

Aula 1: Introdução à Programação Funcional em OCaml

UC: Programação Funcional
2024-2025

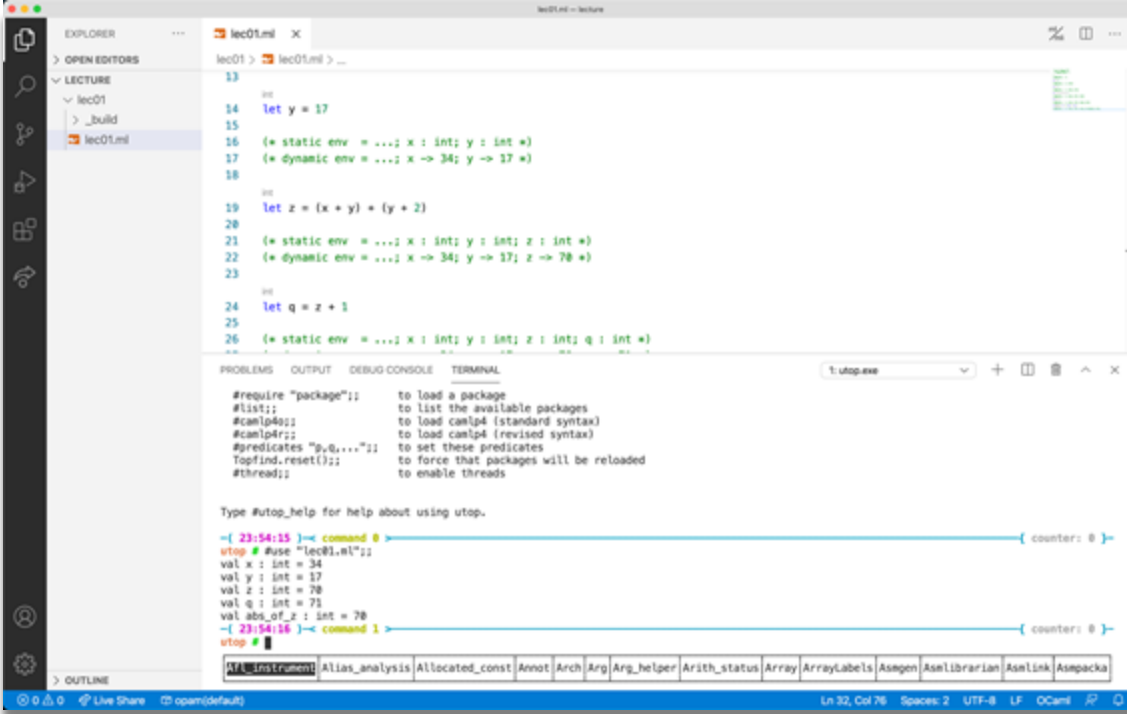
Programação Funcional

- Fundamentos: concepção, aplicação, teoria, prática
- Ver os fundamentos através do [OCaml](#)
 - “Viajar para ver de onde se é”
- Na [programação funcional](#) (FP)
 - *Evitar a mutação* (declarações de atribuição)
 - *As funções são valores*



Um mundo totalmente novo

- Uma nova linguagem (OCaml)
- Um novo editor (VS Code, etc.)
- Vamos preparar-nos e começar a aprender o ecossistema OCaml
- Quanto mais cedo melhor - não adiem a luta contra os problemas da instalação



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left shows a project named 'lec01' with files 'lec01.ml' and '._build'. The main editor window displays the 'lec01.ml' file with OCaml code. The code includes variable declarations, static and dynamic environment updates, and arithmetic operations. Below the editor, the 'TERMINAL' panel is open, showing the output of the 'utop.exe' command. The terminal output lists various OCaml commands and their descriptions, such as '#require', '#list', '#camlp4', and '#thread'. It also shows the execution of a command to load the 'lec01.ml' file and the resulting environment state.

```
13
14 let y = 17
15
16 (* static env = ...; x : int; y : int *)
17 (* dynamic env = ...; x -> 34; y -> 17 *)
18
19 let z = (x + y) + (y + 2)
20
21 (* static env = ...; x : int; y : int; z : int *)
22 (* dynamic env = ...; x -> 34; y -> 17; z -> 70 *)
23
24 let q = z + 1
25
26 (* static env = ...; x : int; y : int; z : int; q : int *)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

Type #utop_help for help about using utop.

