Ocaml List Functions

(https://caml.inria.fr/pub/docs/manual-ocaml/libref/List.html)

```
val List.length : 'a list -> int
```
Return the length (number of elements) of the given list.

```
val List.nth : 'a list -> int -> 'a
```
Return the `n`-th element of the given list. The first element (head of the list) is at position 0.

```
val List.hd : 'a list -> 'a
```
Return the first element of the given list. Raise `Failure "hd"` if the list is empty.

```
val List.tl : 'a list -> 'a list
```
Return the given list without its first element. Raise `Failure "tl"` if the list is empty.

```
val List.rev : 'a list -> 'a list
```
List reversal.

```
val List.append : 'a list -> 'a list -> 'a list
```
Concatenate two lists. Same as the infix operator `@`. Not tail-recursive (length of the first argument).

```
val List.filter : ('a -> bool) -> 'a list -> 'a list
```
`filter f l` returns all the elements of the list `l` that satisfy the predicate `f`. The order of the elements in the input list is preserved.

```
val List.map : ('a -> 'b) -> 'a list -> 'b list
```
`List.map f [a1; ...; an]` applies function `f` to `a1, ..., an`, and builds the list `[f a1; ...; f an]` with the results returned by `f`. Not tail-recursive.

```
val List.iter : ('a -> unit) -> 'a list -> unit
```
`iter f [a1; ...; an]` applies function `f` in turn to `[a1; ...; an]`. It is equivalent to `f a1; f a2; ...; f an`.

```
val List.fold_left : ('a -> 'b -> 'a) -> 'a -> 'b list -> 'a
```
`List.fold_left f a [b1; ...; bn]` is `f (... (f (f a b1) b2) ...) bn`.

```
val List.fold_right : ('a -> 'b -> 'b) -> 'a list -> 'b -> 'b
```
`List.fold_right f [a1; ...; an] b` is `f a1 (f a2 (... (f an b) ...))`. Not tail-recursive.