

Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 4 - Sequence 1: Functions As First-Class Values



Functions Are Values

- ▶ Expressions may denote integers, boolean, ..., or *functions* :
`function x -> ...`
- ▶ In functional languages, functions are just values of a particular type
- ▶ Uniform way of naming a value : `let y = ...`
- ▶ Types govern function application: We can apply e_1 to e_2 when
 - ▶ e_1 has a type $t_1 \rightarrow t_2$
 - ▶ t_1 matches the type of e_2

First Class

- ▶ This doesn't stop here: Functions may, as any other values
 - ▶ be part of a structured data value, like a list,
 - ▶ be actual arguments of functions,
 - ▶ be the result value of a function application.
- ▶ We may say : *Functions are First-Class Values.*

Data Structures Containing Functions I

```
let fl = [(function x -> x+1);(function x -> 2*x)];;  
# val fl : (int -> int) list = [<fun>; <fun>]
```

```
(List.hd fl) 17;;  
# - : int = 18
```

Functions Taking Functions as Argument I

```
let apply_twice f x = f (f x);;  
# val apply_twice : ('a -> 'a) -> 'a -> 'a = <fun>
```

```
apply_twice (function x -> 2*x) 1;;  
# - : int = 4
```

```
let rec apply_n_times f n x =  
  if n <= 0  
  then x  
  else apply_n_times f (n-1) (f x);;  
# val apply_n_times : ('a -> 'a) -> int -> 'a -> 'a = <fun>
```

```
apply_n_times (function x -> 2*x) 10 1;;  
# - : int = 1024
```

Functions Returning Functions as Result I

```
let compose f g = (function x -> f(g x));;  
# val compose : ('a -> 'b) -> ('c -> 'a) -> 'c -> 'b = <fun>
```

```
compose (function x->x+1) (function x->2*x);;  
# - : int -> int = <fun>
```

```
(compose (function x->x+1) (function x->2*x)) 10;;  
# - : int = 21
```

Functions Returning Functions as Result II

```
compose (function x-> x+1) (function x -> x *. 3.14);;
```

Characters 43-52:

```
compose (function x-> x+1) (function x -> x *. 3.14);;  
                                ~~~~~
```

Error: This expression has **type** float but an expression was expected
of type
int

Function Pitfalls

- Functions apply in order from left to right:

`exp1 exp2 exp3`

is equivalent to

`(exp1 exp2) exp3`

- We say: *function application associates to the left*

Order of Function Application I

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

```
double double 5;;
```

```
# Characters 1-7:
```

```
double double 5;;  
^^^^^^
```

Error: This **function** has **type** int -> int

It is applied **to** too many arguments; maybe you forgot a **';**'.

```
double (double 5);;
```

```
# - : int = 20
```