COMP3522 Lab 2: Off With a Bang

**Classes and Objects**

Rahul Kukreja & Jeffrey Yim

jyim3@bcit.ca

**Welcome!**

In today’s lab, you will:

1. Implement and use classes in simple object oriented design
2. Make decisions about where to put logic in order to maximize encapsulation and information hiding
3. Demonstrate proficiency using the random module to generate pseudorandom integers
4. Demonstrate proficiency using the datetime module to ensure events take place when required
5. Reacquaint yourself with the joy of programming. No seriously. This is fun stuff!

**Grading**

This lab and all future labs will be marked out of 10

For full marks this week, you must:

1. **(2 point)** Correctly use git/GitHub and the repository
2. **(6 points)** Generate a correct solution to the problem(s) in this lab
3. **(2 point)** Correctly format and comment your code.

**Requirements**

Please complete the following:

1. Clone the project from: [**https://classroom.github.com/a/EqV4CAkM**](https://classroom.github.com/a/EqV4CAkM)
2. Open a browser window to <https://docs.python.org/3.7/library/random.html#random.randint> . You will use the randint function available in the random module to generate random integers. Generating (pseudo-)random numbers in Python is trivially easy. Import the random module, and then access the function by using the module name and dot syntax. It accepts a lower bound (included) and an upper bound (included).
3. Open a browser window to <https://docs.python.org/3.7/library/datetime.html> . You will use the datetime module in this lab to ensure a program tracks movement over time.
4. Implement a class called Asteroid in a file called asteroid.py:
   * An Asteroid stores three attributes. Circumference in metres, position, velocity. Position and velocity are vectors composed of x, y, and z coordinates in 3D space measured in metres per second. Will you store them separately or as a list or as a tuple? Which choice creates the most maintainable code that is easiest to understand?

Optional: See if you can create a Vector class that represents a vector in X, Y and Z dimensions and consists of the following:

* + - Implements an add(self, vector) method which adds another vector to itself.
    - Implements properties that let you access the x, y and z values
    - A method that returns the vector as a tuple.
    - A \_\_str\_\_(self) method that would let you print a vector as a tuple.
  + Create accessors and mutators as required. (or even properties!)
  + Add a move( ) method. The move( ) method modifies the position using the velocity and returns the new position as a tuple of x, y and z coordinates. Recall that velocity is measured in metres per second. We have stored the velocity in three dimension. In order to implement move( ), we just need to add the amounts in the x, y, and z components of the velocity to the current position.
  + Use statics to ensure each asteroid receives a sequential unique ID when it is created (start with 1). You will probably need to modify your initializer and create a helpful class method.
  + Add a \_\_str\_\_(self) method to return the Asteroid's information in a single brief line of output.

1. Implement a class called Controller in driver.py:
   * Controller maintains a list of asteroid objects in the field.
   * The \_\_init\_\_(self, ...) method must create some Asteroids and store them in the list. Each Asteroid should be assigned a random circumference, starting position and starting velocity.
     + Let's make 100 Asteroids with radius in the range [1, 4]. Calculate the circumference of the asteroid based on the radius.
     + Assuming that they are starting in a cube that is 100 metres per side
     + Each asteroid should have a velocity no greater than 5 metres per second in each direction.
     + Where should these methods go? In the controller? Or should some helper methods be added to the Asteroid (or Vector class if you have one)? Consider encapsulation and information hiding.
   * Add a simulate(self, seconds) method. This method accepts a number of seconds. This method must ensure that every asteroid moves once per second for the number of specified seconds. You must use the datetime module to ensure that the looping begins precisely 'on the second'
     + If the current time is 5:05:05.7 there are 0.3 seconds until the next second begins. We must wait 0.3 seconds before we begin counting ‘on the second’
   * Keep looping the simulation precisely on the second until the specified number of seconds have elapsed.
2. Implement a main method where you instantiate a controller of Asteroids.
3. Engage! (That's a Captain Picard reference)
4. Ensure you push your work to github classroom. I'd like to see sensible git commits comments, and commits must take place at logical points in development.

Ensure the output looks similar to the following

