CPSC 231 Assignment 2

Due Date: Friday, October 18th, 2019 11:59 PM

Weight: 8%

Collaboration

Discussing the assignment with others is a reasonable thing to do, and an excellent way to learn. However, the work you hand-in must ultimately be your work. This is essential for you to benefit from the learning experience, and for the instructors and TAs to grade you fairly. Handing in work that is not your original work, but is represented as such, is plagiarism and academic misconduct. Penalties for academic misconduct are outlined in the university calendar.

Here are some tips to avoid plagiarism in your programming assignments.

1. Cite all sources of code that you hand-in that are not your original work. You can put the citation into comments in your program. For example, if you find and use code found on a web site, include a comment that says, for example:
   ```
   # the following code is from
   https://www.quackit.com/python/tutorial/python_hello_world.cfm.
   ```
   Use the complete URL so that the marker can check the source.
2. Citing sources avoids accusations of plagiarism and penalties for academic misconduct. However, you may still get a low grade if you submit code that is not primarily developed by yourself.
3. Discuss and share ideas with other programmers as much as you like, but make sure that when you write your code that it is your own. A good rule of thumb is to wait 20 minutes after talking with somebody before writing your code. If you exchange code with another student, write code while discussing it with a fellow student, or copy code from another person's console, then this code is not yours.
4. We look for plagiarism in all code submissions, possibly using automated software designed for the task. For example, see Measures of Software Similarity (MOSS - https://theory.stanford.edu/~aiken/moss/).

Remember, if you are having trouble with an assignment, it is always better to go to your TA and/or instructor to get help, than it is to plagiarize.

Late Penalty:

Late assignments will not be accepted.

Submission Instructions:

Your program must be submitted electronically. **Use the Assignment 2 dropbox in D2L** for the electronic submission. You can submit multiple times over the top of a previous submission. Don't wait until the last minute to attempt to submit. You are responsible if you attempt this and time runs out.

Charting Expressions (Graphing Calculator)

You will be creating a small graphical Python 3 program. This program will use the **turtle** library to draw based on information taken from the user. You should already have all the experience you need with this library from Assignment 1.

The assignment requires an additional understanding of converting between Cartesian coordinate systems, using loops, and completing functions as outlined by someone else. You will be provided with a starter code that already implements the basic interaction with the user (something you learned in the previous assignment). This code consists of a completed **main** function, which sets up the window using a **setup** function. Neither of these functions should be changed.

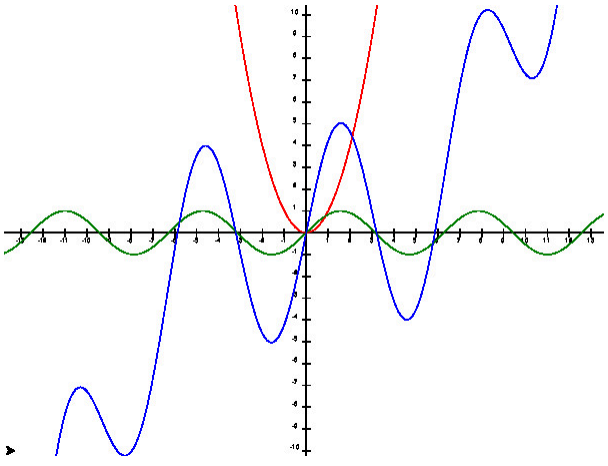
The **main** function makes use of several incomplete functions. Your assignment is to **complete these functions** so that the **main** function can use them to draw what the user requests. **All your code should be written within the incomplete functions. You are allowed to create constants and new functions as long as they are not redundant to existing ones.** There should be no global variables in your code (except if you are doing the bonus). There is a bonus that involves identify local minima/maxima and labelling them described at the end of the assignment.

Your program will be drawing in an 800-pixel by 600-pixel window with (0,0) as the bottom left corner and (800,600) as the top right corner. The starter code has a setup() function, which setups up the window and returns the turtle drawing object **pointer** so you can use it. This **pointer** is passed into the existing starter code functions for you to use. The included starter code already prompts the user for the pixel coordinates for where the chart origin should be placed in the window. The code then prompts the user for a ratio of how many pixels are 1 step of the chart. For example, an origin of $(xo, yo) = (400,300)$ should be the centre of the screen. If the ratio of pixels per single step is 100 pixels per step. Then a position one step up from the origin point to $(x, y) = (0,1)$ in the charting coordinate system is

$$(xo + ratio * x, yo + ratio * y) = (400 + 100 * 0,300 + 100 * 1)$$

$$= (400,300 + 100) = (400,400).$$

Below is an image of 3 curves drawn for a chart with an origin at 400,300 and a ratio of 30.



The existing starter code uses the included (but incomplete) functions to draw the x and y axes in black, and then loops to get expressions from the user. The colour used to draw an expression is determined

based on the number of previously drawn expressions. So, you must track this number using a counter variable.

