

# 211lab01

## Lists

Originally created by Dr. Griener

*Instructions for students & labbies:* Start with short group discussion. Students use DrScheme on the exercises at their own pace, while labbies wander among the students, answering questions, bringing the more important ones to the lab's attention. Example solutions are attached at the bottom of the page.

- Follow the design recipe, including using templates.
- Use the stepper on at least a few examples.
- It's likely you won't have time to do all the examples during lab. Students should look over the rest on their own.

## Review: Definitions of Lists

- What is the data definition of a list of numbers?
- What are the constructors, selectors, and predicates for lists of numbers, and what are their contracts?
- What are some examples of lists of numbers?
- What is the template for functions on lists of numbers?

## Practice Exercises

### Functions on Lists of Numbers

All of these examples use the same input type: a list of numbers. So, don't forget to reuse what you can to save yourself some time. Use your data examples (i.e., sample lists of numbers) for each of the function examples (i.e., what the function does on each of those inputs). Copy-and-paste the type's template as your starting point for each function.

1. Develop a program `count-positive` which takes a list of numbers and returns a count of how many are positive, i.e., greater than zero. Use the stepper on `count-positive` applied to a list of length 2. How many times is your function called (counting both the initial call and recursive calls)?
2. Develop a program `map-add` which takes a number and a list of numbers and returns a list of numbers, where the resulting list is constructed by adding the given number to each element in the list. E.g.,

```
> (map-add 3 (cons 1 (cons 4 (cons 8 empty))))  
(cons 4 (cons 7 (cons 11 empty)))
```

Later in the course, we'll generalize this idea of *mapping* some operation over a list to generate a new list of the results.

3. Develop a program `filter-positive` which takes a list of numbers and returns a list of all those numbers which are positive. E.g.,

```
> (filter-positive (cons 1 (cons -5 (cons 3 empty))))  
(cons 1 (cons 3 empty))
```

Later in the course, we'll generalize this idea of *filtering* a list by some test condition.

### Functions on Structures

We've seen lists of symbols and lists of numbers; you can of course have a list with elements of any given type--including other structures like `_posn_s`. This exercise just uses ideas you've already learned, but in new contexts.

1. Define the type list of `_posn_s`.
2. Write the template for list of `posn_s`. (NB: There are two non-primitive types here, an inductive type and one that simply consists of a structure. So writing a program that processes a list of positions may require using two templates; a template for lists of `_posn_s` and a template for processing `_posn_s`. If you need to use two templates, you *also* need to write two functions, one to process a list of positions and one to process positions.)
3. Write the function `positive-quadrant?`, which takes in a single `posn` and returns true only when its components are both positive.
4. Develop `count-positive-quadrant`, which takes a list of `_posn_s` and returns a count of those in the upper-right quadrant.
5. Develop `filter-positive-quadrant`, which takes a list of `_posn_s` and returns a list of those in the upper-right quadrant.

## Optional: Variants of the List Definition

1. Make a data definition for non-empty lists of numbers. Hint: The base case should not be empty, since that is not a non-empty list of numbers! What is a description of the shortest non-empty list of numbers?
2. Develop a program which takes a non-empty list of numbers and returns the average (a.k.a, arithmetic mean) of all the numbers.

3. Develop a program which takes a standard, potentially empty list of numbers and returns the average of all the numbers. For this example, arbitrarily define the average of an empty list to be false.

Here are [two reasonable definitions](#) for non-empty lists.

- **Solution:** [lab01.ss](#)

## !! Access Permissions

- Set ALLOWTOPICCHANGE = Main.TeachersComp211Group