

Linguagens e Ambientes de Programação

(Aula Teórica 2)

LEI - Licenciatura em Engenharia Informática

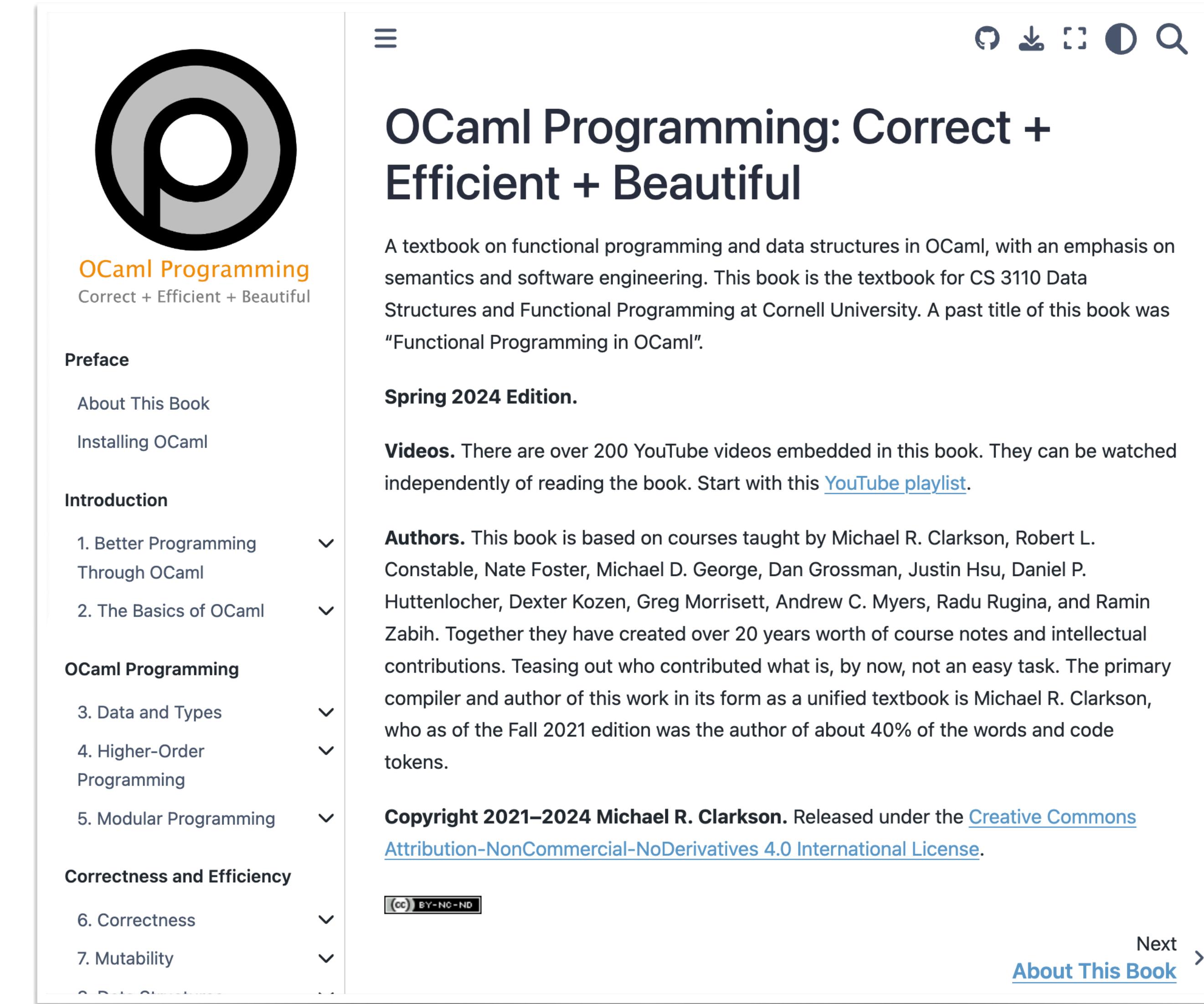
João Costa Seco (joao.seco@fct.unl.pt)

Agenda para hoje

- Introdução à programação funcional.
- A linguagem OCaml.
- Expressões, variáveis e tipos.
- Funções biblioteca. Input/Output básico.

