Functions as Values

When making larger programs, it is necessary to reuse other programs. There are many ways of doing this.

It is possible to pass a function to another function as an argument. For example, the function `map` below applies function f to every element in list l.

```
map : ('a -> 'b) -> 'a list -> 'b list

let rec map f l =
  match l with
    [] -> []
  | h::t -> f h :: map f t
```

It is also possible to use a function within another function without assigning a name in the namespace to the argument function. We call this type of function an *anonymous function*. The syntax is `fun name -> `*`expression`*. This can be used when a function is only applied in one place and is short. For example, below we define a function that returns true for an element if it's even.

```
evens : int list -> bool list

let evens l =
  map (fun x -> x mod 2 = 0) l
```

OCaml allows you to convert an operator into a function with the syntax `( `*`operator`*` )`. For example, `( <= ) 3 4` has the value `true` and `( + ) 3 4` has the value `7`.