Your task is to complete the included functions, so they function in the way that the comments and assignment description require. You will likely find the visual examples of operation included in the assignment description helpful. I recommend approaching the functions in stages:

Part 1: screenCoor:

First, complete the coordinate conversion function **screenCoor**. This function uses the parameters that the **main** function obtains from the user, along with an (x,y) coordinate to perform the conversion. It returns this coordinate as a pixel screen coordinate following the math described above in the example. Use this function in your code any time you want to plot a chart location. You will lose marks if you duplicate the code functionality of this function somewhere else instead of calling it.

Part 2: getColor:

Second, complete the colour-determination function **getColor**. This function takes the counter tracked in the expression input loop and returns one of three different colours: red, green, or blue. You should find the remainder operator (also called modulus) helpful here because it allows you to change an infinitely increasing integer sequence into a range of integers.

Part 3: drawXAxis, drawYAxis

Third, complete the two functions that draw the x and y axes. These two functions are structurally similar, so complete one of them first then you can easily complete the other function.

If you choose the x-axis, initially you can ignore updating the return values for xmin and xmax. You should be able to add these in later once you have the drawing working. When first drawing your axes, you can ignore the tick marks and labels. These can also be added in once you have the lines drawn successfully.

My advice is to use two loops. Start at the chart origin (0,0) and use a loop to move 1 chart step away. Convert this point from a chart location into a screen location. Draw a line from the previous point to the current point. Once you draw to a location outside the screen, then stop looping. Go back to the origin and draw in the other direction.

Once you can draw this line step by step, you should notice that you are stopping at each label/tick location. Make a call to the associated label/tick draw function included in the assignment. Then fill in the code of this function to draw the tick/labels.

Once you complete the drawing part of the drawXAxis function, you will want to ensure the xmin and xmax hold the correct values when the function completes. These two values start at 0,0 for the chart origin location. You should update these values any time you draw at a chart x-location when drawing the axis. If the x-location is less than xmin update it, if it is greater than xmax then update it. You can use the min(),max() functions to do this, or simply use conditional statements if you desire.

Part 4: drawExpr

Finally, you will draw the actual expression that the user inputted. Python can evaluate a string using its **eval** function. For example, the following 3-line program is a simple calculator:

```
expression = input("Enter an arithmetic expression: ")
result = eval(expression)
print("The result of that expression is", result)
```

Python is also able to evaluate expressions which include a single variable **x.** The implementation of this function can be surprising at first. When the **eval** function is called, it uses whatever value is stored in the variable called **x** to evaluate the given expression.

As a result, your program will need to include a variable named **x**, which represents the **x** coordinate in the Cartesian coordinate system. The value of **x** will change in a loop. Call **eval** inside of the loop so that you can compute the value of **y** for many different **x** values. For example, the following program evaluates an expression that includes the variable **x** for each integer value of **x** from 0 up to and including 5:

```
expr = input("Enter an arithmetic expression: ")
for x in range(0, 6):
      y  = eval(expr)
      print("When x is", x, "the value of the expression is", y)
```

You can use a similar technique to compute the **y** position of the curve for many different values of **x**. Those **x** and **y** values are what you need to draw the curve.

You cannot complete this function correctly unless the previous functions are operational. Use the xmin and xmax that were determined in your previous function to loop through x-coordinates of the curve. You will need to pick a delta that makes the curve smooth. A delta of 0.1 or 0.2 should provide sufficient smoothness. Since you will be stepping through floating-point numbers, you will need to use a while loop instead of a for loop.

Additional Specifications:

Ensure that your program meets all the following requirements:

- You should have the class, your name, your tutorial, your student id, the date, and description at the top of your code file in comments. Marks are given for these.
- The code file should be **CPSC231A2<LastName>.py** (ex. Mine would be CPSC231A2Zaamout.py)
- Import the necessary libraries
- Use constants appropriately. Your TA may note one or two magic numbers as a comment, but more will result in lost marks.
- Use descriptive variable names. Using the same variable names that appear in mathematical equations is OK.
- Draw your axes black, alternate your colours for curves through red, green, and blue.
- Use in-line comments to indicate blocks of code and describe decisions or complex expressions.

- Do not put any code outside the functions given. You can add functions, but none should duplicate functionality described in a current function. For example, don't create functions to draw the labels/tick marks as two already exist.
- Break and continue commands are generally considered bad form. As a result, you are NOT allowed to use them in this assignment. In general, their use can be avoided by using a combination of if statements and writing better conditions on your while loops.
- You do not need to perform any error checking in your program. You may assume that the user always enters a valid expression or a blank line indicating that they want to quit. A valid expression contains x, operators, numeric constants and standard mathematical functions.