Livro de texto

- 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 "A textbook on functional programming and data structures in OCaml, with an emphasis on semantics and software engineering." is followed by a note about it being the textbook for CS 3110 Data Structures and Functional Programming at Cornell University. It also mentions a past title, "Functional Programming in OCaml". A "Spring 2024 Edition" notice is present. The main content area has a sidebar with links to "Preface", "About This Book", and "Installing OCaml". The main content includes sections like "Introduction" (with chapters 1 and 2), "OCaml Programming" (with chapters 3, 4, and 5), and "Correctness and Efficiency" (with chapters 6 and 7). A Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License logo is at the bottom left, and a "Next >" link and "About This Book" link are 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

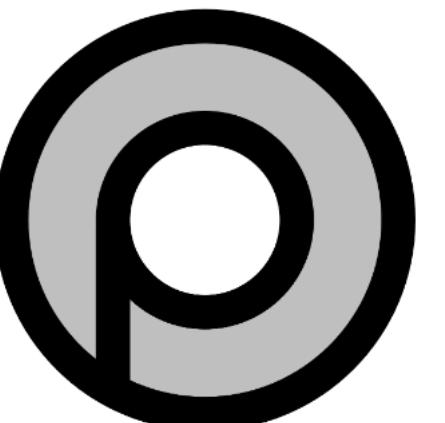
Principais Características da Programação Funcional

- É uma programação declarativa (especificar “o que é”, em vez de “como é”)
- As construções típicas são:
 - Todas as construções são expressões
 - Funções como valores da linguagem
 - Composicionalidade
 - Valores imutáveis
 - Sistema de tipos forte
 - Construções usadas noutras linguagens.



**“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



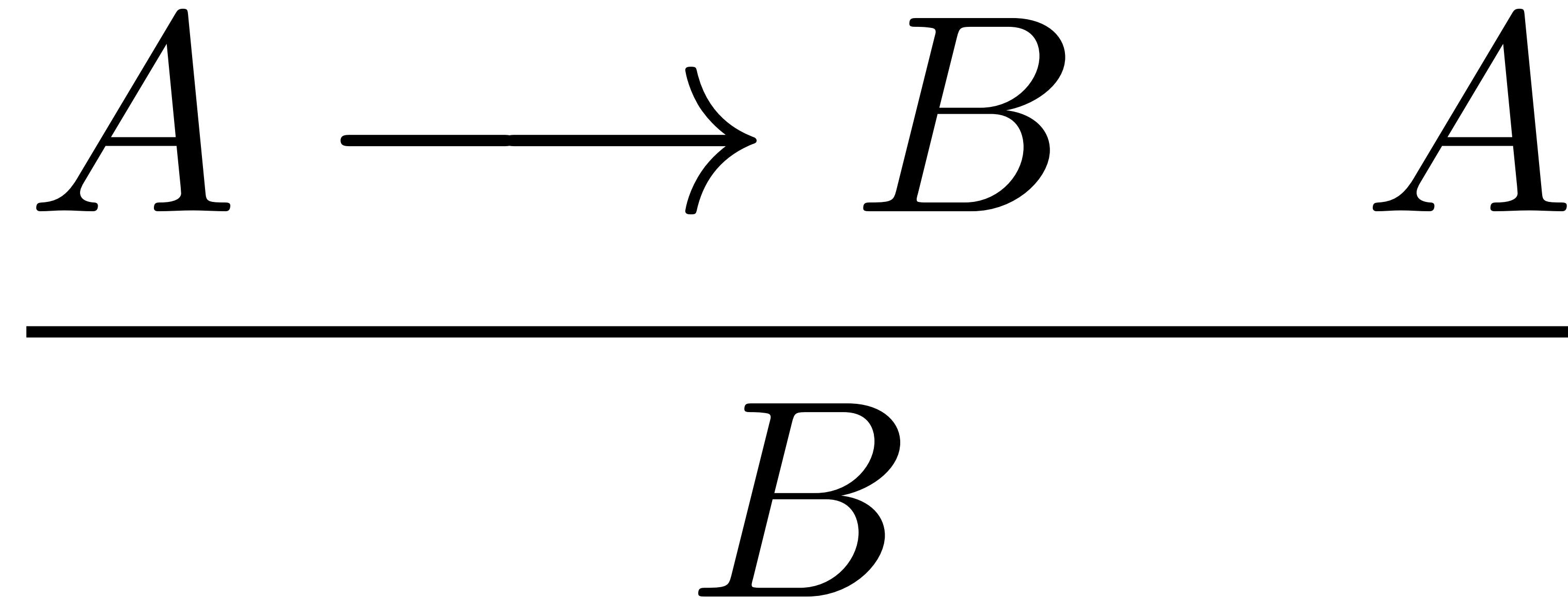
Gradiente do paradigma funcional em linguagens

- Linguagens funcionais puras - Haskell
- Linguagens funcionais com características imperativas (estado) - OCaml
- Linguagens imperativas com mecanismos funcionais - Rust
- Linguagens imperativas (procedimentais ou object oriented) - Java<17

Linguagens funcionais (logo game)



ML (OCaml) is logic in programming



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())))

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

$$D$$

OCaml: uma linguagem de expressões

- É um idioma da linguagem ML
- É uma linguagem de expressões que:
 - Ou denotam um valor,
 - Ou terminal com uma exceção
 - Ou não terminam...
- É fortemente tipificada com inferência de tipos



facebook

 Microsoft

 docker

 Jane
Street

Bloomberg

ts

ahrefs

Ambientes de programação

- Interpretador: ocaml / utop
- Compilador: ocamlc + make/dune
- Visual Studio Code + OCaml plugin
- Jupyter Notebook + OCaml kernel



