COMP105: Programming Paradigms
Lab Sheet 2

Filename: Lab2.hs

This lab covers material from Lectures 5 and 6. Enter all solutions in a file named Lab2.hs, and upload it to Codegrade when you are finished.

1. **Tuples.** We covered tuples in Lecture 5.

   (a) Write a function `square_and_cube x` that returns a two-element tuple, where the first element is x*x, and the second is x*x*x

   (b) Write a function `add_tuple (a, b)` that takes a tuple with two elements called a and b, and returns $a + b$.

   (c) Write a function `first` that takes a tuple with two elements, and returns the first element. Write a function `second` that takes a tuple with two elements and returns the second element. (These functions are known as `fst` and `snd` in Prelude.)

   (d) Write a function `swap` that takes a two-element tuple, and swaps the order of the elements of that tuple.

   (e) Write a function `two_to_three (a, b) c` that takes a tuple with two elements and a second argument c, and returns a three element tuple that contains a, b, and c in that order.

2. **Lists.** We covered lists in Lecture 5. These exercises cover basic operations on lists.

   (a) Use the `head` function to write a function `head_squared list` that takes a list as an argument, and returns the square of the head of that list.

   (b) Use the `!!` operator, write a function `third list` that returns the third element of the input list.

   (c) Using the `tail` library function, Write a function `second_tail list` that returns all but the first two elements of `list`.

   (d) Using the `head` and `tail` library functions, write a function `third_head list` that returns the third element of the input list.

   (e) Write a function `first_plus_last list` that takes a list with at least two elements, and returns the result of adding the first and last elements of the list together. Remember that the `last` function from Prelude returns the last element of a list.

(f) Using the `:` operator, write a function `prepend_two list a b` that takes a list and two other arguments, and returns a new list with a and b added to the front.

3. **List functions.** These exercises cover the Prelude list functions discussed in Lecture 5.

   (a) Use the `length` function to write a function `two_lengths list1 list2` that takes two lists, and returns the sum of their lengths.

   (b) Use the `reverse` function and the `++` operator to write a function `make_palindrome list` that returns the list followed by the reverse of the list.

   (c) Use the `sum` and `product` functions to write a function `sum_and_product list` that returns a tuple where the first element is the sum of the list, and the second element is the product of the list.

   (d) Use the `take` and `drop` functions to write a function `four_through_six list` that returns a list containing elements four, five, and six of the input list.

   (e) Use the `elem` function to write a function `both_in list x y` that returns True if both `x` and `y` are in `list`.

4. **List ranges.** These exercises cover List ranges, a concept that we encountered in Lecture 6. There is no need to write anything in your file for these questions. In GHCI, use a list range to write a query that outputs:

   (a) The list of all numbers between 101 and 200.

   (b) The list of all even numbers between 1000 and 1050.

   (c) The list of all numbers between 20 and 1 counting backwards.

   (d) An infinite list of all numbers divisible by 3 starting from 999. Press control+c to stop the print out.

5. **List comprehensions.** These exercises cover list comprehensions, which we encountered in Lecture 6.

   (a) In GHCI, use the `^` operator to write a list comprehension that outputs the first ten powers of two.

   (b) Write a function `only_odds list` that returns only the odd elements of the input list.

   (c) Write a function `between a b list` that takes two numbers $a < b$, and returns the elements of `list` that are (strictly) between a and b.

   (d) Write a function `number_of_es string` that returns the number of times that `'e'` occurs in the input string.

   (e) (*) Write a function `proper_fizzbuzz` that returns an infinite list with the following properties. In position i of the list,

   - if i is divisible by 3 then the list should contain `"fizz"`
   - if i is divisible by 5 then the list should contain `"buzz"`

- if i is divisible by both 3 and 5 then the list should contain `"fizzbuzz"`
- if i is not divisible by 3 or 5 then the list should contain the number i (the `show` function from Prelude will turn an integer into a string.)

Lab complete. Remember to upload Lab2.hs to Codegrade!