Laboratory: Haskell (2)

Summary: We continue our exploration of the Haskell programming language.

Prerequisites: The first Haskell lab Tate, Section 8.3.

Contents:

- Preparation
- Exercises
  - Exercise 1: Characters to Numbers
  - Exercise 2: Strings to Numbers
  - Exercise 3: Strings to Numbers, Revisited
  - Exercise 4: Selecting Elements
  - Exercise 5: Fun with Patterns
- For Those with Extra Time

Preparation

a. Create a directory for the lab.

b. Open a browser window on Tate’s examples, in case you want to try any of them.

Exercises

Exercise 1: Characters to Numbers

Write or find a function that converts digit characters to the corresponding number. (If you can’t find one, you should find the `fromEnum` function useful.)

```
Prelude> digitToInt '5'
5
```

Exercise 2: Strings to Numbers

Write (do not find) a function that converts strings that represent integers to the corresponding integer. For example,
Exercise 3: Strings to Numbers, Revisited

This exercise is based on a problem from Tate.

As you no doubt have noted, we often write larger numbers with commas to separate the portions. E.g., 3,123,876. Revise your atoi function from the previous step to handle situations like this.

Exercise 4: Selecting Elements

a. Write a function, everyOther, that takes a list as input and returns a list in which you’ve selected every other element of the input list. E.g.,

```
Prelude> everyOther [1,2,3,4]
[1,3]
```

b. Write a function, everyThird, that takes a list as input and returns a list in which you’ve selected every third element of the input list. E.g.,

```
Prelude> everyThird [1,2,3,4,5,6,7]
[1,4,7]
```

c. What do you expect the composition of everyOther and everyThird to give?

d. Check your answer experimentally.

e. You likely wrote very similar code for everyOther and everyThird. By this point in your career, you should have learned that once you find yourself duplicating code, you should factor out the common code.

Write a function, everyNth N lst, that selects one of N elements in a list.

f. Rewrite everyOther and everyThird in terms of everyNth.

Exercise 5: Fun with Patterns

a. Consider the following definition.
stupid n = stupid (n-1)

Why might this definition be useful in experimenting with programs?

b. As you may recall from our first day of Haskell, one of the nice things about lazy evaluation is that you can define if yourself and expect that it behaves the way you would expect. Consider the following definitions.

```
    myif True consequent alternate = consequent
    myif False consequent alternate = alternate
```

What do you expect the following to produce?

- myif True (stupid 1) 5
- myif True 5 (stupid 1)
- myif False (stupid 1) 5
- myif False 5 (stupid 1)

c. Check your answer experimentally.

d. Consider the following “backwards if”

```
    bwif consequent alternate True = consequent
    bwif consequent alternate False = alternate
```

What do you expect the following to produce?

- bwif (stupid 1) 5 True
- bwif 5 (stupid 1) True
- bwif (stupid 1) 5 False
- bwif 5 (stupid 1) False

e. Check your answer experimentally.

f. Consider the following definitions.

```
    whatever True x = 1
    whatever False True = 2
    whatever False False = 2
```

What do you expect the following to produce?

- whatever True (stupid 1)
- whatever False (stupid 1)

g. Check your answer experimentally.

h. Consider the following definitions.
everwhat x True = 1
everwhat True False = 2
everwhat False False = 2

What do you expect the following to produce?

- everwhat (stupid 1) True
- everwhat (stupid 1) False

i. Check your answer experimentally.

**For Those with Extra Time**

If you find yourself with extra time, start reading the next section of Tate.