameters to these methods.
6. Do not create Turtle objects for this assignment. Your `display()` methods can use the turtle module’s methods directly, as in `turtle.pencolor("red")`.

Optional Bonus Problem, Added 5/3:

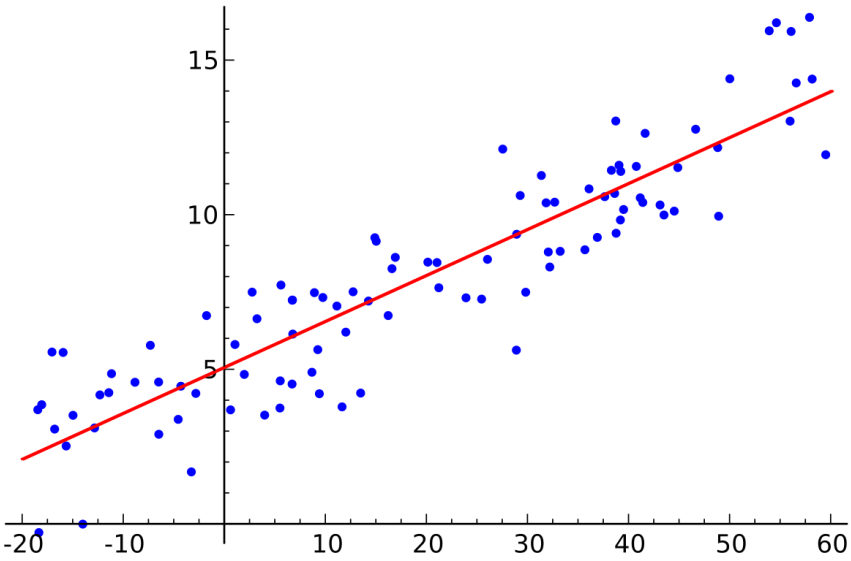
For a small amount of bonus points on this assignment, you may choose to implement axes for your plot. To earn the bonus credit, you must meet the following constraints:

- Draw a horizontal line at y = -300 from x = - 310 to x = 250 to represent the x-axis.
- Draw a vertical line at x = -300 from y = -310 to y = 210 to represent the y-axis
- Draw short vertical tick marks on the x-axis, with one tick mark at the location of each year (2010, 2011, 2012, 2013, 2014, and 2015).
- Draw short horizontal tick marks on the y-axis, with one tick mark at ¼ of the maximum population, one tick mark at ½ of the maximum population, one tick at ¾ of the maximum population, and one tick at the maximum population.
- Label the tick marks on the x-axis with their corresponding years (not pixel-based x-coordinates!).

- Label the tick marks on the y-axis with their corresponding population values (not pixel-based y-coordinates!).

PLEASE NOTE: This is a graded assessment of individual programming understanding and ability, and is not a collaborative assignment; you must design, implement and test the solution(s) completely on your own without outside assistance from anyone except teaching staff. You may not consult or discuss the solution with anyone. In addition, you may not include solutions or portions of solutions obtained from any source other than those provided in class.

Least-Squares Regression



Note: This plot is a general example of least-squares regression-- your plot for this assignment will be significantly simpler.

One of the most dramatic advantages of computer programming is the ability to analyze data quickly and accurately. In this project, you will create classes in Python that compute a least-squares regression model for a collection of population data. You will also write and use functions that do the necessary calculations for least-squares regression. Your models and data will be displayed in turtle graphics.

This project also requires you to implement inheritance, but the exact structure is up to you to design.

Data and Classes

The data you will be analyzing are the populations from 2010 to 2015 of counties and states in the U.S. You will have to retrieve this data from a .csv file which is provided to you. You will then create objects out of these data, from classes that will represent populations of either a county or a state. Another class, `Analysis`, will have only a single instance, and this will contain each instance of both `State` and `County`. As you think about how to write these classes (and any others you find that your design needs), think about how you can implement class inheritance. Which classes have elements in common? For those elements that are the same, they should be in a base/parent class. If you are copying and pasting code from one class to another, think carefully -- can that be written once in a base class instead? The answer is not always "yes," but it often is.

