# Lab 4

Functional Programming (ITI0212)

2022-02-18

This week we learned about function literals and higher-order functions. We can refer to the function with formal parameter `x` and body `t` using the (ASCIIfied) λ notation `\ x => t`. For example, the generic identity function can be written as `\ x => x`.

A *higher-order function* is a function that traffics in other functions, either by taking them as arguments or by returning them as results. We saw how the `map` and `filter` functions for `List` types allow us to perform tasks that would typically be done in imperative programming languages using loops, and how the `fold` function for an inductive type lets us capture its recursion principle as an ordinary function.

You can use the `filter` function for `List`s in the standard library by importing `Data.List` in your script file.

**Task 1**
Before consulting Idris, work out for yourself the types and values of the following two expressions.

```
(map S . filter even)[0, 1, 2, 3]
```

```
(filter even . map S)[0, 1, 2, 3]
```

Then check your understanding by asking Idris to evaluate them for you.

**Task 2**
Write the `map` function for `Maybe` types:

```
map_maybe  :  (a -> b) -> Maybe a -> Maybe b
```

so that

```
Lab4> map_maybe S Nothing
Nothing
Lab4> map_maybe S (Just 41)
Just 42
```

**Task 3**
Use a function literal (λ-expression) to complete the following function that returns the numbers in a list that are multiples of 10:

```
round_numbers  :  List Integer -> List Integer
round_numbers  =  filter ?p
```

For example:

```
Lab4> round_numbers [5,10,15,20]
[10, 20]
```

*Hint:* the functions `mod` and `(==)` will be helpful.

**Task 4**
Use the `fold` for `List` types to complete the following function that adds together all the numbers in a list:

```
sum_list  :  List Integer -> Integer
sum_list  =  fold_list ?c ?n
```

so that

```
Lab4> sum_list [1,2,3]
6
Lab4> sum_list []
0
```

**Task 5**
Write the `fold` function for the `Bool` type, `fold_bool`.

- First determine the type of this function using the algorithm described in the lecture.
- Then write the function definition using the algorithm for that.

Up to argument order, you should recognize this function as a construct present in nearly every programming language, what is it? Idris also supports the conventional syntax for this construct, try it out.

**Task 6**
Write the `fold` function for `Maybe` types, `fold_maybe`.

**Task 7**
Rewrite the `map` function for `Maybe` types from task 2 as a `fold`:

```
map_maybe'  :  (a -> b) -> Maybe a -> Maybe b
map_maybe' f  =  fold_maybe ?g1 ?g2
```