```
~( 23:54:15 )-> command 0
utop # use "lec01.ml";;
val x : int = 34
val y : int = 17
val z : int = 70
val q : int = 71
val abs_of_z : int = 70
~( 23:54:16 )-> command 1
utop #
```

VS Code status bar: Ln 32, Col 76, Spaces: 2, UTF-8, LF, OCaml

Mindset

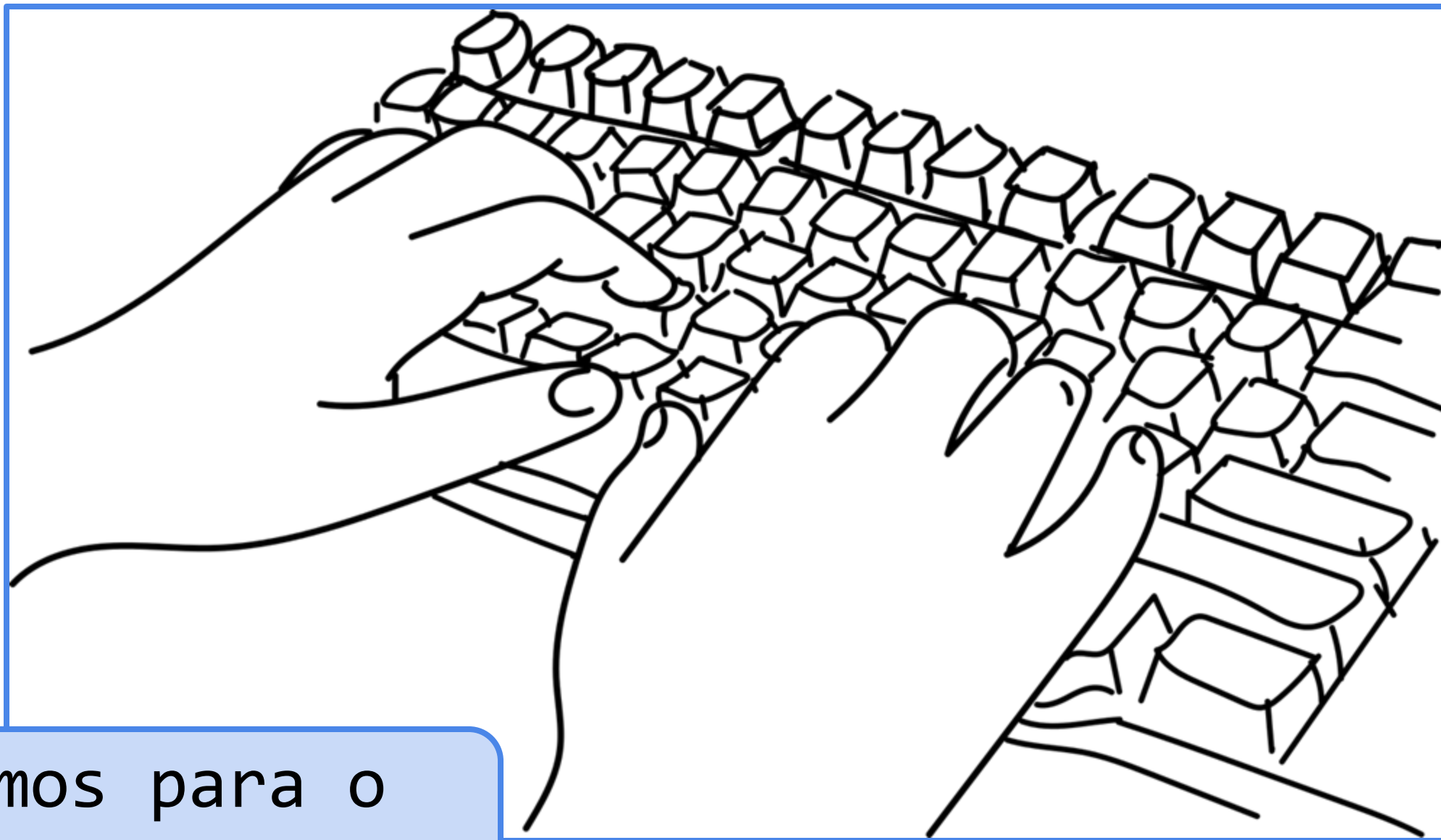
- É demasiado fácil esquecer como a programação foi, inicialmente, alucinante
 - *“Quando estamos a pensar em algo que não compreendemos, temos uma sensação terrível e desconfortável chamada confusão.”*
 - **Aprender coisas novas significa muitas vezes trabalhar através da confusão: não faz mal!**
- É demasiado fácil pensar que tudo é um prego e que só precisamos de martelos
 - *“Tornamo-nos naquilo que contemplamos. Nós moldamos as nossas ferramentas e depois as nossas ferramentas moldam-nos a nós.”*
 - **Aprender um conjunto diversificado de ferramentas torna-nos mais robustos quando enfrentamos novos problemas!**

Mindset

- Tratar o OCaml ("ML") como um novo jogo alienígena que estamos a aprender a jogar 🧐 🎲
- Vamos descrever OCaml de forma sistemática, e depois aplicar a mesma abordagem repetidamente
- Muitos conceitos novos, mas seremos capazes de comparar e contrastar linguagens de forma mais rigorosa
- **Tentar “traduzir tudo para C” vai atrasá-lo**
- Motivaremos a UC... daqui a umas semanas
- Quando tivermos vocabulário partilhado

Sintaxe + Semântica

- Sintaxe: como os programas são escritos
 - Aproximadamente “ortografia e gramática”
- Semântica: o que ***significam*** os programas
 - Verificar tipo “Type checking” : semântica em tempo de compilação (também conhecida como “estática”)
 - Avaliar “Evaluation” : semântica em tempo de execução (também conhecida como “dinâmica”)



Vamos para o
teclado!

Olá mundo!

