Classes: Dealing with Complex Numbers

This problem is meant to introduce you to **object-oriented** principles and their implementation in Python. Instead of writing complex code using just functions and conditionals, it often makes sense to encapsulate logic within an object. In the case of complex numbers, this means creating a class which is capable of handling all the operations we'd care about performing on a complex number: addition, subtraction, multiplication, division, and modulus.

In general, a class could be thought of as a data structure. It's like a blueprint or template, containing methods (functions) and attributes (variables owned by the class). When you actually use the class, you *instantiate* it. Here is a great intro to python classes.

A solution to this problem can be accomplished by writing class methods, but a neat feature to be aware of is *overloading* methods. This essentially means redefining what those methods mean in a more specific context.

- `__add__` is Python's addition method
- `__sub__` is subtraction
- `__mul__` is multiplication
- `__truediv__` is division (`__div__` in Python 2)
- `__str__` is what's used to cast the object to string type for printing
- `__repr__` is optional, see here for more details
- Feel free to look up the others!

This enables writing clear, readable code like the following. Remember to customize your object constructor method by defining the `__init__` method.

```
class Complex(object):
    def __init__(self, real, imaginary): # `self` is always required
        self.real = float(real)
        self.imaginary = float(imaginary)

    def __add__(self, no):
        return Complex(self.real + no.real,
                       self.imaginary + no.imaginary)

    def __sub__(self, no):
        return Complex(self.real - no.real,
                       self.imaginary - no.imaginary)
```

```
    def __mul__(self, no):
        real = self.real * no.real - self.imaginary * no.imaginary
        imaginary = self.real * no.imaginary + self.imaginary * no.real
        return Complex(real, imaginary)

    # An additional method to simplify the others
    def conj(self):
        return Complex(self.real, -self.imaginary)

    def __truediv__(self, no):
        if no.imaginary == 0:
            return Complex(self.real / no.real,
                           self.imaginary / no.real)
        else:
            return (self * no.conj()) / (no * no.conj())

    def mod(self):
        squared = (self * self.conj()).real
        return Complex(math.sqrt(squared),0)

    def __str__(self):
        if self.imaginary == 0:
            result = "%.2f+0.00i" % (self.real)
        elif self.real == 0:
            if self.imaginary >= 0:
                result = "0.00+%.2fi" % (self.imaginary)
            else:
                result = "0.00-%.2fi" % (abs(self.imaginary))
        elif self.imaginary > 0:
            result = "%.2f+%.2fi" % (self.real, self.imaginary)
        else:
            result = "%.2f-%.2fi" % (self.real, abs(self.imaginary))
        return result
```