Programming Languages and Environments (Lecture 2)

LEI - Licenciatura em Engenharia Informática

João Costa Seco (joao.seco@fct.unl.pt)

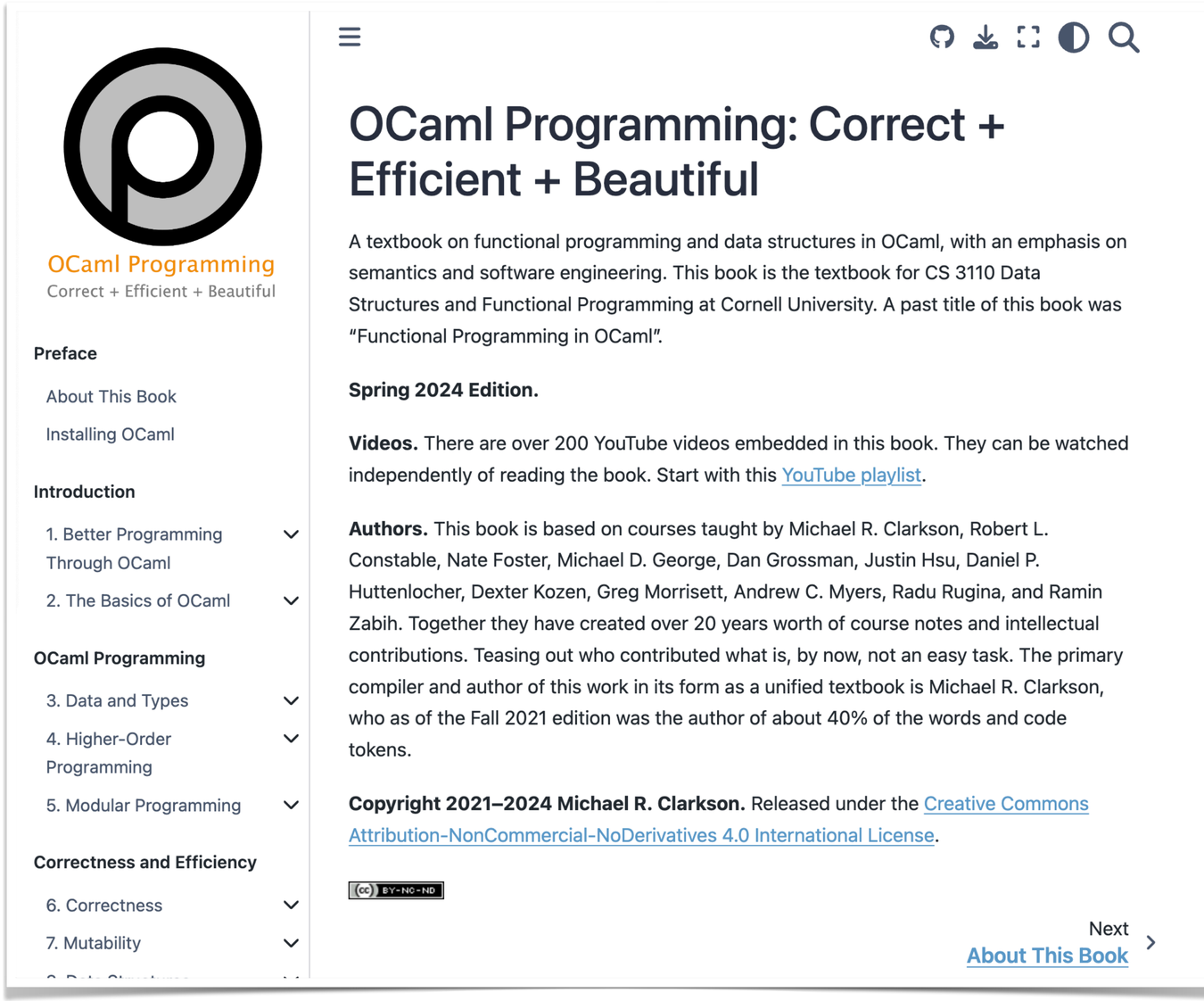
NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Syllabus

- Introduction to Functional Programming.
- The OCaml Programming Language.
- Expressions, variables and types.

Course textbook

- OCaml Programming:
  Correct + Efficient + Beautiful
- Cornell University
- Michael R. Clarkson et al.
- Online resources
  (book, exercises, videos)



Main Characteristics of Functional Programming

- Declarative Programming
  (thinking about "what", rather than "how")
- Typical features:
  - All constructs are **expressions**
  - Functions as first-class values
  - Composability
  - Immutable values
  - Strong type systems
- Features used in other languages.