```
(* o nosso primeiro programa *)  
let x = print_string "Olá Programação Funcional!\n"
```

- Um *programa* é uma sequência de ligações (“*bindings*”)
- Um tipo de *binding* é uma “*variable binding*”
- A avaliação avalia os *bindings* por ordem
- Para avaliar um *variable binding*:
 - Avaliar a expressão (à direita de `=`) no ambiente criado pelos *bindings* anteriores.
 - Isto produz um valor.
 - Estender o ambiente (“*top-level*”), ligando a variável ao valor.

Definição da ligação de variáveis

let **x** = **e**

Sintaxe:

- “Keyword”: **let**
- Variável: **x**
- Expressão: **e**
 - Várias Formas!
 - Definido *recursivamente*!

```
(* static env = ... *)  
(* dynamic env = ... *)  
let x = 34  
  
(* static env = ...; x : int *)  
(* dynamic env = ...; x -> 34 *)
```

Definição da ligação de variáveis

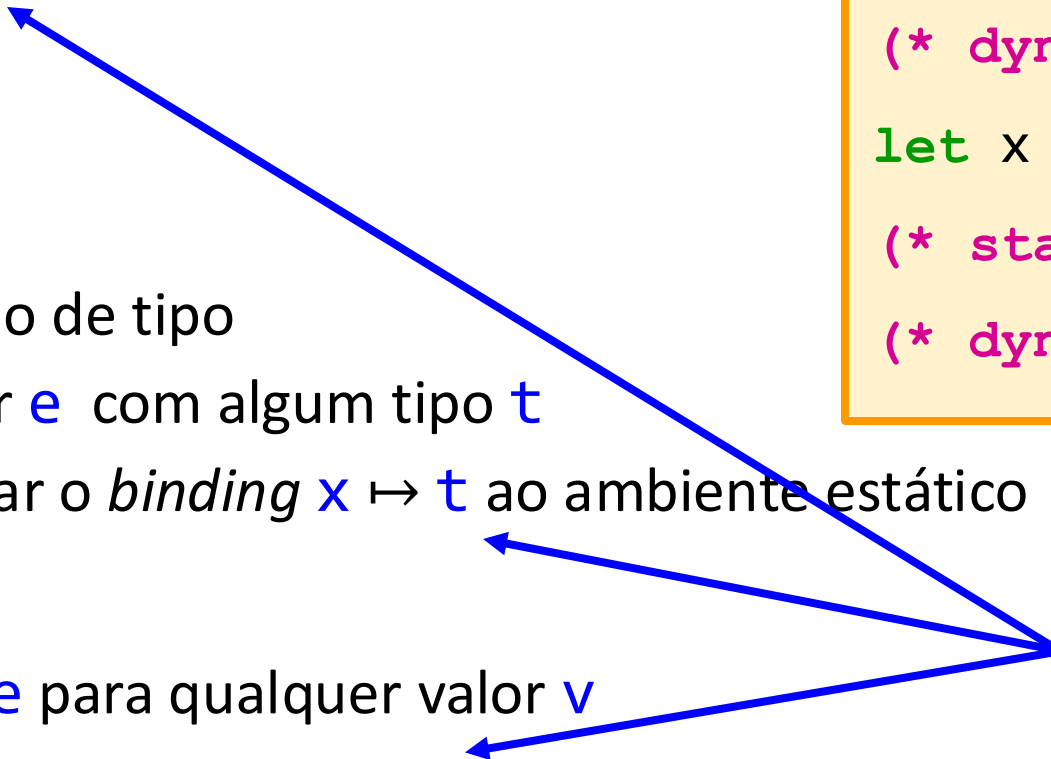
let *x* = *e*

Semântica:

- Verificação de tipo
 - Verificar *e* com algum tipo *t*
 - Adicionar o *binding* $x \mapsto t$ ao ambiente estático
- Avaliar
 - Avaliar *e* para qualquer valor *v*
 - Adicionar o *binding* $x \mapsto v$ ao ambiente dinâmico

