Introduction to Functional Programming in *OCaml*

Roberto Di Cosmo, Yann Régis-Gianas, Ralf Treinen

Week 5 - Sequence 2: Getting information in and out

Back to the toplevel

We have been using *OCaml*'s toplevel extensively up to now.

- it *reads* our program, *incrementally*
- it *prints* the result of the execution

  ... we could do without input/output operations!

For real programs, *OCaml* offers a rich set of I/O primitives.
We will now look at some of them.

But let's first meet the `unit` type.

The `unit` type

```
# ();;
- : unit = ()
```

The `unit type`

- the typical input or result type of a function with side effects
- has only one value
- also called *unit*
- written `()`
- why this syntax? will be clear in a few slides

Simple output

Printing an integer

```
# print_int;;
- : int -> unit = <fun>
```

This function

- takes an `integer`
- prints the integer on *standard output*
- returns the value `()` of the `unit` type

Simple output

Printing an integer

```
# print_int 12345;;
12345- : unit = ()
```

What happens

- `12345` is printed on *standard output*
- the *toplevel* prints its message, which says
  - the evaluation returns the value `()`
  - of the `unit` type
  - there is no identifier bound to it `-` :

Simple input

Reading a line

```
# read_line;;
```

*- : unit -> string = <**fun**>*

This function

- takes as input the value `()` of the `unit` type
- reads a line of characters from *standard input* as a string

Simple input

Reading a line

```
# read_line();;
```

```
some text
- : string = "some text"
```

What happens

- `read_line` receives the argument `()`
- it starts reading from *standard input*
- we type some text and hit return
- the *toplevel* prints its message, which says
  - the evaluation returns the value `"some text"`
  - of the `string` type
  - there is no identifier bound to it `- :`

About the syntax

See why `()` for the unique value of the `unit` type?

```
read_line()
```

This looks like *a function with no argument* in other languages.
It's more familiar for outsiders!

Remember, it really is:

```
read_line ()
```

Simple input and output

Printing other base types

```
print_char : char -> unit
print_string : string -> unit
print_float : float -> unit
```

Flushing and newline

```
val print_newline : unit -> unit
```

Print a newline and flush standard output.

Simple input and output

There is much more

- *standard input*, *standard output* and *standard error*
- create, open and close *files*
- read and write on *channels*
- sophisticated parsing, like `scanf`, *well typed*!
- *see the manual section on* `Pervasives`

Notice: some of these functions are not implemented
in the toplevel running in your browser.

Summary

Unit type, Input and Output

- The `unit` type is often used with functions with side effects, like `print_int : int -> unit`
- `read_line()` is really `read_line` applied to `()`
- We now know how to perform basic input/ouput
- *OCaml* has many more sophisticated input/output functions, look at the reference manual to know more