Reading
You should read the first two chapters of *Introduction to Functional Programming using Haskell* (IFPH).
For an introduction to the Hugs interpreter and using it effectively inside Emacs, I **strongly encourage** you to work through the following lab. However, you do not need to turn this lab in.

Logistics
This homework is officially due in class on **Wednesday, January 22**. However, it comes with an automatic extension: anything submitted by **noon on Thursday, January 23** will be accepted as being on time.
You may work singly or in pairs on this assignment.

What to turn in
You should turn in a hard copy of your source code and a transcript demonstrating convincingly that your code is correct. You should also submit your code (but not the transcript) electronically. Look on the course web page for details on what I will be expecting.

Exercises (due Wednesday, January 22)

1. A **Pythagorean triple** is a collection of three positive integers (say, m, n, and p) such that $m^2 + n^2 = p^2$. For example, 3, 4 and 5 form a Pythagorean triple (since $3^2 + 4^2 = 5^2$), as do 65, 72, and 97.
   
   (a) Write a Haskell function
   
   ```haskell
   allPositive :: Integer -> Integer -> Integer -> Bool
   such that allPositive x y z returns True if x, y, and z are all positive; it should return False otherwise.
   ```

   (b) Write a Haskell function
   
   ```haskell
   pythTriple :: Integer -> Integer -> Integer -> Bool
   such that pythTriple x y z returns True if and only if x, y, and z form a Pythagorean triple.
   Your function should work correctly regardless of the order of the arguments or their sign: for example, pythTriple 3 4 5 and pythTriple 4 5 3 should both return True, whereas pythTriple (-3) 4 5 should return False.
   ```

2. The number of ways to choose n items out of a total of m items (assuming $m \geq n \geq 0$) is given by the formula:

   $$\binom{m}{n} = \frac{m!}{n! (m-n)!}$$

   (a) Write a function
   
   ```haskell
   choose :: Integer -> Integer -> Integer
   ```
such that \( \text{choose } m \ n \) returns the value \( \binom{m}{n} \), provided that \( m \geq n \geq 0 \). If the condition \( m \geq n \geq 0 \) doesn’t hold, then your function should return the value \(-1\).

**Note:** This version of the function should be based on the formula above. Recall that Lab 1 contains code for the factorial function, which you can reuse.

(b) Another way to compute \( \binom{m}{n} \) is to make use of the following recurrence relation:

\[
\binom{m}{n} = \begin{cases} 
\frac{m \cdot (m-1)}{n \cdot (n-1)}, & \text{for } n > 0 \\
1, & \text{when } n = 0 
\end{cases}
\]

Write a function

\[
\text{choose'} :: \text{Integer} \to \text{Integer} \to \text{Integer}
\]

such that \( \text{choose'} m \ n \) uses this recurrence relation to calculate \( \binom{m}{n} \). As above, if the condition \( m \geq n \geq 0 \) does not hold, then your function should return the value \(-1\).

(c) Yet another way to compute \( \binom{m}{n} \) is to make use of the following recurrence relation:

\[
\binom{m}{n} = \begin{cases} 
\binom{m-1}{n} + \binom{m-1}{n-1}, & \text{for } n > 0 \\
1, & \text{when } n = 0 
\end{cases}
\]

Write a function

\[
\text{choose''} :: \text{Integer} \to \text{Integer} \to \text{Integer}
\]

such that \( \text{choose''} m \ n \) uses this recurrence relation to calculate \( \binom{m}{n} \). As above, if the condition \( m \geq n \geq 0 \) does not initially hold, then your function should return the value \(-1\). (Note that some recursive calls will violate this constraint, so you may want to use a helper function.)