HW 01: Newton's Method for finding roots of polynomials

For many people, their first opportunity to do some "useful" programming was to make a program that computes roots of quadratic polynomials using the quadratic formula. For any of you who wished for such a method that works with other polynomials, I give you **Newton's Method**.

Polynomials are very important to many areas of computer science. One of the most basic operations to do on a polynomial f is to find its roots. That is, find the value x such that $f(x) = 0$. High school algebra classes spend a lot of time solving this problem on quadratic polynomials either by factoring or by the quadratic formula. More generally, there is a classic numerical method for finding roots of a polynomial. It called Newton's Method. It doesn't get the exact answer, rather it approximates the answer. The idea is to start with a guess and then update the guess based on the slope of the graph of the polynomial (i.e., it's derivative). Graphically, the idea is to imagine that the polynomial is linear and find the root of the linear approximation at your current guess. Then repeat with the new guess.

Technically, the algorithm starts with a guess x_0 and then repeatedly computes

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i),}$$

where f is the polynomial, and f' is the derivative of f. You may have seen this in a calculus class. When you code it up, there is no reason to store the previous guesses. That is, if $=$ is assignment, one might as easily write:

$$x = x - \frac{f(x)}{f'(x),}$$

Step 1: Compute Derivatives

To compute the derivative of a polynomial is rather simple. First let's consider the case of monomials. For a constant monomial (the exponent is zero), the derivative is 0. Otherwise, for a monomial ax^b with $b \neq 0$, the derivative is abx^{b-1}. For example,

$$\text{if } f(x) = 3x^7 \text{ then } f'(x) = 21x^6.$$

As another example,

$$\text{if } g(x) = 2x^5 \text{ then } g'(x) = 10x^4.$$

Todo: Write a method called `prime` for the `Monomial` class that returns the derivative of the monomial (as a new Monomial). So, the following should work.

```
m = Monomial(3,7)
assert(m.prime() == Monomial(21,6))
```

To compute the derivative of a polynomial, one merely adds up the derivatives of the (monomial) terms. So,

$$\text{if } p(x) = 3x^7 + 2x^5 + x^2 + 135 \text{ then } p'(x) = 21x^6 + 10x^4 + 2x + 0.$$

Todo: Next, write a method called `prime` in the `Polynomial` class that returns a new `Polynomial` that is the derivative of `self` .

Step 2: Find the roots

Start with a guess. It could be anything, but let's just try 0 for simplicity. Store that guess in a variable. Then, repeatedly update the value of that variable using Newton's Method. Do some experiments to see what might be a good number of iterations for polynomials with integer roots.

Todo: Write a method called `root` in the `Polynomial` class that computes a root of the polynomial. The method should take two parameters, a `guess` and a number of `iterations` . Set the default `guess = 0` and the default `iterations = 50` .

Make some examples for yourself

To make polynomials with fixed roots $r_1, \ldots, r_k$, you can simply multiply together the polynomials $(x - r_1)(x - r_2)\cdots(x - r_k)$. You can also check answers using Wolfram alpha. Note that Wolfram alpha will also give the complex roots.

Watch out for division by zero

A common problem is that you might get a `ZeroDivisionError` if you find a value for which the derivative is zero. Sometimes, the function may also be equal to zero in which case you found the root. Otherwise, you will need to start over from a different guess. This is a tricky case, and you can just add one to the initial guess and start over if it happens.

Just for fun: Calculate π

The Taylor series expansion of $arctan(x)$ is

$$x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \cdots$$

The ith term is $\frac{(-1)^i x^{2i+1}}{2i+1}$.
We can easily make a list of such terms using a list comprehension. Because $\arctan(1) = \pi/4$, we can compute an approximation to π by evaluating such a polynomial at 1 and multiplying by 4. Here it is in code.

```
from polynomial import Polynomial
terms = [((-1)**i/(2*i+1), 2*i+1) for i in range(10)]
arctan = Polynomial(terms)
print(4 * arctan(1))
```

It doesn't look very good. Maybe we need more terms.

```
from polynomial import Polynomial
terms = [((-1)**i/(2*i+1), 2*i+1) for i in range(100)]
arctan = Polynomial(terms)
print(4 * arctan(1))
```

That's a little better, but not quite as good an approximation as $22/7$. More terms!

```
from polynomial import Polynomial
terms = [((-1)**i/(2*i+1), 2*i+1) for i in range(100000)]
arctan = Polynomial(terms)
print(4 * arctan(1))
```

Hmm...

If you want to get really fancy...

For those of you who want an extra challenge, write a class for complex numbers that implements addition, multiplication, and division. Then, make polynomials with complex coefficients. See if you can use Newton's method to find complex roots. Maybe, take a look at http://www.polynomiography.com/ for inspiration.