CSci 2041:

Advanced Programming Principles

University of Minnesota,
Prof. Van Wyk,
Spring 2018

Exercise #3:Write `drop_while`

```
drop_while:  'a list -> ('a -> bool) -> 'a list
```

Exercise #4:Function composition

Write a function to compose two functions.

Your function should be named `compose` and the type

```
compose :  ('b -> 'c) -> ('a -> 'b) -> 'a -> 'c.
```

Exercise #5:What is the type of `map`?

Recall our examples:

```
map inc [1;2;3;4]
```

or

```
map Char.code [ 'a'; '^'; '4' ]
```

Exercise #6:What is the OCaml implementation of `map`?

Recall our examples:

```
map inc [1;2;3;4]
```

or

```
map Char.code [ 'a'; '^'; '4' ]
```

Exercise #7:What is the type of `filter`?

Recall our examples:

```
filter even [1;2;3;4;5;6;7]
```

or

```
filter positive [1.2; 3.4; -5.6; -7.8; 9.0 ]
```

Exercise #8:What is OCaml implementation of `filter`?

Recall our examples:

```
filter even [1;2;3;4;5;6;7]
```

or

```
filter positive [1.2; 3.4; -5.6; -7.8; 9.0 ]
```

Exercise #9:

Write a function that returns its input `char list` after removing all upper case letters from it.

(Well, lets just consider `'A'`, `'B'`, `'C'`, and `'D'` to keep this simple.)

And do it without using an `if-then-else` expression. Use `match`.

And, of course, use `filter`.

Exercise #10:What is the type of `fold`?

Recall our example:

```
fold (+) 0 [1;2;3;4]
```

Exercise #11:What is OCaml implementation of `fold`?

Recall our example:

```
fold (+) 0 [1;2;3;4]
```

We can see this as

```
1 + (2 + (3 + (4 + 0)))
```

Exercise #12:What is OCaml implementation of `fold`?

Or when we see

```
fold (+) 0 [1;2;3;4]
```

as

```
((((0 + 1) + 2) + 3) + 4)
```

Exercise #13:Partitioning

Can we write a function that partitions a list into two sub-lists based on a predicate?

```
partition:  ('a -> bool) -> 'a list ->
            'a list * 'a list
```

```
partition even [1;2;3;4;5;6;7;8;9;10]
```

evaluates to

```
( [2; 4; 6; 8; 10], [1; 3; 5; 7; 9] )
```

Exercise #14:Grouping by 3

Can we write a function that groups a list of elements into a list of sub-lists of length 3 (or fewer, for the last sub-list).

```
group_by_3 [1;2;3;4;5;6;7;8;9;10]
```

evaluates to

```
[[1; 2; 3]; [4; 5; 6]; [7; 8; 9]; [10]]
```