Coverage & Logistics
This homework covers material through the seventh chapter of *Haskell: The Craft of Functional Programming* (HCFP).

This homework is officially due in class on **Thursday, February 18**. However, it comes with an automatic extension: anything submitted to the CIS 252 bin near CST 4-226 by **noon on Friday, February 19** will be accepted as 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 that demonstrates convincingly that your code is correct.

Exercises
In writing your functions, make sure to abide by the following constraints:

(i) All of your functions should be **recursive** functions.

(ii) **Do not use** `fst`, `snd`, `head`, `tail`, or `!!` in any of the functions you write. Instead, use Haskell’s pattern-matching capabilities as discussed in lecture.

1. **Use list recursion** to define a Haskell function

   ```haskell
   sumSquares :: [Integer] -> Integer
   sumSquares ns
   ```

   such that `sumSquares ns` calculates the sum of the squares of the numbers in `ns`. For example, `sumSquares [4,1,6,10]` returns 153 (= $4^2 + 1^2 + 6^2 + 10^2$).

2. **Use list recursion** to define a Haskell function

   ```haskell
   whisper :: String -> String
   ```

   such that `whisper str` returns the string obtained by replacing all uppercase letters by their lowercase equivalents; all other characters are left alone. For example, your function should have the following behavior:

   ```
   Main> whisper "One, two. THREE, Four!"
   "one, two. three, four!"
   ```

3. **Use list recursion** to define a Haskell function

   ```haskell
   countCaps :: String -> Int
   ```

   such that `(countCaps cs)` returns the number of characters in `cs` that are uppercase letters. For example, `(countCaps "Mary? She had a LAMB!")` returns 6.

4. **Use list recursion** to define a function

   ```haskell
   interleave :: [a] -> [a] -> [a]
   ```

   that interleaves the elements of two lists; the resulting list should be twice the length of the shorter of the two lists. For example, `interleave "abcde" "XYZ"` should return "aXbYcZ".

5. One common way to store a table is with an **association list**, which is a list of pairs. Each pair `(k,v)` is a table entry, where `k` is the lookup-key and `v` is the value stored for lookup-key `k`. For example, here is an association list that gives a table for the point values of letters in Scrabble:

   ```haskell
   scrabblePoints :: [(Char, Int)]
   scrabblePoints = [('a',1), ('b',3), ('c',3), ('d',2), ('e',1),
                    ('f',4), ('g',2), ('h',4), ('i',1), ('j',8),
                    ('k',5), ('l',1), ('m',3), ('n',1), ('o',1),
                    ('p',3), ('q',10), ('r',1), ('s',1), ('t',1),
                    ('u',1), ('v',4), ('w',4), ('x',8), ('y',4),
                    ('z',10)]
   ```

   Thus, the Scrabble point value of 'd' is 2 and the value of 'z' is 10.

   (a) **Use list recursion** to define a function

       ```haskell
       getVal :: Char -> [(Char, Int)] -> Int
       ```

   such that `(getVal c ps)` returns the value of character `c` in the association list `ps`. For example, `(getVal 'w' scrabblePoints)` should return 4. If the character `c` does not have an entry in the association list `ps`, then 0 should be returned. If the character `c` has more than one entry in the association list `ps`, then the value of the first entry should be returned.

   (b) **Use list recursion** to define a function

       ```haskell
       scoreWord :: String -> Int
       ```
that, given a \texttt{String} of lowercase letters, returns the Scrabble scoring of the word. For example, \texttt{(scoreWord "zowie")} should return 17 (=10+1+4+1+1).

6. \textbf{Use list recursion} to define a Haskell function

\begin{verbatim}
pairUp :: [a] -> [(a,a)]
\end{verbatim}

such that \texttt{(pairUp xs)} returns the list obtained by pairing up the first two elements of \texttt{xs}, then the third and fourth elements, and so on; if \texttt{xs} has an odd number of elements, the final element is paired with itself.

For example, \texttt{(pairUp [3,5,2,9])} returns \texttt{[(3,5),(2,9)]}, whereas \texttt{pairUp "abcde"} returns \texttt{[(‘a’,’b’),(‘c’,’d’),(‘e’,’e’)]}.

7. \textbf{Use list recursion} to define a Haskell function

\begin{verbatim}
simplify :: String -> String
\end{verbatim}

such that \texttt{simplify cs} “condenses” \texttt{cs} by removing all consecutive repetitions of the same character. For example, \texttt{simplify "abaaacabbbbbacccc"} returns \texttt{"abacabac"}. 