

Linguagens e Ambientes de Programação (Aula Teórica 3)

LEI - Licenciatura em Engenharia Informática

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



NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

Agenda para hoje

- Declaração de nomes, outra vez.
- Declaração de funções, com e sem parâmetros.
- Funções como valores.
- Avaliação de expressões por substituição.
- Funções de biblioteca
- Input/output básico
- Sequências, como ignorar valores intermédios

Declarações

Declaração de variáveis (matemáticas)

- declarações top-level
- declarações locais

```
▷ ▾  
    let x = 42  
[12] ✓ 0.0s  
... val x : int = 42
```

```
▷ ▾  
    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=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 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=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 importantes para a avaliação de uma expressão.

Âmbito das declarações locais

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

```
let w = 3+x in  
w + x
```

x

Âmbito das declarações locais

let x =

let y = 1 in

let z = 2 in

y + z

Y

in

let w = 3+x in

w + x

Âmbito das declarações locais

let x =

let y = 1 in

let z = 2 in

y + z

z

in

let w = 3+x in

w + x

Âmbito das declarações locais

```
let x =  
  let y = 1 in  
  let z = 2 in  
  y + z  
in  
  let w = 3+x in
```

w + x

w

Avaliação das declarações por substituição

let x =

```
let y = 1 in
```

```
let z = 2 in
```

```
y + z
```

in

```
let w = 3+x in
```

```
w + x
```

Avaliação das declarações por substituição

```
let x =  
  let y = 1 in  
  let z = 2 in  
  y + z  
in  
  let w = 3+x in  
  w + x
```

Avaliação das declarações por substituição

let x =

```
let z = 2 in
```

```
1 + z
```

in

```
let w = 3+x in
```

```
w + x
```

Avaliação das declarações por substituição

let x =

let z = 2 in

1 + z

in

let w = 3+x in

w + x

Avaliação das declarações por substituição

let $x =$

$1 + 2$

in

let $w = 3 + x$ in

$w + x$

Avaliação das declarações por substituição

let $x =$

3

in

let $w = 3 + x$ in

$w + x$

Avaliação das declarações por substituição

```
let w = 3+3 in  
w + 3
```

Avaliação das declarações por substituição

let $w = 3+3$ in
 $w + 3$

Avaliação das declarações por substituição

let $w = 9$ in

$w + 3$

Avaliação das declarações por substituição

9 + 3

Avaliação das declarações por substituição

12

Funções

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)
- os parâmetros são listados na declaração.

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

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)
- os parâmetros são listados na declaração.
- as funções “sem parâmetros” têm um parâmetro de tipo null.

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

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)
- os parâmetros são listados na declaração.
- a declaração com parâmetros é uma forma sintática alternativa à utilização de valores de tipo função (o símbolo seta é composto por dois caracteres ->)

`let f = fun x → x+1 in f (1+1)`

Definição e chamada (aplicação) de funções

- A aplicação de funções pode ser definida por substituição do parâmetro pelo valor do argumento.
- OCaml implementa uma estratégia de avaliação *call-by-value*, o que quer dizer que os argumentos são avaliados antes de expandir o corpo da função.

$$\begin{aligned} & (\text{fun } x \rightarrow x+1) (1+1) \\ & \quad (\text{fun } x \rightarrow x+1) 2 \\ & \qquad 2+1 \\ & \qquad 3 \end{aligned}$$

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);  
    }  
}
```

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)
- os parâmetros são listados na declaração.

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

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)
- os parâmetros são listados na declaração.

`let f = fun x y → x+y in f 1 1`

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)
- os parâmetros são listados na declaração.

```
let f = fun x → fun y → x+y in f 1 1
```


Tipo Seta (Função)

Anotações de tipos

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)
- os parâmetros são listados na declaração.
- as funções “sem parâmetros” têm um parâmetro de tipo null.

