

Programming Languages and Environments (Lecture 2)

LEI - Licenciatura em Engenharia Informática

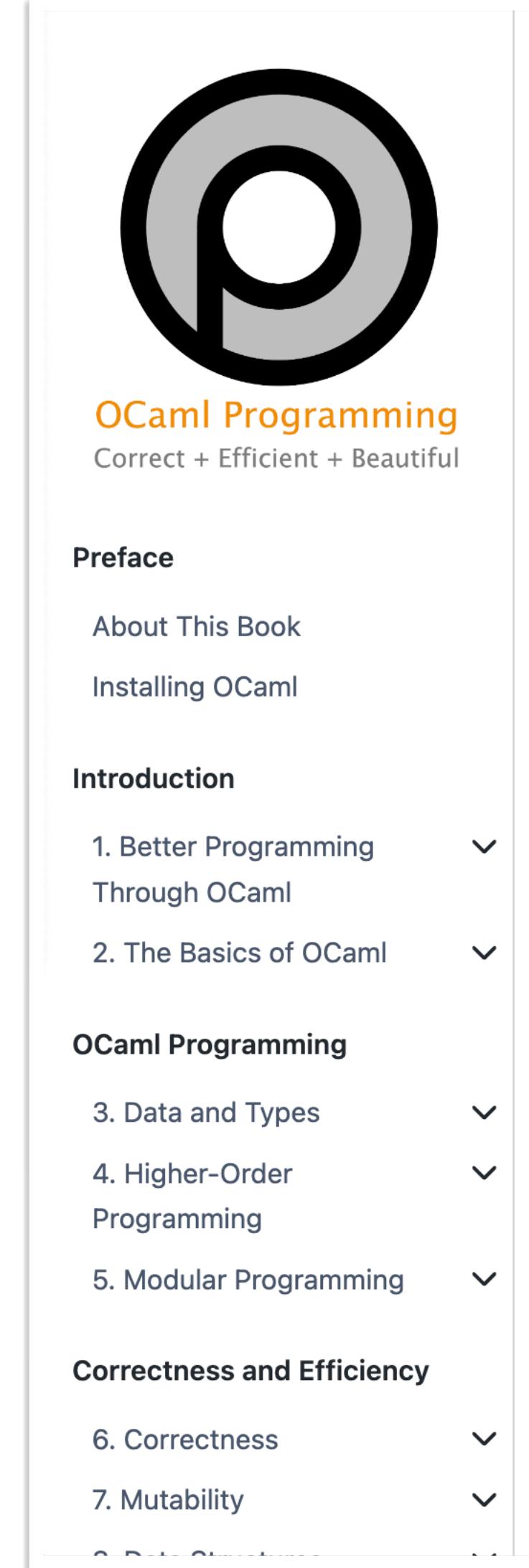
João Costa Seco (joao.seco@fct.unl.pt)

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)



The screenshot shows the front page of the OCaml Programming book. At the top right are GitHub, download, and search icons. The title "OCaml Programming: Correct + Efficient + Beautiful" is centered. Below it is a large circular logo with a stylized 'p'. The subtitle "OCaml Programming" and the tagline "Correct + Efficient + Beautiful" are displayed. A sidebar on the left contains links for "Preface", "About This Book", and "Installing OCaml". The main content area starts with "Introduction" and a list of chapters: 1. Better Programming Through OCaml, 2. The Basics of OCaml, 3. Data and Types, 4. Higher-Order Programming, 5. Modular Programming, and 6. Correctness. Below this is a section titled "Correctness and Efficiency" with chapters 7. Mutability and 8. Concurrency. To the right of the introduction, there is a block of text about the Spring 2024 edition, YouTube videos, authors, and copyright information. A Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License logo is at the bottom right.

≡

OCaml Programming: Correct + Efficient + Beautiful

A textbook on functional programming and data structures in OCaml, with an emphasis on semantics and software engineering. This book is the textbook for CS 3110 Data Structures and Functional Programming at Cornell University. A past title of this book was "Functional Programming in OCaml".

Spring 2024 Edition.

Videos. There are over 200 YouTube videos embedded in this book. They can be watched independently of reading the book. Start with this [YouTube playlist](#).

Authors. This book is based on courses taught by Michael R. Clarkson, Robert L. Constable, Nate Foster, Michael D. George, Dan Grossman, Justin Hsu, Daniel P. Huttenlocher, Dexter Kozen, Greg Morrisett, Andrew C. Myers, Radu Rugina, and Ramin Zabih. Together they have created over 20 years worth of course notes and intellectual contributions. Teasing out who contributed what is, by now, not an easy task. The primary compiler and author of this work in its form as a unified textbook is Michael R. Clarkson, who as of the Fall 2021 edition was the author of about 40% of the words and code tokens.

Copyright 2021–2024 Michael R. Clarkson. Released under the [Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License](#).

