CSE 1729 - Introduction to Principles of Programming	Due: September 10, 2018
Problem Set 1	

Honor code. As a student in 1729, you have pledged to uphold the **1729 honor code**. Specifically, you have pledged that any work you hand in for this assignment must represent your individual intellectual effort.

1. Write Scheme definitions for the functions below. Use the interpreter to try them out on a couple of test cases to check that they work, and include this output with your solutions.

   NOTE: Scheme provides built-in support for exponentiation (via the `expt` function, defined so that `(expt x y)` yields x^y). For the exercises below, however, please construct the functions $x \mapsto x^k$ using only $*$ and function application.

   (a) `square`, the function $\texttt{square}(x) = x^2$.

   (b) `cube`, the function $\texttt{cube}(x) = x^3$.

   (c) `sp`, the polynomial function $\texttt{sp}(x) = (x^5 + 11x^4 + 24x^3 - x + 39)^2$.
   (Hint: Of course it is possible to expand $(x^5 + 11x^4 + 24x^3 - x + 39)^2$ as a polynomial of degree 10 and write a Scheme function to compute this by brute force. You can avoid all this computation by defining `p` in stages—first define the polynomial $q(x) = x^5 + 11x^4 + 24x^3 - x + 39$ as a Scheme function; now use this function to define `sp`.)

   (d) Using your function `cube`, write the function $\texttt{eighth}(x) = x^8$.

   (e) Using the function `eighth`, write the function $\texttt{sixty-fourth}(x) = x^{64}$. Recall that $64 = 8 \times 8$.

2. Write and test the following functions that deal with triangles.

   (a) `(is-triangle? a b c)`, a function of three parameters (the lengths of the three sides) that evaluates to `#t` if a triangle with these three sides could exist, and `#f` if it could not. The lengths are in no particular order.

   (b) `(area a b c)`, a function of three parameters (the lengths of the three sides of a triangle, in any order) that calculates the area of that triangle.

   Hint: use Heron's formula. If the three side lengths are a, b, and c:

$$\text{Area} = \sqrt{s(s-a)(s-b)(s-c)}$$
$$\text{where } s = \frac{a+b+c}{2}.$$

   Note: Scheme has a built-in function, `sqrt`, which can be used to calculate the square root of a number. For example, `(sqrt 4)` evaluates to 2.

(c) `(op-angle a b c)`, a function of three parameters (the lengths of the three sides of a triangle) that calculates the angle opposite side `a` of that triangle in radians.

Hint: Remember the cosine law. If the three side lengths are a, b, and c:

$$\cos A = \frac{b^2 + c^2 - a^2}{2bc}$$

where A is the angle opposite a. You can use the `acos` function to compute the arccosine function, the inverse of the cosine (in radians).

3. We will see the Fibonacci numbers a good deal in this course. We define this sequence:

$$\text{Fib}_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ \text{Fib}_{n-1} + \text{Fib}_{n-2} & \text{otherwise} \end{cases}$$

So the sequence looks like 0,1,1,2,3,5,13,21,...

There is a closed-form formula for Fibonacci numbers:

$$\text{Fib}_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right]$$

Write a Scheme function (fib-cf n) that calculates the `nth` Fibonacci number using that closed-form formula. For this problem you may use the `expt` function mentioned above. Observe how your function works as `n` gets larger.

4. Remember the *Quadratic formula*, which can be used to find the roots of a quadratic eguation? For a quadratic equation $ax^2 + bx + c = 0, a \neq 0$, the formula is

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Notice that this gives us two different roots (because of the $\pm$) whenever $b^2 - 4ac \neq 0$.

Write the following Scheme functions.

(a) `(root1 a b c)` that gives us the root corresponding to the plus in the $\pm$ in the quadratic formula (that is, calculate $\frac{-b+\sqrt{b^2-4ac}}{2a}$).

(b) `(root2 a b c)` that gives us the root corresponding to the minus in the $\pm$ in the quadratic formula (that is, calculate $\frac{-b-\sqrt{b^2-4ac}}{2a}$).

(c) `(number-of-roots a b c)` which calculates the number of distinct roots to the equation $ax^2 + bx + c = 0, a \neq 0$ (which will either be 1 or 2).

(d) `(real-roots?  a b c)` is a boolean function that evaluates to `#t` when the roots of $ax^2 + bx + c = 0, a \neq 0$ are real numbers. Note that you do not have to calculate the roots to determine whether they are real or complex numbers.

Note: there are some common calculations done in the above functions; it may make sense to write some auxiliary functions to make your code simpler.