HASKELL PROGRAMMING PROBLEM SET 1

LENNART JANSSON AND BRANDON AZAD

Read Chapter 2 of *Learn You a Haskell*, then use the techniques described to solve the following problems. No more advanced techniques are needed.

Simple Functions

Problem 1. Triangle numbers

Write a function `nthTri` that takes an `Int` n and returns the nth triangle number.

```
> nthTri 0
0
> nthTri 2
3
> nthTri 4
10
```

Problem 2. Palindromes

Write a function `isPalindrome` that takes a string and returns `True` if it's a palindrome and `False` if it's not.

```
> isPalindrome "racecar"
True
> isPalindrome "palindrome"
False
```

Problem 3. Parity

Write a function `sameParity` that takes a list of `Ints` and returns `True` if the first and last elements of the list have the same parity (even or odd) and `False` if they don't.

```
> sameParity [1, 4, 2]
False
> sameParity [3, 2, 6, 7]
True
```

List Comprehensions and Ranges

Problem 4. Summing integers

Write a function `specialSum` that takes an `Int` n and returns the sum of all positive integers less than n not divisible by 3 or 7.

```
> specialSum 8
12
```

Problem 5. Squares and ranges

Write a function `isSquareBetween` that takes three `Int`s, a, b, and c, and returns `True` if some integer between b and c inclusive, when squared, is a.

```
> isSquareBetween 9 2 3
True
```

```
> isSquareBetween 16 2 3
False
```

To do the next problem, you might need the function `concat`, which takes a list of lists and concatenates all of them to make a single list.

```
> concat ["ab", "cd", "ef"]
"abcdef"
```

Problem 6. String manipulation

Write a function `tripleLetters` that takes a string and returns the string with every letter repeated three times, with every triple of letters separated by a `-`.

```
> tripleLetters "Hello"
"HHH-eee-lll-lll-ooo"
```

Challenge Problems

These can be done with only the functions described in Chapter 2!

Problem 7. Combinations

Write a function `twoCombo` that takes a list of `Int`s and returns a list of all unordered combinations without replacement of 2 elements in the list. You can assume the list already consists of distinct elements.

```
> twoCombo [1, 2, 3, 5]
[(1, 2), (1, 3), (1, 5), (2, 3), (2, 5), (3, 5)]
```

Problem 8. More combinations

Was that too easy? Write a function `twoCombo'` that does the same thing as `twoCombo`, but works for any type, not just types that can be compared with `==` or `<`. Again assume the list already consists of distinct elements. (If you've read about types, this means the function must be able to have the type signature `twoCombo' :: [a] -> [(a, a)]`.)

Types and Typeclasses

Read Chapter 3 of *Learn You a Haskell*, then it's time for a round of... Name That Type! You can use `:t` in `ghci` to give the answers, or practice guessing the types yourself as an exercise.

Problem 9. Name That Type!

Give the types of all the following expressions.

(1) `"hello" ::` ________________
(2) `3.0 ::` ________________
(3) `[1, 3, 5] ::` ________________
(4) `func1 a b = a ++ b`
`func1 ::` ________________
(5) `show 100 ::` ________________

(6) `func2 a = a * a`
`func2 ::` ________________
(7) `tail ::` ________________
(8) `func3 x = [succ e | e <- x]`
`func3 ::` ________________
(9) `func4 x y = [(show a, show b) | a <- x, b <- y]`
`func4 ::` ________________

Problem 10. More type annotations

Annotate all the functions you wrote above with explicit type signatures.