

Exercise: map, filter and list comprehensions

Guy McCusker¹

¹University of Bath

Exercises using map

- ▶ Write a function which calculates the length of a list, using `map` and addition. (You need to add together a big list of 1s).
- ▶ Using a similar idea, write a function which counts how many even numbers there are in a list of numbers.
- ▶ Generalize your function so that it can be given a predicate as well as a list, and it counts how many elements of the list satisfy that predicate, so e.g. `(count zerop mylist)` counts the number of zeros in a list.
- ▶ Can you use a similar idea to write `filter` in terms of `map` and some other list functions?

The last one is a bit harder than the others. Hint: You need to `map` the list so that the elements you want to keep are kept and the ones you want to throw away are erased. How can you apply a function to each element which sometimes returns “nothing”?

List comprehensions

Some functional languages provide a notation known as *list comprehensions*.

A list comprehension looks something like this:

```
[x + y | x <- list1, x < 4, y <- list2]
```

and means “the list of all the values $x + y$, where x is taken from `list1` and satisfies $x < 4$ and y is taken from `list2`”.

So if `list1` is `(1 2 3 4)` and `list2` is `(10 20 30)` then this list comprehension gives

```
(11 21 31 12 22 32 13 23 33)
```

Comprehensions using map and filter

Most list comprehensions can be implemented using map and filter. Let's try to do it!

Implement these list comprehensions using map and filter:

- ▶ `[x+1 | x <- list1]`
- ▶ `[x+1 | x <- list1, x < 4]`
- ▶ `[x+y | x <- list1, y <- list2]`
- ▶ `[x+y | x <- list1, x < 4, y <- list2]`

The first two should be quite easy. The third one is pretty tough (you may need to produce a list of lists and then join them together), but then the fourth one should be easy.

General list comprehensions

We can transform most list comprehensions using map, filter and list concatenation. There are four key transformations. See if you can work out what they are.

- ▶ `[x | x <- list1] = ???`
- ▶ `[(f x) | x <- list1] = ???`
- ▶ `[expr | x <- list1; (p x); ...] = ???`
- ▶ `[expr | x <- list1; y <- list2; ...] = ???`

Here f is some function, p is some predicate, and $expr$ is any expression.

If you can do the first three, you can be happy that you understand map and filter. If you can do the fourth one you're doing very well.