County

Each county in the file censusdata.csv will be represented by an instance of the `County` class. This class will have a string attribute representing the county's name and another attribute which is a list of its populations (in the correct order; that is, the 2010 population is at index 0, 2011 at index 1, and so on). In the constructor, you will also compute the least-squares linear regression model for the data, which gives the best-fit line. Most of the mathematical work is done for you, but if you want an in-depth explanation of the following algorithm, click here.

First, recall that lines follow the form: $y = mx + b$, where m is the slope and b is the y-intercept. In our situation, with x representing the number of years since 2010, we have six data points from $x = 0$ to $x = 5$, so m and b are given by

$$\begin{bmatrix} m \\ b \end{bmatrix} = \begin{bmatrix} -1/7 & -3/35 & -1/35 & 1/35 & 3/35 & 1/7 \\ 11/21 & 8/21 & 5/21 & 2/21 & -1/21 & -4/21 \end{bmatrix} \cdot \vec{p}$$

where $\vec{p}$ is your list of population values in vector form. If you aren't familiar with matrix multiplication, this means simply that m is equal to the sum of $-1/7$ multiplied by the population in 2010, $-3/35$ multiplied by the population in 2011, and so on, while b is equal to the sum of $11/21$ multiplied by the 2010 population, $8/21$ multiplied by the 2011 population, and so on. Once you have computed m and b, assign them as attributes of the county.

You may write whatever getters you need, but setters won't be necessary. You *will* need to overload the `__lt__(self, rhand)` operator (less than) so that a county object will be considered "less than" another if its growth (the slope of its least-squares line, or

m) is less than that of the other county. This overloaded operator will allow your counties to be compared and sorted by their growth rate.

Your `County` class will also have a `display(self,y_max=200)` method that will use turtle graphics to draw each data point in red at even intervals from x = `-300` to x = `250`, where x corresponds to the year (2010 is considered year 0). In your plots, please use the x-coordinates -300, -190, -80, 30, 140, and 250 to represent the six years of data, with -300 being associated with the year 2010 and 250 being associated with the year 2015. The y coordinate will range from `-300` to `y_max`, where y = `-300` corresponds to a population of 0 and `y_max` corresponds to the county's maximum population over the six years. Your method will then draw the least-squares linear regression model over these data, and use `turtle.write()` at the rightmost point of this line to display the line's equation, $y = mx + b$, with four decimal places in m and b. Finally, write the county's name at the top of the window (note that you should write this name within the turtle screen, rather than setting the name as the window header).

State

Each state in the .csv file will be represented by an instance of the `State` class. In addition to the attributes the `County` class has (name, population data, growth rate, and y-intercept of regression line), the `State` class will also have an attribute for a list of all the counties it contains. The `State` class will overload the `__lt__(self, rhand)` operator just like the `County` class. Finally, the `State` class will have a similar, `display(self)` method, but it will draw in blue instead of red, and the maximum y coordinate will always be `200` (so there won't be a `y_max` parameter).

Note that since methods below will require `State`'s `display()` method and `County`'s `display()` method to execute at the same time, you should make sure that they do not write their names in the same location.

Analysis

Create an `Analysis` class that contains each `State` instance in a list attribute. This class must also have the following methods:

`displayState(self, name)`: Calls the `display()` method of the state whose name attribute matches `name`.

`displayStateGreatestCounty(self, name)`: Calls the `display()` method of the state with that name and the `display()` method of that state's county whose

growth is the highest. Be careful to calculate `y_max` such that the county is displayed on the same scale as the state is.

`displayStateLeastCounty(self, name)`: As above, but for the county whose growth is the least in that state.

`clear(self)`: Clears the turtle screen.

`greatestState(self)`: Prints the name of the state with the highest growth rate.

`leastState(self)`: Prints the name of the state with the least growth rate.

censusdata.csv

The .csv (comma-separated values) file provided to you contains on each line either the name of a state or the name of a county (always contains “County” in the name), followed by the populations of that state or county in the years 2010 through 2015. Every line that contains a state is followed by all the counties contained in that state, followed by the next state.

File Structure

Your file will contain the class definitions, any import statements or helper functions you need, and the main program. In the main program (which should be at the bottom of your program, outside of any function) you must open censusdata.csv, instantiate the correct objects to contain the data, and instantiate an `Analysis` object with the variable name analysis (to enable the testers to use its methods at will).