“A language that doesn’t affect the way you think about programming is not worth knowing.”

—Alan J. Perlis (1922-1990), first recipient of the Turing Award

OCaml Programming

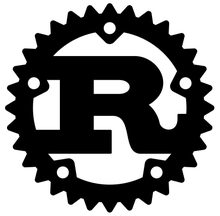
Correct + Efficient + Beautiful

Programming Languages Gradient

- Pure Functional PLs (Haskell)

- Functional PLs with Imperative constructs (OCaml)

- Imperative PLs with functional mechanisms (Rust)



- Imperative PLs (procedural or object-oriented) (Java 17 and older)

Functional Programming Languages (logo game)

ML (OCaml) is logic in programming

$$\frac{A \longrightarrow B \qquad A}{B}$$

ML (OCaml) is logic in programming

$$\frac{f : A \longrightarrow B \quad x : A}{f\ x : B}$$

Example in Java

```
B f1(A a) {...}

C f2(B b) {...}

D f3(C c) {...}

D d = f3(f2(f1(new A())))
```

$$f1 : A \longrightarrow B$$
$$f2 : B \longrightarrow C$$
$$f3 : C \longrightarrow D$$

$$\dfrac{\dfrac{\dfrac{A \quad A \longrightarrow B}{B} \quad B \longrightarrow C}{C} \quad C \longrightarrow D}{D}$$

OCaml

- Dialect of the ML family.
- A language where expressions:
  - They are either values,
  - They either terminate with the raise of exceptions,
  - They do not terminate...
- Strongly typed with type inference

Xavier Leroy

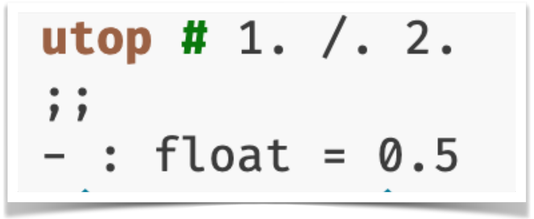
Didier Rémy

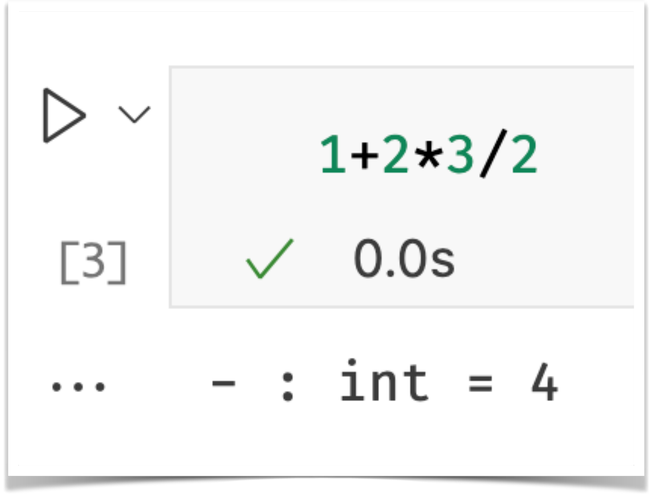
Damien Doligez

Jérôme Vouillon

Hands-on OCaml

- Interpreter: `ocaml` / `utop`
- Compiler: `ocamlc` + `make` / `dune`
- Visual Studio Code + OCaml plugin
- Jupyter Notebook + OCaml kernel





Basic types

Type	Literal	Operator	Function
int	1 2 3 0b101 0x42	+ - * /	
float	1. 2. 3.5 4e10	+. -. *. /.	float_of_int
bool	true false	&& ||	
char	'a' 'b'		char_of_int, int_of_char
strings	"hello" ""	^	string_of_int
unit	()		

```
3.14 * 2.
```

[4] 0.0s

```
File "[4]", line 1, characters 0-4:
1 | 3.14 * 2.
    ^^^^
Error: This expression has type float but an expression was expected of type
         int
```

Basic types

Type	Literal	Operator	Function
int	1 2 3 0b101 0x42	+ - * /	
float	1. 2. 3.5 4e10	+. -. *. /.	float_of_int
bool	true false	&& ||	
char	'a' 'b'		char_of_int, int_of_char
strings	"hello" ""	^	string_of_int
unit	()		

```
3.14 *. (float_of_int 2)
```

