Consolidating – functions over lists

The aim of this activity is to help you to consolidate your work on definition functions over lists, if you would like to do some more practice.

Joining lists together

Two of the principal functions over lists join two lists together (the `++` operator) and `lists:concat/1` joins a list of lists into a single list. For example

```
"hel"++"lo" = "hello"
lists:concat(["goo","d","","by","e"]) = "goodbye"
```

Write your own definitions of these functions. In the case of `++` you'll need to define a function (say) `join/2` as you can't define your own operators in Erlang. **Hint:** think about how you could use `join` (or `++`) in the definition of `concat`.

Testing membership

Define a function `member/2` that tests whether its first argument is a member of its second argument, which is a list. For example,

```
member(2,[2,0,0,1])  = true
member(20,[2,0,0,1]) = false
```

Text handling

Define a function `getWord/1` that extracts the longest word from the beginning of a list. A word is made up of any characters apart from the *whitespace* characters in this list: `[$\n,$\t,$ ]`; these are the newline,tab and space characters. For example,

```
getWord("hello there") = "hello"
getWord(" hello there") = ""
```

(cont.)

Sorting lists

A list can be sorted in a number of ways, including these algorithms described informally:

- *Merge sort:* divide the list into two halves of (approximately) equal length, sort them (recursively) and then merge the results.
- *Quicksort:* split the list into two according to whether the items are smaller than (or equal to) or larger than the *pivot*, often taken to be the head element of the list, sort the two halves and join the results together.
- *Insertion sort:* sort the tail of the list and then *insert* the head of the list in the correct place.

Try to implement each of these sorting algorithms in Erlang.

Permutations

A permutation of a list `xs` consists of the same elements in a (potentially) different order. Define a function that gives all the permutations of a list, in some order. For example,

```
perms([]) = [[]]
perms([1,2,3]) = [[1,2,3],[2,3,1],[3,1,2],[2,1,3],[1,3,2],[3,2,1]]
```