

Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 1 - Sequence 2: Expressions



Expressions

- ▶ expressions compute values
- ▶ expressions play a prime role in functional programming
- ▶ very rich language of expressions

Conditional Expressions

- ▶ `if ... then ... else ...`
- ▶ is an *expression*, not an instruction!
- ▶ type is the type of the expressions in `then` and `else`, which must be the same
- ▶ default value in case of missing `else` : not what you might expect! (see Week 5)

Conditional Examples I

```
if 1<2 then 6+7 else 67/23;;
```

```
# - : int = 13
```

```
if 6=8 then 1 else 77.5;;
```

```
# Characters 20-24:
```

```
if 6=8 then 1 else 77.5;;  
~~~~~
```

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

```
(if 6=3+3 then 3<4 else 8 > 7) && 67.8 > 33.1;;
```

```
# - : bool = true
```

```
if (if 1=1 then 2=2 else 4.0 > 3.2) then 2<3 else 3<2;;
```

```
# - : bool = true
```

Function Application

- ▶ The type of a function with n arguments is like this:

$type\text{-}argument_1 \rightarrow \dots \rightarrow type\text{-}argument_n \rightarrow type\text{-}result$

- ▶ To apply function f to n arguments:

$f\ expression_1 \dots expression_n$

- ▶ Example:

Type: `String.get` : `string` \rightarrow `int` \rightarrow `char`

Application: `String.get "abcd" 2`

- ▶ Use parentheses to indicate structure

Function Application Examples I

```
String.get "abcd" 2;;
```

```
# - : char = 'c'
```

```
String.get ("Hello,_" ^ "World") (5-2);;
```

```
# - : char = 'l'
```

```
String.get (string_of_int 65) (int_of_string "0");;
```

```
# - : char = '6'
```

Expression Pitfalls

- ▶ local definitions can be used to cut large expressions into pieces
(see next sequence)
- ▶ functions may be under-supplied with arguments
(see Week 4)
- ▶ $f(e_1, e_2)$ is *not* an application of f to two arguments
(see Week 2 for an explanation)

Polymorphic Operators

- ▶ Operators have an infix syntax, like $(3 + 5) * 5$
- ▶ Operators, like functions, always have a type : $+$: $\text{int} \rightarrow \text{int} \rightarrow \text{int}$
- ▶ Some have a *polymorphic type*: $>$: $'a \rightarrow 'a \rightarrow \text{bool}$
- ▶ Polymorphic types contain *type variables*, indicated by an initial quote.
- ▶ $'a$ reads *alpha*, $'b$ reads *beta*, etc.
- ▶ Type variables can be instantiated by any type

Applying a function with polymorphic type I

```
12 > 56.1;;
```

```
# Characters 5-9:
```

```
12 > 56.1;;
```

```
^^^^
```

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

```
(73>42) && (1e10>0.1) && ('B'>'A');;
```

```
# - : bool = true
```

Expression Pitfalls

- ▶ The operator for checking equality of values is =
- ▶ An operator == exists but does something else (see Week 2)

To Know More

The OCaml Manual:

- ▶ The OCaml language
 - ▶ Expressions