```
(* static env = ... *)  
(* dynamic env = ... *)  
let x = 34  
  
(* static env = ...; x : int *)  
(* dynamic env = ...; x -> 34 *)
```

metavariáveis em azul



Algumas variações

```
() (* Unit *)  
let x = print_string "Hello, World!\n"  
(* igual ao anterior, sem nada ligado a () *)  
let _ = print_string "Hello, World!\n"  
(* o mesmo mas com variáveis e uma função de concatenação infix *)  
let h = "Hello, "  
let w = "World!\n"  
let _ = print_string (h ^ w)  
(* função f: ignora o seu argumento e imprime *)  
let f x = print_string (h ^ w)  
(* para que ambos sejam impressos (a chamada é justaposta) *)  
let y1 = f 37  
let y2 = f f (* passar a própria função *)  
let y3 = y1 (* mas aqui não acontece o mesmo, y1 liga a () *)
```

Compilação/execução

<code>ocamlc file.ml</code>	compilar para <i>bytecode</i> (executável)
<code>ocamlopt file.ml</code>	compilar para código máquina - nativo (1-5x mais rápido, não deverá ser necessário por agora)
<code>ocamlc -i file.ml</code>	imprimir tipos de todas os top-level bindings (uma interface)
<code>ocaml</code>	read-eval-print loop (ver manual de instruções)
<code>ocamlprof,</code> <code>ocamldebug, ...</code>	consultar o manual (provavelmente desnecessário)

- Mais tarde: vários ficheiros

Instalar, aprender

- Ligações a partir da página Web:
 - **`www.ocaml.org`**
 - O manual on-line - <https://v2.ocaml.org/manual/> (excelente referência)
 - Um livro on-line (menos referência, mas muito útil – “OCaml From the Very Beginning” [Versão online](#))
 - Instruções de instalação/utilização (1º Capítulo do “OCaml Programming: Correct + Efficient + Beautiful” [Versão online](#))
- Contactem-nos rapidamente se tiverem problemas de instalação!
- Coloquem questões (conhecemos a linguagem, queremos partilhar, o paradigma funcional é divertido 🎲)

Porquê duas semânticas?

- A verificação de tipos (semântica estática) dá-nos primeiro uma garantia (prova o invariante!)
 - Acontece antes de um programa começar a ser executado
- A avaliação (semântica dinâmica) executa o programa
 - Devido às verificações de tipo de programa, muitos erros de tempo de execução são **impossíveis**
 - Exemplo: **1 + "hello"**
 - Não existe uma semântica dinâmica para este [não-]programa
 - Não nos interessa! Nunca tentaremos executar um programa que avalie essa expressão

Recapitular

- *Um programa é apenas uma sequência de ligações (“bindings”)*
- Verificação de tipo (“*typecheck*”) de cada ligação é feita por ordem
 - Utilizar **ambiente estático** de ligações anterior
- Avaliar cada ligação por ordem
 - Utilizar o **ambiente dinâmico** de ligações anterior
- Até agora, vimos apenas as ligações de variáveis (“*variable bindings*”)
 - Mais tipos de ligações em breve!

Créditos para Dan Grossman.