

Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 1 - Sequence 4: Functions



Defining Functions

- ▶ Global definition of a function with one argument :
 $\text{let } f\ x = \text{exp}$
- ▶ Local definition of a function with one argument :
 $\text{let } f\ x = \text{exp1 in exp2}$
- ▶ Scoping rules as before (sequence 3) : local definitions hide more global ones
- ▶ Application of function named f to expression e : $f\ e$
- ▶ Parenthesis indicate structure of expressions

Function Definition and Application I

```
let f x = x+1;; (* global definition *)
```

```
# val f : int -> int = <fun>
```

```
f 17;;
```

```
# - : int = 18
```

```
let g y = 2*y    (* local definition *)
```

```
in g 42;;
```

```
# - : int = 84
```

```
f f 1;;
```

```
# Characters 1-2:
```

```
  f f 1;;
```

```
  ^
```

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

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

Function Definition and Application II

```
(f f) 1;;
```

```
# Characters 4-5:
```

```
(f f) 1;;  
  ^
```

```
Error: This expression has type int -> int  
      but an expression was expected of type int
```

```
f (f 1);;
```

```
# - : int = 3
```

Lexical Scoping

Lexical Scoping: identifier used in the definition of a function refers to the identifier visible at the moment of function *definition*

Dynamic Scoping: ... visible at the moment of function *invocation*

Lexical Scoping I

```
(* with local definitions *)
```

```
let f x = x+1 in
```

```
let g y = f (f y) in
```

```
let f x = 2*x in
```

```
g 5;;
```

```
# Characters 71-72:
```

```
  let f x = 2*x in
```

```
    ^
```

```
Warning 26: unused variable f.
```

```
- : int = 7
```

```
(* with global definitions *)
```

```
let f x = x+1;;
```

```
# val f : int -> int = <fun>
```

Lexical Scoping II

```
let g y = f (f y);;  
# val g : int -> int = <fun>  
let f x = 2*x;;  
# val f : int -> int = <fun>  
g 5;;  
# - : int = 7
```

Identifiers are not Variables

- ▶ An identifier may be hidden by a new definition for the same name
- ▶ Do not confuse with “changing the value of a variable”
- ▶ Static binding can give you indirect access to an otherwise hidden identifier

Redefinition is not Assignment I

```
let a = 1;;  
# val a : int = 1
```

```
let f x = x + a;;  
# val f : int -> int = <fun>
```

```
f 2;;  
# - : int = 3
```

```
let a = 73;;  
# val a : int = 73
```

```
f 2;;  
# - : int = 3
```