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

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

Tipos básicos, literais, operadores e funções

int	1 2 3 4	+ - * /	
float	1. 2. 3.5 4e10	+. -. *.	float_of_int
bool	true false	&&	
char	'a' 'b'		char_of_int ,
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
```

Tipos básicos, literais, operadores e funções

int	1 2 3 4	+ - * /	
float	1. 2. 3.5 4e10	+. -. *.	float_of_int
bool	true false	&&	
char	'a' 'b'		char_of_int ,
strings	“hello” “”	^	string_of_int
unit	()		

A screenshot of a code editor showing a computation result. The code is:

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

The output is:

[5] ✓ 0.0s

... - : float = 6.28

Tipos básicos, literais, operadores e funções

int	1 2 3 4	+ - * /	
float	1. 2. 3.5 4e10	+. -. *.	float_of_int
bool	true false	&&	
char	'a' 'b'		char_of_int ,
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

Tipos básicos, literais, operadores e funções

int	1 2 3 4	+ - * /	
float	1. 2. 3.5 4e10	+. -. *.	float_of_int
bool	true false	&&	
char	'a' 'b'		char_of_int ,
strings	“hello” “”	^	string_of_int
unit	()		

The screenshot shows a code editor interface with a tooltip or dropdown menu. The menu contains the expression `"abc".[0]`. Below the expression, there is a status bar with the text `[7] ✓ 0.0s`, indicating the result of the operation. At the bottom of the screen, a portion of the code is visible, showing the assignment `... - : char = 'a'`.

Igual e Igual

=	<>	structural
==	!=	physical (references, arrays, etc.)

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

Expressões condicionais

- as expressões condicionais são expressões onde os dois ramos têm o mesmo tipo

```
▷ ▾ [9] 4 + (if 'a' = 'b' then 1 else 2)
          ✓ 0.0s
...
... - : int = 6
```

- o ramo “else” é obrigatório para todos os tipos excepto **unit**

```
▷ ▾ [10] if "hello" = "world" then 1
          ✘ ✦ 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
```

Declaração de variáveis (matemáticas)

- declarações top-level

```
▶ ~ let x = 42
[12] ✓ 0.0s
... val x : int = 42
```

- declarações locais

```
▶ ~ let x = 42 in (string_of_int x)^": the ultimate question of life, the universe, and everything"
[13] ✓ 0.0s
... - : string =
"42: the ultimate question of life, the universe, and everything"
```

```
▶ ~ let x = 1 in (let x = 2 in x + 1) + x
[14] ✓ 0.0s
... - : int = 4
```

```
▶ ~ let x = 1 in let y = 2 in x + y
[15] ✓ 0.0s
... - : int = 3
```

Declaração de variáveis (âmbito)

- A declaração de um nome (x) é limitada ao corpo da declaração (e2)
- o nome (x) não é visível na expressão que define o valor (e1)

```
let x=e1 in e2
```

```
let y=let y=y+1 in y+1 in let y=y+2 in y+2
```

- As declarações obedecem ao princípio da irrelevância dos nomes onde os nomes escolhidos não devem ser relevantes para a avaliação de uma expressão.

Declaração e chamada (aplicação) de funções

- A declaração de um nome (*f*) é limitada ao corpo da declaração (e2)
- o nome (*f*) não é visível na expressão que define o valor (e1)

```
let f x = x+1 in f 1
```

Declaração recursivas (âmbito)

- A declaração de um nome (x) é visível no corpo da declaração (e2) e na expressão de definição do nome (e1).

```
let rec x = e1 in e2
```

```
let rec fact x = if x = 0 then 1 else x * fact(x-1)
```

Declaração recursivas (âmbito)

- A declaração de um nome (x) é visível no corpo da declaração (e2) e na expressão de definição do nome (e1).

```
let rec x = e1 in e2
```

```
(* pre: x ≥ 1
   returns true if x is even false otherwise *)
let rec even x = if x = 0 then true else if x = 1 then false else odd(x-1)

(* pre: x ≥ 1
   returns true if x is odd false otherwise *)
and odd x = if x = 0 then false else if x = 1 then true else even(x-1)
```

Declarações mutuamente recursivas em C

- Tem que se declarar a função sem a definir.

```
bool odd(int x);
```

```
bool even(int x) {  
    if( x == 0 ) {  
        return false;  
    } else if( x == 1 ) {  
        return false;  
    } else {  
        return odd(x-1);  
    }  
}
```

```
bool odd(int x) {  
    if( x == 0 ) {  
        return false;  
    } else if( x == 1 ) {  
        return true;  
    } else {  
        return even(x-1);  
    }  
}
```