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

This homework is officially due in class on **Thursday, February 4**. However, it comes with an automatic extension: anything submitted to the CIS 252 bin near CST 4-226 by **noon on Friday, February 5** will be accepted as being on time.

You may work singly or in pairs on this assignment.

Exercises

**General advice:** Work on one problem at a time: get it working before moving on to the next problem. (Writing all your code at once will likely result in a series of errors and a lot of frustration on your part. It is usually much easier to work in small increments.)

Generate your final transcript only after all of your code works. Take pity on the graders, and make sure that your solutions will be easy to grade/verify.

1. Write a Haskell function

   \[
   \text{compareChars :: Char} \to \text{Char} \to \text{Char} \to \text{String}
   \]

   such that \((\text{compareChars a b c})\) returns a string indicating how many of the characters \(a\), \(b\), \(c\) are equal to one another:

   - If all three are the same, it returns "All equal".
   - If exactly two of them are the same, it returns "Two match".
   - If all three are different, it returns "All distinct".

   For example, \((\text{compareChars 'R' (toUpper 'r') 'R'})\) returns "All equal", \((\text{compareChars 'R' 'r' 'R'})\) returns "Two match", and \((\text{compareChars 'a' 'B' 'c'})\) returns "All distinct".

2. 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.

   Write a Haskell function

   \[
   \text{pythTriple :: Integer} \to \text{Integer} \to \text{Integer} \to \text{Bool}
   \]

   such that \(\text{pythTriple x y z}\) returns True if and only if \(x\), \(y\), and \(z\) form a Pythagorean triple.

   Your code should work correctly regardless of the order of arguments or their signs: for example, \(\text{pythTriple 3 4 5}\) and \(\text{pythTriple 4 5 3}\) should both return True, whereas \(\text{pythTriple (-3) 4 5}\) and \(\text{pythTriple 13 4 5}\) should return False.

3. Write a Haskell function

   \[
   \text{splitNum :: Integer} \to \text{(Integer, Integer, Integer)}
   \]

   with the following behavior:

   - When \(\text{num}\) is a nonnegative number with at most three digits, \(\text{splitNum num}\) returns a triple containing the one's digit, the ten's digit, and the hundred's digit (in that order) of \(\text{num}\).
   - When \(\text{num}\) is negative or contains more than three digits, \(\text{splitNum num}\) returns the triple \((-1,-1,-1)\).

   For example, \(\text{splitNum 824}\) returns \((4,2,8)\), \(\text{splitNum 27}\) returns \((7,2,0)\), and \(\text{splitNum (-4)}\) returns \((-1,-1,-1)\).

   
   In your code, to create a triple, simply use parentheses and commas in the obvious way: for example, if \(x\) has the value 10, then \((x, x+1, x-3)\) evaluates to the triple \((10,11,7)\).

4. A small car-rental agency offers two different pricing plans to its customers:

   - The **standard-pricing plan** takes into account (i) how long the customer keeps the car and (ii) how many miles the customer puts on the car:
     - The company charges $135 for up to two days' rental, and $49 per additional day.
     - Each mile put on the car costs $0.70.
   - Under this plan, a customer who rents a car for 10 days and puts 37 miles on it will pay $552.90 (= $135 + (10-2)*$49 + 37*$0.70), whereas a customer who rents a car for one day and puts 315 miles on it will pay $355.50 (= $135 + 315*$0.70).
• The **unlimited-mileage plan** takes into account only how long the customer keeps the car: the customer pays $68 per day, regardless of how many miles they put on the car.
Under this plan, a customer who rents a car for 10 days will pay $680, regardless of the number of miles driven.

*For simplicity, we will assume that billing supports fractional days and mileage: that is, a customer might pay for 3.8 days and 100.72 miles.*

(a) Write a Haskell function

```haskell
    cost :: Float -> Float -> Bool -> Float
```

such that *(cost days miles std)* calculates the cost of renting a car, where *days* indicates the number of days, *miles* indicates the number of miles driven, and *std* is true precisely when the standard-pricing plan is used. *If either days or miles is negative, your function should return -1.*

For example, your function should have the following behavior:

*Main> cost 10 37 True
552.9
*Main> cost 1 315 True
355.5
*Main> cost (-10) 315 False
-1

(b) Write a Haskell function

```haskell
    bestPlan :: Float -> Float -> String
```

such that *(bestPlan days miles)* determines which pricing plan is the better choice (i.e., less expensive) for a customer who expects to keep the car for *days* days and put *miles* miles on the car. It returns its answer as a string: "Standard", "Unlimited", or "Same cost". *You may assume that days and miles are nonnegative.*

For example, *(bestPlan 10 37)* should return "Standard", whereas *(bestPlan 1 315)* should return "Unlimited".

**What to turn in**

Turn in hard copies of (1) your source code and (2) a transcript demonstrating convincingly that your code is correct. As always, include a disclosure cover sheet.

**Note:** To provide a convincing demonstration of correctness, include test cases that handle the various input possibilities your code may encounter.