CPSC 217 Assignment 2

Due: Monday October 26, 2015 at 12:00 noon
Weight: 7%
Sample Solution Length: 98 lines, including comments and the code to mark the minima and maxima

Individual Work:

All assignments in this course are to be completed individually. Students are advised to read the guidelines for avoiding plagiarism located on the course website. Students are also advised that electronic tools may be used to detect plagiarism.

Late Penalty:

Late assignments will not be accepted.

Submission Instructions:

Your program must be submitted both on paper and electronically. Paper submissions, consisting of a printed copy of your .py file, should be deposited in the appropriate assignment drop box on the second floor of the math sciences building. Your electronic submission should be submitted to the Assignment 2 drop box in D2L.

Description

In this assignment you will create a program that graphs mathematical functions entered by the user. Your program will prompt the user for the right side of an expression and then graph the function. For example, if the user enters `sin(x)` as input then your program will graph `y = sin(x)`. Your program should include a loop that allows the user to graph several curves in a single run. The user will enter a blank line to indicate that no more curves will be drawn. Call the `close()` function from the SimpleGraphics library after a blank line is entered so that the graphics window closes automatically and your program completes after the blank line is entered.

Completing this assignment will require the use of loops and if statements. You do not need to write your own Python functions to complete this assignment, though you may if you would like to. If you choose to write your own functions you must use them properly, including thorough documentation.

The following sections provide additional requirements and suggestions for implementing your program.

Drawing the Axes

Your program should begin by drawing the axes. When you draw the axes, they must be oriented so that the centre of the Cartesian coordinate system (0, 0) is at the center of the screen (400, 300). Each unit in the Cartesian coordinate system must be represented by 30 pixels. As such, the visible portion of the x-axis will be from approximately -13 to +13, and the visible portion of the y-axis will be from approximately -10 to +10. Your set of axes should include a tick mark and a label for each unit (you can optionally omit the 0s at the origin for a cleaner look). While it is possible to create the axes using a

(long) sequence of function calls alone, you must use loops appropriately to keep the total number of lines of code to a more reasonable amount.

Drawing the Curve

The user will enter the right side of an equation as a string that you will read with Python's input function. Python can evaluate a string as code 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)
```

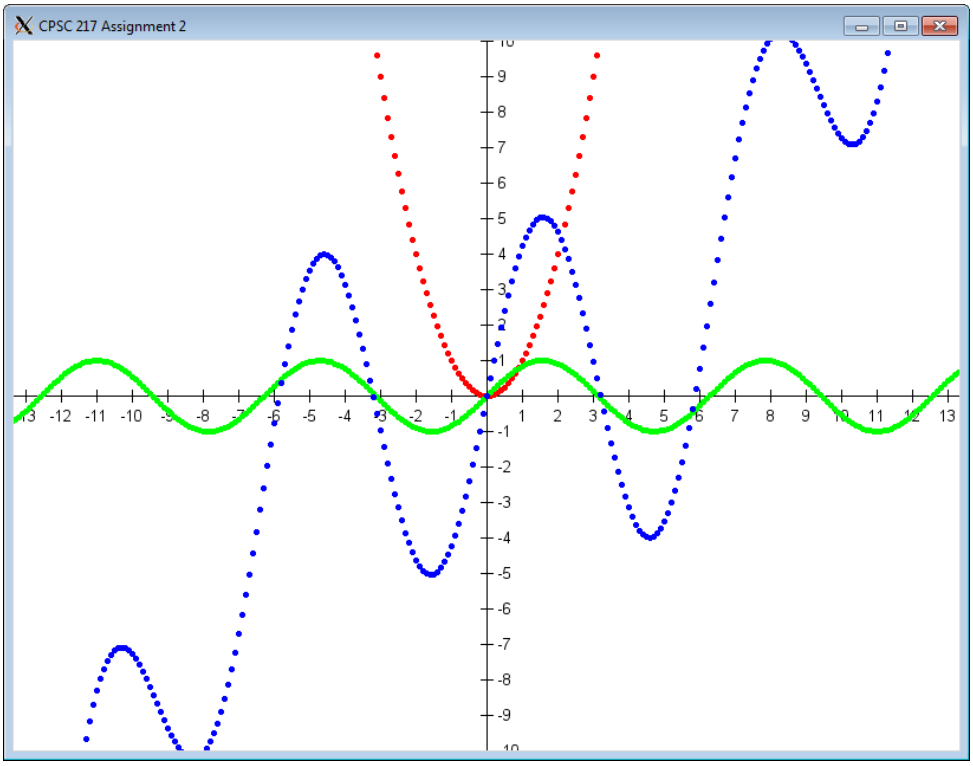
The string that the user enters for your program will typically include an x in it in addition to mathematical operators and numeric constants. As a result, your program will need to include a variable named x which represents the x coordinate in the Cartesian coordinate system. The value of x will change 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:

