**Simulated Spring Mass Damper**

1. Write a function, called circle\_area that takes the radius and returns the area of a circle.
2. In a new file, write code that prints out the radii and areas of a number of circles. Your output should look (exactly) like this:

0.0 -> 0.0  
0.5 -> 0.785398163397  
1.0 -> 3.14159265359  
1.5 -> 7.06858347058  
2.0 -> 12.5663706144  
2.5 -> 19.6349540849  
3.0 -> 28.2743338823  
3.5 -> 38.4845100065  
4.0 -> 50.2654824574  
4.5 -> 63.6172512352  
5.0 -> 78.5398163397  
5.5 -> 95.0331777711  
6.0 -> 113.097335529  
6.5 -> 132.732289614  
7.0 -> 153.938040026  
7.5 -> 176.714586764  
8.0 -> 201.06192983  
8.5 -> 226.980069222  
9.0 -> 254.469004941  
9.5 -> 283.528736986  
10.0 -> 314.159265359

**Do not** copy your circle area code into this new file; bring it in with an import statement. Use a for loop.

1. OK. Enough of circles, let's do something more interesting. A spring-mass-damper system is characterized by the following equation:

http://web.engr.oregonstate.edu/~smartw/me499/labs/lab1/equation.png

where *m* is the mass in kg, *k* is the spring constant in newtons per meter, and *c* is the damping coefficient in kg per second. Write a function to calculate the current acceleration, given *m*, *k*, *c*, *x*, *x\_dot*.

Next, modify your function to take additional time parameter, *t*. Your current function calculates the acceleration after a second of movement. You can calculate smaller time intervals by multiplying the whole thing by the time parameter before returning it.

1. Now, we're going to try a simulation. Assign variables for values of the constants *m*, *k*, *c* in your code. Assign a value of 0.01 for *t*. Set the initial *x* and *x\_dot* (don't make these both zero). The main body of the simulation is going to be a for loop that
   1. calculates the current acceleration, using your function
   2. adds the acceleration to the current velocity, *x\_dot*
   3. adds the current velocity to the current position, *x*
   4. prints out the iteration number and the x position

Your output should look something like this:

0 1.0  
1 0.99  
2 0.9702  
3 0.940896  
4 0.90247608  
5 0.8554155984  
6 0.800271565632  
7 0.737676257535  
8 0.668330139944  
9 0.59299418213  
10 0.512481642072

(for all constants set to 1.0, *x* = 1.0, *x\_dot* = 0.0, and *t* = 0.01).

1. Since columns of numbers are hard to interpret, let's plot the results. We're going to use an open-source plotting program called gnuplot for this. Open a new terminal, move to the lab1 directory, and type gnuplot.

Modify your code to print out 1,000 time steps of 0.01 seconds. Set your constants and initial conditions to those in the previous part. Assuming that your file is called foo.py, type this at the command line:

./foo.py > bar

This will send the output from your code to a file named bar. Make sure it exists before proceeding.

Now, in the gnuplot program, type:

plot "bar" with lines

This should, if all goes well, display something like this

