Introduction to
Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 4 - Sequence 2: Functions With Several Arguments

Expressions for Multi-Argument Functions

- An anonymous function with several arguments is written

$$\texttt{fun } \texttt{p1 ... pn -> exp}$$

  where the `pi` are patterns
- Unlike `function`, the `fun` form *only admits one case (or branch)*, for example,
  `fun (x,y) 1 -> x | (x,y) 2 -> y` is not accepted

Functions Returning Functions

- ▶ Functions are *First-Class Values*
- ▶ The return value of a function may be a function:

$$\texttt{function n -> (function x -> x+n)}$$

- ▶ The type of this function is something we have seen earlier!

Functions Returning Functions I

```
let f1 = function n -> (function x -> n+x);;
# val f1 : int -> int -> int = <fun>

(f1 17) 73;;
# - : int = 90

f1 17 73;;
# - : int = 90

let f2 = fun n x -> n+x;;
# val f2 : int -> int -> int = <fun>

f2 17 73;;
# - : int = 90
```

Functions Returning Functions II

```
(f2 17) 73;;
# - : int = 90
```

The Truth About Functions With Multiple Arguments

▶ Functions with multiple arguments are the same thing as functions returning functions as values!

▶ More precisely:
    `fun` x1 ... xn -> e
is just an abbreviation for
    `function` x1 -> ... -> `function` xn -> e

Four equivalent function definitions I

```
type expr =
  | Var of string
  | Add of expr * expr;;
# type expr = Var of string | Add of expr * expr
```

```
let rec eval = fun environment expr -> match expr with
  | Var x -> List.assoc x environment
  | Add(e1,e2) -> (eval environment e1)
                    + (eval environment e2);;
# val eval : (string * int) list -> expr -> int = <fun>
```

```
eval [("x",2); ("y",5)]
  (Add (Var "x", Add (Var "x", Var "y")));;
# - : int = 9
```

Four equivalent function definitions II

```
let rec eval =
  function environment ->
    function expr -> match expr with
      | Var x -> List.assoc x environment
      | Add(e1,e2) -> (eval environment e1)
                        + (eval environment e2);;
# val eval : (string * int) list -> expr -> int = <fun>

eval [("x",2); ("y",5)]
  (Add (Var "x", Add (Var "x", Var "y")));;
# - : int = 9
```

Four equivalent function definitions III

```
let rec eval = function environment -> function
  | Var x -> List.assoc x environment
  | Add(e1,e2) -> (eval environment e1)
                   + (eval environment e2);;
# val eval : (string * int) list -> expr -> int = <fun>

eval [("x",2); ("y",5)]
  (Add (Var "x", Add (Var "x", Var "y")));;
# - : int = 9
```

Four equivalent function definitions IV

```
let rec eval environment = function
  | Var x -> List.assoc x environment
  | Add(e1,e2) -> (eval environment e1)
                  + (eval environment e2);;
# val eval : (string * int) list -> expr -> int = <fun>

eval [("x",2); ("y",5)]
  (Add (Var "x", Add (Var "x", Var "y")));;
# - : int = 9
```