EN.540.635 "Software Carpentry"

Weekly Challenge 2 - Quadratic Equation Solver

For this assignment, you will need to be able to run Python. In lab, we will go over connecting to a remote machine and running code there (say, on Rockfish). We will also cover installing Anaconda and Python in class, so that we can clear up any problems that you may have run across so that you can set up your programming environment exactly as you want it. Before getting to the assignment, we will first review a few important topics that are relevant to the assignment.

Variables: A variable is a way to store values in codes. Essentially, they point the code/program to where the data is stored. Data itself can take on many forms, the simplest of which are the following:

- **int**: An integer (0, 1, 2, -3, -5, ...). In math, this is the numbers in the set $\mathbb{Z}$.
- **float**: A real number (1.2, 3.423, -12.32, ...). In math, this is the numbers in the set $\mathbb{R}$.
- **str**: Strings is a computer term for text. Strings can hold any characters!

In class we learned how to print text to screen using the **print** command:

```
>>> print("Hello World")
Hello World
```

As well as how to do basic arithmetic:

```
>>> a = 2
>>> b = 4
>>> a + b
6
>>> a * b
8
>>> a ** b
16
```

We also learned that variables of different data types don't play nicely together (in Python 2.7):

```
>>> 1/3
0
>>> 1/3.0
0.3333333333333333
>>> 10 ** (1/3)
1
>>> 10.0 ** (1/3)
1.0
>>> 10 ** (1/3.0)
2.154434690031884
```

Finally, we learned that we can **cast** between datatypes:

```
>>> a = float(1)
>>> b = 1
>>> print(a)
1.0
>>> print(b)
1
```

In this assignment, you will learn about making function(s). A function is just like what you learned in math. For instance, let's look at the function of $f(x) = a * x^2 + b$. In python, this would look like the following:

```
>>> def f(x):
...     a = 3.0
...     b = 2.1
...     return a * x ** 2 + b
...
>>> f(3)
29.1
```

Note, I've assigned the variables a and b within the function. A nicer way to go about this though would be to give the user the option to specify a and b, but leave a default if they don't want to change it:

```
>>> def f(x, a=3.0, b=2.1):
...     return a * x ** 2 + b
...
>>> f(3)
29.1
>>> f(3, a=2, b=1)
19
>>> f(3)
29.1
```

Notice how we've assigned a default value to a and b in the function call, but allow the users to override this as they see fit. In this instance, x is an *argument* to the function, while a and b are known as *keyword arguments*.

Now, there is something called conditionals in coding that gives us a lot more power. For instance, say we want to determine if a number is positive, negative, or zero. We can do the following:

```
>>> a = -2.3
>>> if a < 0:
...     print("a is negative")
... elif a > 0:
...     print("a is positive")
... else:
...     print("a must be equal to 0")
a is negative
```

Note, we can string together as many *elif* as we want (but keeping one's code concise is also important).

WEEKLY CHALLENGE 2:
Using what you've learned, write a Python program that will calculate the roots of a quadratic equation.

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

We should be able to run your program using a function call to "quadratic_solver" that takes 3 numbers (a, b, and c, in that order) and returns the roots of the quadratic equation. You may want to start off with assuming c will be 0 (Note, we will not tell you how to return multiple values from a function - use Google!). Once you have figured out that part, you can now focus on making a more robust quadratic solver. We started out with the simplification that $c = 0$ because, if it isn't, there is a chance that $b^2 - 4ac < 0$, which would give us an imaginary root! Although Python allows us to have imaginary numbers with the letter j, when taking the square root of a negative number it chooses to throw an error instead. Using conditional statements, and splitting the quadratic function apart into separate values, you can check if it would become an imaginary number and handle it accordingly. Note - there are many ways to go about accomplishing this.

On Canvas, you are to turn in a single .py file with your Python code. Do not write code in the terminal, copy, paste, and turn that in (we will grade harshly if you do so, as the code will 100% not run).