

# Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 4 - Sequence 0: Functional Expressions



# Overview of Week 4

0. Functional Expressions
1. Functions as First-Class Values
2. Functions with Multiple Arguments
3. Partial Function Application
4. Mapping Functions on Lists
5. Folding Functions on Lists

# Functional Expressions in OCaml

- ▶ Syntax: `function id -> exp`
- ▶ Function taking one argument `id`, and returning the value of expression `exp`.
- ▶ Example: `function x -> x+1`
- ▶ Scope of `id` restricted to `exp`
- ▶ Type:  $t_1 \rightarrow t_2$  where
  - ▶  $t_1$  is the type of `id`
  - ▶  $t_2$  is the type of `exp`

# Functional Expressions I

```
function x -> x+1;;  
# - : int -> int = <fun>
```

```
function y -> [ [y+2; y+3]; [y; y*y] ];;  
# - : int -> int list list = <fun>
```

```
(function x -> 2*x) 5;;  
# - : int = 10
```

# Defining Functions

- ▶ The previous way of defining functions

```
let f x = e
```

is just an abbreviation for

```
let f = function x -> e
```

- ▶ One uniform way of defining identifiers: `let`

# Defining Functions I

```
let double x = 2*x;;  
# val double : int -> int = <fun>
```

```
double 3;;  
# - : int = 6
```

```
let double = (function x -> 2*x);;  
# val double : int -> int = <fun>
```

```
double 3;;  
# - : int = 6
```

# Functions With Pattern Matching

- ▶ The general form of a function definition is:

```
function
```

```
| pattern_1 -> expression_1
```

```
....
```

```
| pattern_n -> expression_n
```

- ▶ The form `function x -> exp` is just a special case.

# Functional Expressions with Pattern Matching I

```
let rec length = function
  | [] -> 0
  | _::r -> 1+ length r
;;
# val length : 'a list -> int = <fun>
```

```
length [17; 42; 73];;
# - : int = 3
```



# Functional Expressions with Pattern Matching II

```
type expr =  
  | Int of int  
  | Add of expr * expr  
  
let rec eval = function  
  | Int n -> n  
  | Add(e1,e2) -> (eval e1) + (eval e2);;  
# type expr = Int of int | Add of expr * expr  
val eval : expr -> int = <fun>  
  
eval (Add (Add (Int 2, Int 5), Int 7));;  
# - : int = 14
```