```
expression = input("Enter an arithmetic expression: ")

for x in range(0, 6):
  result = eval(expression)
  print("When x is", x, "the value of the expression is", result)
```

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 in order to draw the curve.

Your ultimate goal should be to draw the curve using a collection of short line segments. However, as an easier first approximation, consider drawing a series of ellipses or rectangles that are close together (say only 0.1 or 0.2 in the Cartesian coordinate system). This can be accomplished by constructing a loop that iterates over the visible portion of the x-axis (approximately -13 to +13), computing the y position of the curve at each x value using python's `eval` function, and then displaying the rectangle or ellipse. For example, the output of your first approximation might look like this:



While drawing rectangles or ellipses gives a general sense of the shape of the curve, it doesn't result in a connected curve in many cases. As a result, you must extend your program so that the curve is drawn as a collection of short line segments. Drawing a line segment requires two end points. You will want to compute the value of y at location x, and a second y-value at position x + delta (where delta is a reasonably small value like 0.1 or 0.2). This will give you two end points for a short line segment. Repeatedly drawing such segments you will result in a good approximation of the curve, as shown later in this document.

Additional Specifications:

Your program should follow good programming practices. This includes using meaningful variable names, avoiding the use of magic numbers and including adequate comments. You should also ensure that you include a comment at the top of the file with your name, student number and a brief description of the purpose of your program.

You do not need to perform any error checking in your program. In particular, you may assume that the user always enters a valid expression containing x, operators, numeric constants and standard mathematical functions (or a blank line indicating that they want to quit). Your program should import the entire math library using the following statement so that the user can include trigonometric functions in their expression:

```
from math import *
```

Your program should draw the first curve in red, the second curve in green, and the third curve in blue. If more than three curves are drawn then your program should go back to red and repeat the colors in

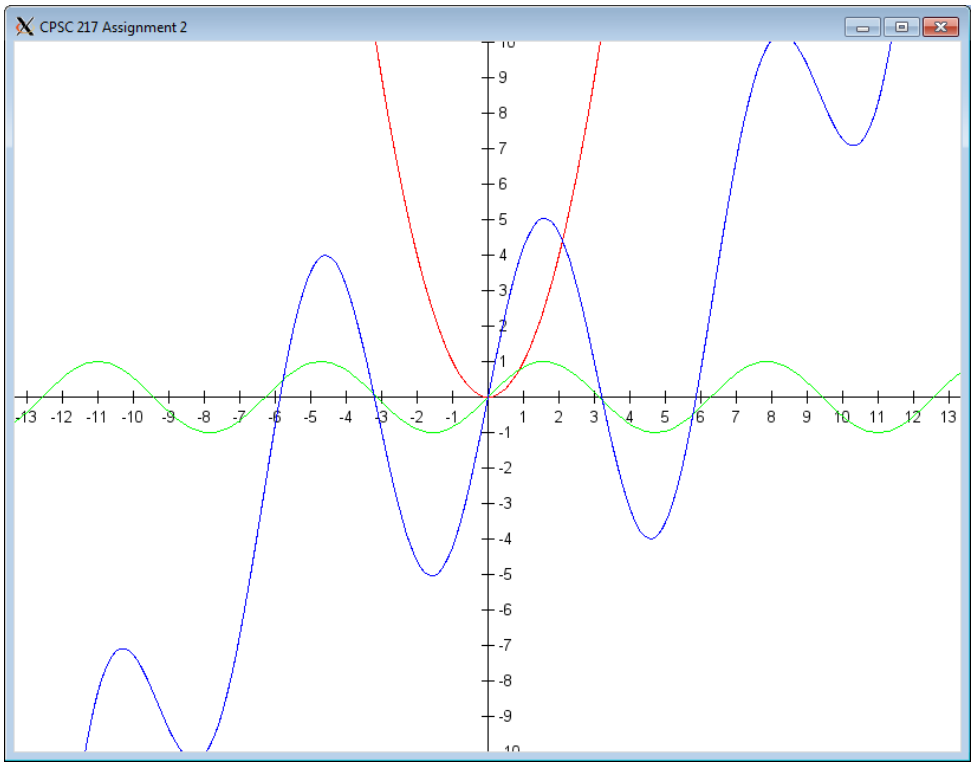
sequence. Hint: You may find the remainder operator helpful when fulfilling this requirement. Keep a count of how many curves you have drawn and then compute the remainder when dividing by 3. If it's 0 then set the color to red. If it's 1 then set the color to green. Otherwise set the color to blue.

Your program should close the window after the user enters a blank line. Use the close() function in the `SimpleGraphics` library.

Sample Input and Output:

The following image shows the result of plotting three curves. User input is shown in bold face.

```
Enter the expression (blank line to quit):
y = x ** 2
Enter the expression (blank line to quit):
y = sin(x)
Enter the expression (blank line to quit):
y = 0.01 * x ** 3 + 5 * sin(x)
Enter the expression (blank line to quit):
y = <Enter>
```



Grading:

This assignment will be graded on a combination of functionality and style. A base grade will be determined from the general level of functionality of the program (Does it plot a curve successfully? Does it plot multiple curves successfully? Does it draw the axes correctly? Is the curve connected properly?). The base grade will be recorded as a mark out of 12.

Style will be graded on a subtractive scale from 0 to -3. For example, an assignment which receives a base grade of 12 (A) because it generates perfect output, but has several stylistic problems such as magic numbers and missing comments resulting in a -2 adjustment will receive an overall grade of 10 (B+).

Total Score (out of 12)	Letter Grade
12	A
11	A-
10	B+
9	B
8	B-
7	C+
6	C
5	C-
4	D+
3	D
0-2	F

Looking for an A+? Improve the program so that it marks every local minimum in orange and every local maximum in purple. For example: