CpSc 418

Mini Assignment 1

Due: Sept. 10, 2012, 3:29pm.
No late solutions accepted.

Please submit your solution using the handin program. Submit the program as

cs418 mini-09-10

This requires you to have an account on the UBC Computer Science undergraduate machines. If you need an account, go to: `https://www.cs.ubc.ca/students/undergrad/services/account` to request one.

Your submission should consist of one file:

`mini.erl` – Erlang source (ASCII text).

Important: Please give your Erlang module the name specified here, `mini.erl`, and please use the names stated in the problem for the functions you are asked to write. Points will be taken off if you use other names. You can get a template for this (i.e. most of the work done for you) here:
`http://www.ugrad.cs.ubc.ca/~cs418/2013-1/mini/09.10/mini.erl`
Each question has a reasonable, one-line solution. The entire assignment requires you to write nine lines of code.

1. Write an Erlang function `nonMultiple(P, List)` where `P` is an integer and `List` is a list of integers. `nonMultiple(P, List)` returns a list of all the elements of `List` that are *not* divisible by `P`. Here's an outline for your function:

```
nonMultiple(_P, []) -> You write this part;
nonMultiple(P, [Head | Tail]) when (Head rem P) == 0 -> You write this part;
nonMultiple(P, [Head | Tail]) when (Head rem P) /= 0 -> You write this part.
```

   Note that `rem` is the built-in "remainder" operator in Erlang. For example, `17 rem 5` produces the value `2`.

   Here are a few test cases:

```
1> c(mini).
ok
2> mini:nonMultiple(2, [2, 3, 4, 5, 6, 7, 8, 9, 10]).
[3, 5, 7, 9]
3> mini:nonMultiple(3, [5, 7, 9, 11, 13, 15, 17, 19, 21, 23]).
[5, 7, 11, 13, 17, 19, 23]
4> mini:nonMultiple(5, [7, 11, 13, 17, 19, 23, 25, 29, 31, 35, 37]).
[7, 11, 13, 17, 19, 23, 29, 31, 37]
```

2. Erlang has function-valued expressions (more formally known as "lambda expressions"). See:
   `http://www.erlang.org/doc/reference_manual/expressions.html#id79056`
   or `http://learnyousomeerlang.com/higher-order-functions#anonymous-functions`
   As an example, consider adding the following function to `mini.erl`:

```
plus(N) -> fun(X) -> X+N end.
```

   The function `plus(N)` returns a function that adds `N` to its argument and returns the result. For example,

```
5> c(mini).
ok
6>F = mini:plus(5).  #Fun<mini.0.73342375> 7>F(14).  19
```

   Add a function to module `mini` called `times(N)`. The function `times(N)` should be a function that multiplies its argument by `N` and returns the result.

3. Erlang provides functions that implement many common patterns of computation. This question, and the next few will use functions from the standard library module `lists`, see
   http://www.erlang.org/doc/man/lists.html
   or http://learnyousomeerlang.com/higher-order-functions#maps-filters-folds
   In this question we will use `lists:map(Fun, List1) -> List2`. The `map` function constructs `List2` by applying the function `Fun` to each element of `List1`. For example,

   ```
   8> lists:map(fun(X) -> 2*X end, [1, 2, 3, 4]).
   [2, 4, 6, 8]
   ```

   Write a function, `sqList(List1) -> List2`, where `List2` is obtained by squaring each element of `List1`. For example

   ```
   9> mini:sqList([1, 2, 3, 4]).
   [1, 4, 9, 16]
   ```

   Your implementation must use the `lists:map` function.

4. Another useful function from the `lists` module is "fold", it comes in two versions `lists:foldl` (fold from the left), and `lists:foldr`. The fold functions can be used to combine the elements of a list to produce a single value. This question uses `lists:foldl(Fun, Acc0, List) -> Value`. As an example,

   ```
   lists:foldl(Fun, Acc0, [X, Y, Z]) -> Fun(Z, Fun(Y, Fun(X, Acc0))).
   ```

   Or, to make that more concrete

   ```
   10> lists:foldl(fun(X, Y) -> X*Y end, 1, lists:seq(1, 10)).
   3628800    % 10!
   ```

   Write a function, `sum(List)`, that computes the sum of the elements of `List`. For example:

   ```
   11> mini:sum([1, 2, 3, 4]).
   10
   12> mini:sum([1, 4, 9, 16]).
   30
   ```

5. Combine your functions `sqList` and `sum` to produce a function, `euclidLength(List)` that computes the Euclidean length of the vector represented by the list. This function should return the square-root of the sum of the squares of the elements of `List`. For example:

   ```
   13> mini:euclidLength([3, 4]).
   5.0
   14> mini:euclidLength([3, 4, 12]).
   13.0
   ```

   Please use the `math:sqrt` function in your implementation.

6. (a) Write a function called `collaborate` that returns a list of strings. The list is the names of any one you collaborated with on this assignment. If you did not collaborate with anyone, that's fine, just return an empty list. Remember that collaboration is great, but you must list your collaborators.

   (b) Write a function called `timeSpent` that returns a list of four elements. The first two are the number of hours and minutes that you spent reading to learn the Erlang that you needed for this assignment. The last two are the number of hours and minutes that you spent programming (including debugging and testing).

Why?

What is to goal of this mini-assignment?

To achieve a basic familiarity with Erlang including simple expressions and functions. After doing this assignment, you should be able to read simple examples written in Erlang.

What is left out?

Lots of things.

- Functional language features such as pattern matching, tuples, `case` and `if` statements, list comprehensions.
- Support for parallel and distributed programming: `spawn`, send (i.e. the `!` operator), and `receive`.
- An overview of handy functions from the Erlang library.

How does this fit with the September 10 lecture?

The September 10 lecture will give an overview of functional program design. We'll look at some common algorithms such as sorting, and see how to design a functional implementation. In this process, we'll go over some of the other features of Erlang such as pattern matching. I'll also give a brief introduction to processes and communication in preparation for the September 12 mini-assignment and lecture.