CSE 341, Spring 2018, Assignment 8
Ruby Project
Due: Wednesday May 30, 10:00pm

The purpose of this assignment is to give you additional experience programming in Ruby, including how to integrate new classes with existing system ones (particularly numeric ones).

30 points total; up to 10% extra for the extra credit question.

You can use at most ~~1 late day~~ 2 late days for this assignment (the TAs need to get the grading done).

Define a Ruby class `Polynomial` that represents symbolic polynomials in n variables. Instance of this class should understand the messages `+`, `-`, `*`, `to_s`, and `initialize`, along with any helper methods you need. Any helper methods that aren't used outside the polynomial class should be either private or protected. The `+` method should return a new polynomial that is the sum of the receiver and the argument; and similarly for `-` and `*`. The `to_s` method must print out the polynomial in normalized form, as a sum of 1 or more terms. Each term should be a coefficient (which can default to 1) times zero or more variables. Any term with a zero coefficient should be dropped. There shouldn't be multiple terms with the same set of variables — these should be combined. The only other requirement is that the expression must be legal Ruby code, which would evaluate to the correct value if you were to give bindings for any variables — otherwise you can print out the polynomial in any convenient form.

Also define a `asPolynomial` method for Numeric that converts a number to a polynomial (with a single term with the number as the coefficient and no variables); and a `asPolynomial` method for String that converts a string to a polynomial with one variable, whose name is the string.

Finally, polynomials should interoperate correctly with ordinary Ruby numbers: if `p` is a polynomial, `2*p`, `p*2`, `2+p`, `p+2`, `2-p`, and `p-2` should all work.

You should define a set of unit tests that tests all this functionality.

`3.asPolynomial` should return a new polynomial with a single term, consisting of a coefficient of 3 and no variables. In the sample solution it prints as `Polynomial(3)`.

`"x".asPolynomial` should return a new polynomial with a single term with a coefficient of 1 and a single variable x. In the sample solution it prints as `Polynomial(x)`.

Here is some sample output. We first declare a couple of variables `x` and `y` to hold polynomials, and then use them in expressions. Notice that terms with the same variables are combined (even for example if the terms are written as `2*x*y` and `y*3*x` — these both have the same variables, namely `x` and `y`). Also in the sample solution, minus isn't handled particularly elegantly for `to_s` — that's OK (the expression part just needs to be legal Ruby).

```
>> x = "x".asPolynomial
 => Polynomial(x)

>> y = "y".asPolynomial
 => Polynomial(y)

>> -10.asPolynomial
 => Polynomial(-10)

>> 2*x*y +3
 => Polynomial(2*x*y + 3)
```

```
>> (x+3)*y
 => Polynomial(x*y + 3*y)

>> x-8
 => Polynomial(x + -8)

>> 2*x*y*x*x +3
 => Polynomial(2*x*x*x*y + 3)

>> (x+1)*(x-1)
 => Polynomial(x*x + -1)

>> (3*x+5)*0
 => Polynomial(0)

>> 10*y+3*x
 => Polynomial(3*x + 10*y)

>> 2*x*y + x*3*y
 => Polynomial(5*x*y)

>> 10*x*y + 1 + y*3*x*2 + 10
 => Polynomial(16*x*y + 11)

# variable names can be any legal Ruby variable name
>> squid = "squid".asPolynomial
 => Polynomial(squid)
```

Turnin: Turn in two .rb files: one the polynomial class definition, along with any additional methods for system classes; and another file with unit tests.

Hints: Picking a good representation for your polynomial will make your code much cleaner. Represent your polynomial in a normalized form, to make it easy to combine like terms and for printing. Consider using `Hash`.

If you've defined a `asPolynomial` method for `Numeric`, then it should be easy to make expressions like `x+2` work. But what about `2+x`? That ought to work as well, but here 2 is getting the message + with a `Polynomial` as an argument. To make this work, you'll need to define an appropriate `coerce` method for `Polynomial`. See the complex number example on the class web page, as well as the Ruby documentation. And also try `coerce` on integers and floats to explore coercion works in Ruby. The RomanNumeral example in the printed *Programming Ruby* book provides an additional example.

In the sample solution I always constructed polynomials (both for testing in irb and the unit tests) by binding some variables to simple polynomials (with just a single variable), and then building more complex ones using `+`, `-` and `*`. The sample output illustrates this.

We are not expecting `Polynomial.new("10*x^2 + y")` to work (which would require implementing your own parser) — instead, let the Ruby language itself do the work, and construct this polynomial by evaluating the following code:

```
x = "x".asPolynomial
y = "y".asPolynomial
10*x*x + y
```

Exactly how you implement the initialize method for `Polynomial` is up to you — it's not part of the spec, so do whatever is convenient and a clean solution. In the sample solution, calling `new` with no arguments returns the 0 polynomial. There are two optional parameters: a variable and a coefficient. I only used these parameters when implementing `asPolynomial` for `Numeric` and `String` — for everything else, just build polynomials from simpler ones.

Extra Credit (max 10% extra):

Clean up printing to deal with minus in a cleaner way. (This is pretty easy.)

Add support for exponentiation by defining the `**` operator for polynomials. You can assume that the exponent will be a non-negative integer. In addition, clean up the way polynomials are printed (and represented internally) by using exponents rather than just repeating variables in terms. For example `x*10*x*x` should result in `Polynomial(10*x**3)`, and `(x+1)**3` should result in `Polynomial(x**3 + 3*x**2 + 3*x + 1)`.