Next >

[About This Book](#)

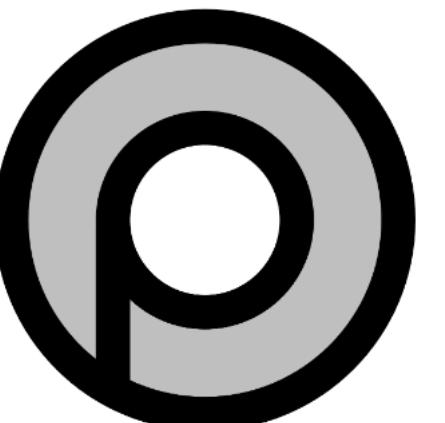
Main Characteristics of Functional Programming

- 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.



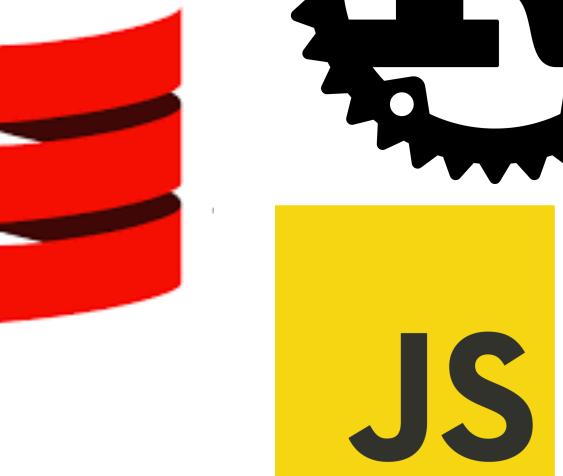
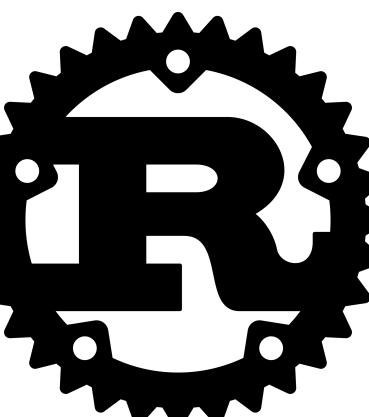
**“A language that doesn’t affect the way you think about
programming is not worth knowing.”**

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



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

$A \rightarrow B$

B

ML (OCaml) is logic in programming

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

Example in Java

B f1(A a) {...}

$$f1 : A \rightarrow B$$

C f2(B b) {...}

$$f2 : B \rightarrow C$$

D f3(C c) {...}

$$f3 : C \rightarrow D$$

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

$$\frac{\begin{array}{c} A \quad A \rightarrow B \\ \hline B \end{array}}{\begin{array}{c} B \quad B \rightarrow C \\ \hline C \end{array}} \quad \begin{array}{c} C \rightarrow D \\ \hline \end{array}$$

$$D$$



- 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

facebook

 Microsoft

 docker

 Jane
Street

Bloomberg

τ

 ahrefs

Hands-on OCaml

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



```
utop # 1. /. 2.  
;;  
- : float = 0.5
```

```
▶ ▾ [3] ✓ 0.0s  
1+2*3/2  
... - : int = 4
```

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 |
| ... | ... | | |

```
▷ "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.

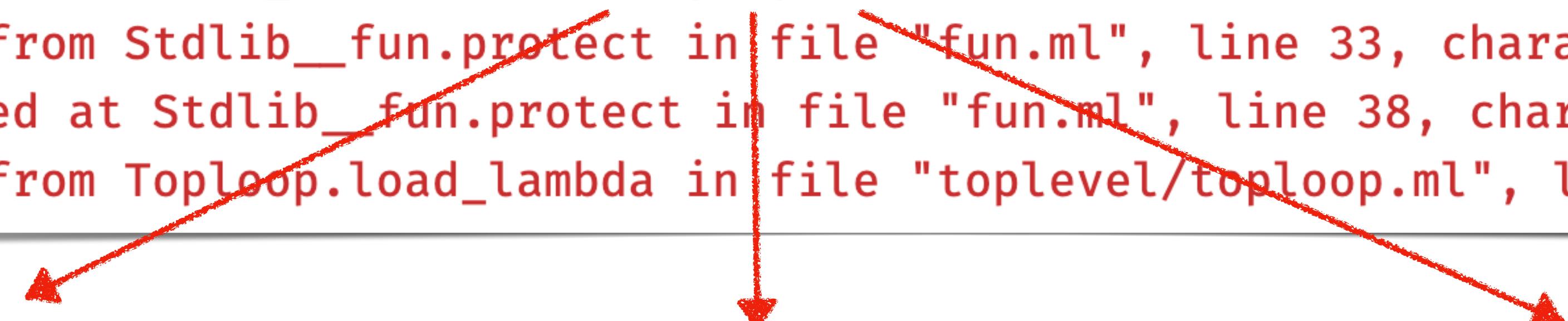
| Comparison | Structural | Physical |
|-------------------|-------------------|--------------------|
| Equality | = | == |
| Inequality | <> | != |
| Compares | Contents | Memory locations |
| Example | [1;2;3] = [1;2;3] | [1;2;3] == [1;2;3] |

Assertions

- Assertions are an effective way of testing functionality.

```
▶ 
  assert (int_of_string "42" = 43)
[8] ✘ 0.0s
...
Exception: Assert_failure ("[8]", 1, 0).
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
```

Kernel cell number Line number Column number



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
```