Lists – doing it yourself

This activity gives you a chance to try out a larger problem using what you have already learned about functional programming in Erlang. You have at your disposal

- Erlang data types: numbers, atoms, booleans, strings, lists and tuples, and combinations of them.
- Erlang modules, containing Erlang function definitions.
- Function definitions making use of pattern matching to distinguish between cases and to extract components from complex data structures.
- Recursion to help you to define functions over numbers and lists.

In solving these problems you'll need to think about different ways of modelling the data involved, and then how to define the functions that you need to over these data representations.

Rather than trying all the examples, it's recommended to pick one and follow that example through. They are all somewhat open-ended, so that you are able to keep on adding features to them.

Indexing a file

The aim of this exercise is to index a text file, by line number. We can think of the input being a text string, where the lines are separated by newline symbols: `$\n`. The output of the main function should be a list of entries consisting of a word and a list of the ranges of lines on which it occurs.

For example, the entry

```
{ "foo" , [{3,5},{7,7},{11,13}] }
```

means that the word `"foo"` occurs on lines 3, 4, 5, 7, 11, 12 and 13 in the file.

To take the problem further, you might like to think about these ways of refining the solution.

- Removing all short words (e.g. words of length less than 3) or all common words (you'll have to think about how to define these).
- Sorting the output so that the words occur in lexicographic order.
- Normalising the words so that capitalised (`"Foo"`) and non capitalised versions (`"foo"`) of a word are identified.
- Normalising so that common endings, plurals etc. identified.
- (Harder) thinking how you could make the data representation more efficient than the one you first chose. This might be efficient for lookup only, or for both creation and lookup.

Text processing

In word processing systems it is customary for lines to be filled and broken automatically, to enhance the appearance of the text. This exercise sheet is no exception. Input of the form

```
The heat bloomed     in December
 as the   carnival  season
            kicked into  gear.
Nearly helpless with sun and glare, I avoided Rio's brilliant
sidewalks
  and glittering beaches,
panting in dark   corners
     and waiting out the inverted southern summer.
```

would be transformed by *filling* to

```
The heat bloomed in December as the
carnival season kicked into gear.
Nearly helpless with sun and glare,
I avoided Rio's brilliant sidewalks
and glittering beaches, panting in
dark corners and waiting out the
inverted southern summer.
```

Define a function that takes an input file in Erlang as a string (list) of characters. and a line length `len` (a positive integer) and which returns a list of lines, each of which is *filled* to include the maximum number of words up to the overall length `len` (including punctuation characters).

You should think about how best to represent lines in solving this problem: is a string the best representation to or is there a better alternative? To take the problem further you might like to try the following.

- To align the right-hand margin, the text is *justified* by adding extra inter-word spaces on all lines but the last:

  ```
  The heat bloomed in December as the
  carnival  season  kicked into gear.
  Nearly helpless with sun and glare,
  I avoided Rio's brilliant sidewalks
  and glittering beaches, panting  in
  dark  corners  and  waiting out the
  inverted southern summer.
  ```

- We've so far just provided one sort of layout. It would be possible to add *formatting commands* of the form `.XX` on a line on their own. Commands could include
  - `.VB` for verbatim (so no formatting, just copy lines from input to output,
  - `.CV` lines are copied from input to output but also each line is centred (if possible),
  - `.LP` for lines filled and aligned to the left,
  - `.RP` for lines filled and aligned to the right, and
  - `.JU` for lines filled and justified.

Supermarket billing

A supermarket billing system will take a sequence of barcodes such as

```
[1234,4719,3814,1112,1113,1234]
```

into a printed bill of the form

```
         Erlang Stores

Dry Sherry, 1lt...........5.40
Fish Fingers..............1.21
Orange Jelly..............0.56
Hula Hoops (Giant)........1.33
Unknown Item..............0.00
Dry Sherry, 1lt...........5.40

Total....................13.90
```

using the data in a simple database such as

```
[ (4719, "Fish Fingers" , 121),
  (5643, "Nappies" , 1010),
  (3814, "Orange Jelly", 56),
  (1111, "Hula Hoops", 21),
  (1112, "Hula Hoops (Giant)", 133),
  (1234, "Dry Sherry, 1lt", 540)]
```

The aim of this exercise is to define the function that will produce the bill from a list of barcodes and the database. You'll need to think about how to structure the solution by defining the right set of auxiliary functions to help you to "divide and conquer" the problem.

To take the problem further you might like to add these features

- You are asked to add a discount for multiple buys of sherry: for every two bottles bought, there is a £1.00 discount.
- Design functions which update the database of bar codes. You will need a function to add new information while removing any entry for the same bar code.
- Re-design your system so that bar codes which do not appear in the database (e.g. `1113` in the example) give no entry in the final bill.