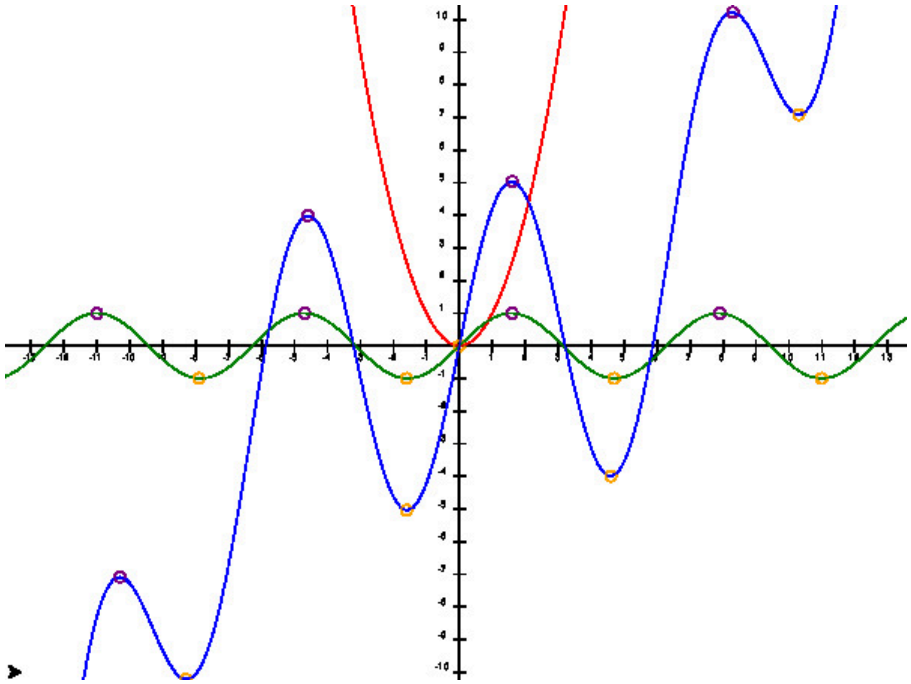
Bonus:

Looking for an A+? Improve the program so that it marks every local minimum in orange and every local maximum in purple. Use a small circle to do this. Create a function that draws a circle for you around a given point. Make sure that your program does not mark inflection points, such as (0, 0) when graphing y = x ** 3 (and other functions that include one or more inflection points). Also, print the largest local minimum (global maximum) and lowest local minimum (global minimum) to the shell window for each expression entered (ignore infinite values). At the same time track the largest global maximum and global minimum across all expression entered. To track these values across expressions, please use **global** variables. These two variables should be the only global variables in your program. Print the updated values tracked for each of these after each expression is entered.

Examples:

The following image shows the result of plotting three curves (w/ bonus). User input is shown in bold.

Enter pixel coordinates of origin: **400,300**
Enter ratio of pixels per step: **30**
Enter an arithmetic expression: **x**2**
Enter an arithmetic expression: **sin(x)**
Enter an arithmetic expression: **0.01*x**3+5*sin(x)**
Enter an arithmetic expression: **<Enter>**



Grading:

Part one of the assignment will be graded out of 12, with the grade based on the program's level of functionality and conformance to the specifications. To get more than an F, your code must not have syntax errors. Code with runtime errors that occur **during proper usage** of the program, every time it runs, will not get better than a C grade. Runtime errors are your program crashing.

Your TA will begin by grading your code with a general functionality grade starting point and subtract marks when smaller specifications are unfilled.

The total mark achieved for the assignment will be translated into a letter grade using following table:

Mark	Letter Grade	Starting Point Guidelines (not a final grading scheme!)
13	A+	Appears to fulfill assignment and bonus spec
12	A	Appears to fulfill assignment spec
11	A-	
10	B+	
9	B	Code draws axes and is partially able to draw curves
8	B-	
7	C+	
6	C	Code draws axes but can't draw curves
5	C-	
4	D+	
3	D	Code has only getColor/screenCoor completed
0-2	F	Syntax errors or barely started code

Example Output for Bonus:
Enter pixel coordinates of origin: **400,300**
Enter ratio of pixels per step: **30**

Enter an arithmetic expression: **x**2**
No expression global maximum
Expression global minimum (-0.00000, 0.00000)
No expression global maximum
Global minimum for all expressions (-0.00000, 0.00000)

Enter an arithmetic expression: **sin(x)**
Expression global maximum (-11.00000, 0.99999)
Expression global minimum (11.00000, -0.99999)
Global minimum for all expressions (-11.00000, 0.99999)
Global minimum for all expressions (11.00000, -0.99999)

Enter an arithmetic expression: **0.01*x**3+5*sin(x)**
Expression global maximum (8.30000, 10.22873)
Expression global minimum (-8.30000, -10.22873)
Global minimum for all expressions (8.30000, 10.22873)
Global minimum for all expressions (-8.30000, -10.22873)

Enter an arithmetic expression: **<Enter>**