Programming Project 0: Molecule Class
Do you want to build a molecule?

This programming project will teach you the basics of how to properly implement a simple molecule class. In doing so, you will learn how to program in Python and how to employ proper coding techniques.

To begin, we will first need to make a Python molecule class

```
class Molecule:
    """
    A Molecule class that implements the basic properties of a molecule
    """
```

This tells us that we have a new class called Molecule, and the docstring (that thing in between the triple quotes) gives us a little information about it. Next we need to write a function to be run every time we create a new molecule.

```
    def __init__(self, geom_str, units="Angstrom"):
        """
        Make a new molecule
        :param geom_str: a string of the geometry
        :param units: a string of the units used (Bohr or Angstrom)
        """
        self.read(geom_str)
        self.units = units
```

The function `__init__` is run every time a new molecule is created. It takes three arguments, `self` (which is a standin for the name of the new object, such as mol in `mol = Molecule(geom_str)`), `geom_str` (which is a string of the geometry), and units (which default to Angstrom). The function then reads the geometry string and sets the units. Thus we need to write a function to actually read the geometry and save it in a usable form. Here is a stub detailing how it should be implemented.

```
    def read(self, geom_str):
        """
        Reads a geometry string and saves the results
        :param geom_str:
        Generates the following object variables
        :self.atoms: a list of the atoms
        :self.geom: a numpy array with the first index corresponding to the atom
                    and the second corresponding to the x, y, and z coordinates
        """
        geom = []
```

Now that we have saved our geometry, it would be nice to be able to get information about our molecule. There are some built-in functions in Python that you may have run across. `len(mylist)` will return you the length of `mylist`. I would recommend glancing over all the available built-in

functions (https://docs.python.org/3/library/functions.htm). We can override the function `len` to allow us to get the length of our molecule. Here is how to start.

```
def __len__(self):
    """
    :return: the length of the molecule
    """
```

Once you have figured how to return the length of the molecule, we need to make a nice way to print it. To do so, we will need to overide the built-in function `str`. We will want it to look just like the original xyz file. Here is some starter code; you will need to review the string format function to determine how to use it.

```
def __str__(self):
    """
    Format the molecule in a nice way
    """
    line_form = "{:2s} {: >15.10f} {: >15.10f} {: >15.10f}\n"
    out = "{:d}\n{:s}\n".format(len(self), self.units)
```

Finally, we would like to be able to convert from Angstrom to Bohr, and back. This can be achieved by multiplying or dividing, respectively, by 1.889725989 each of the coordinates. Since we are using a numpy array, this is very easy, but I will leave it to you to figure out how to accomplish this in one line for each function.

Now that you are done, look at my code, available in the GitHub repo under jevandezande. How does yours compare? See if you can understand some of the tricks I used to simplify my code.