[5] ✓ 0.0s

```
- : float = 6.28
```

Basic types

Type	Literal	Operator	Function
int	1 2 3 0b101 0x42	+ - * /	
float	1. 2. 3.5 4e10	+. -. *. /.	float_of_int
bool	true false	&& ||	
char	'a' 'b'		char_of_int, int_of_char
strings	"hello" ""	^	string_of_int
unit	()		

```
int_of_string "not an int"
```

[6] ⊗ 0.9s

```
Exception: Failure "int_of_string".
Raised by primitive operation at unknown location
Called from Stdlib__fun.protect in file "fun.ml", line 33, characters 8-15
Re-raised at Stdlib__fun.protect in file "fun.ml", line 38, characters 6-52
Called from Toploop.load_lambda in file "toplevel/toploop.ml", line 212, characters 4-150
```

Basic types

Type	Literal	Operator	Function
int	1 2 3 0b101 0x42	+ - * /	
float	1. 2. 3.5 4e10	+. -. *. /.	float_of_int
bool	true false	&& ||	
char	'a' 'b'		char_of_int, int_of_char
strings	"hello" ""	^	string_of_int
unit	()		

```
"abc".[0]
```

[7] ✓ 0.0s

```
- : char = 'a'
```

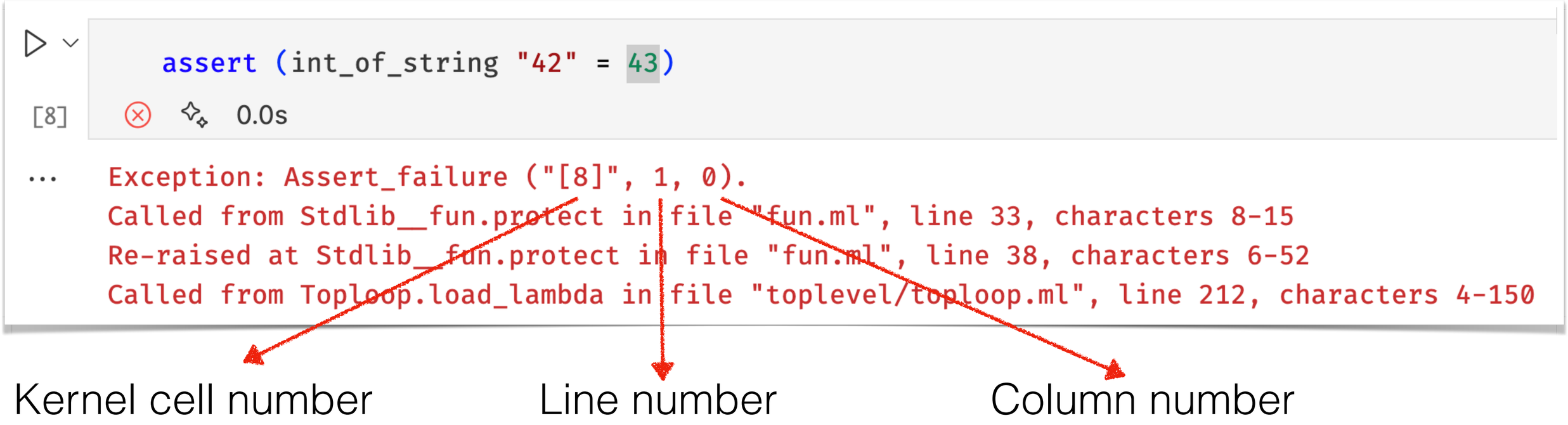
To be or To BE

- There are innumerate ways of defining equality in OCaml, we can redefine its meaning.
- Though, the language defines two essential ways of testing equality.
- It can be **structural** equality, where two values have the same structure and contents.
- Or it can be **physical**, where two values point to the same object in memory.

Comparison	Structural	Physical
Equality	=	==
Inequality	<>	!=
Compares	Contents	Memory locations
Example	`[1;2;3] = [1;2;3]`	`[1;2;3] == [1;2;3]`

Assertions

- Assertions are an effective way of testing functionality.



Conditional Expressions

- Conditional expressions are expressions where both branches have the same type.

```
4 + (if 'a' = 'b' then 1 else 2)
```

```
[9]  ✓ 0.0s
```

```
- : int = 6
```

- Note: the **`else`** branch is mandatory for all types except for **unit**.

```
if "hello" = "world" then 1
```

```
[10]  ⊗ 0.0s
```

```
File "[10]", line 1, characters 26-27:
1 | if "hello" = "world" then 1
                              ^

Error: This expression has type int but an expression was expected of type
         unit
       because it is in the result of a conditional with no else branch
```