Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 1 - Sequence 1: More Data Types

Floating-point arithmetic

- Type: `float`
- Values: Must be written with a decimal point (5.2), or exponential (5e3, 6e-9), or both.
- Operations: $+.$, $-.$, $*.$, $/.$
- Functions: `sqrt`, `sin`, `cos`, `ceil`, `floor`, ...

Floating-point Pitfalls

- Floating-point constants must indicate that they are not integers: use decimal dot or exponent
- Floating-point operations must be written with a dot (`+.` etc.)

Floating-point Examples I

```
3.0 +. 0.01;;
# - : float = 3.01

2e-4 *. 0.1;;
# - : float = 2e-05

2 + 3.0;;
# Characters 5-8:
  2 + 3.0;;
      ^^^
Error: This expression has type float but an expression was expected
    of type
           int
```

Floating-point Examples II

```
4 *. 56;;
# Characters 1-2:
  4 *. 56;;
  ^
Error: This expression has type int but an expression was expected
    of type
           float
```

Conversions between types

- Basic types are *disjoint*: no value belongs to two different basic types
- No implicit conversion between types
- Explicit conversion operations
- Background: implicit conversion would not go well with type inference

Conversion Floating-point $\leftrightarrow$ Integer

- Conversion functions in both directions :
  `float_of_int` : `int` $\rightarrow$ `float`
  `int_of_float` : `float` $\rightarrow$ `int`
- Function application:
  write the name of the function, followed by the argument
- Parentheses only when necessary to indicate structure
- More on functions later

Characters

- Type: `char`
- Values: 256 characters, numbered from 0 to 255
- can be written like `'a'`, `'\087'`, etc (see the manual)
- Conversion functions :
  `Char.chr : int → char`
  `Char.code : char → int`
- More on modules later

Character Examples I

```
Char.code 'A';;
# - : int = 65

Char.code '\122';;
# - : int = 122

Char.chr 65;;
# - : char = 'A'

Char.chr (Char.code 'B');;
# - : char = 'B'
```

Strings

- Type: `string`
- Values: Character strings, written like `"Hello, World!"`
- Operator `^` for string concatenation
- Many functions, like

```
String.length :  string → int
int_of_string :  string → int
float_of_string :  string → float
```

Character and String Pitfalls

- Strings, Characters, Integers, Booleans, Floats are all disjoint
- You must use explicit conversion functions
- Positions in strings are numbered from 0 to its length minus 1
- In earlier versions of *OCaml*, strings used to be modifiable

String Examples I

```
"abc" ^ "def";;
# - : string = "abcdef"

String.length "12345";;
# - : int = 5

int_of_string "12345";;
# - : int = 12345

string_of_int 12345;;
# - : string = "12345"

String.get "abcdef" 1;;
# - : char = 'b'
```

To Know More

The OCaml Manual:

- The core library
  - Module `Pervasives` :
    - Floating Point Arithmetic
- The standard library
  - Module `Char`
  - Module `String`