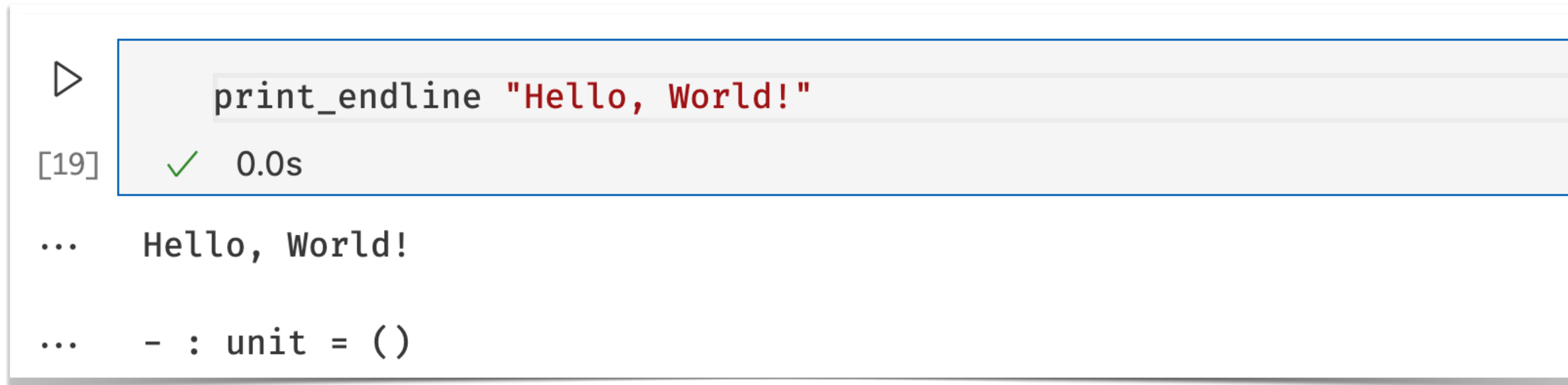
`let f = let x = 1 in fun x → 1+x in f ()`

Exercício arrow

Input/Output

Input/Output Básico

- O tipo `unit` só faz sentido em expressões que têm efeitos laterais, a impressão é um exemplo típico disso
- `print_endline`
- `print_string`
- `print_char`
- `print_int`
- `print_float`



The screenshot shows a code editor with a blue border. Inside, the code `print_endline "Hello, World!"` is written. Below the code, there is a green checkmark and the text `[19] 0.0s`. Below that, the output `... Hello, World!` is shown. At the bottom, there is a line of code `... - : unit = ()`.

Declaração como operador de sequência.

- declarações que ignoram resultados

```
▷ let () = print_string "hello, " in print_endline "world!"  
[14] ✓ 0.0s
```

```
... hello, world!
```

```
... - : unit = ()
```

```
▷ let _ = uma_funcao_com_efeitos_laterais () in 4  
[13] ✓ 0.0s
```

```
... - : int = 4
```

```
▷ print_string "hello, "; print_endline "world!"  
[15] ✓ 0.0s
```

```
... hello, world!
```

```
... - : unit = ()
```

Declaração como operador de sequência.

- declarações que ignoram resultados

```
[18] 1; 2
      ✓ 0.0s

... File "[18]", line 1, characters 0-1:
    1 | 1; 2
        ^

Warning 10 [non-unit-statement]: this expression should have type unit.
File "[18]", line 1, characters 0-1:
    1 | 1; 2
        ^

Warning 10 [non-unit-statement]: this expression should have type unit.

... - : int = 2
```


Declaração como operador de sequência.

- declarações que ignoram resultados

▶

```
let () = print_string "hello, " in print_endline "world!"
```

[14] ✓ 0.0s

... hello, world!

▶

```
(ignore 1); 2
```

[21] ✓ 0.0s

... - : int = 2

✓

```
print_string "hello, "; print_endline "world!"
```

[15] ✓ 0.0s

... hello, world!

... - : unit = ()

Sumário

- Declaração de nomes, outra vez.
- Declaração de funções, com e sem parâmetros.
- Funções como valores.
- Avaliação de expressões por substituição.
- Funções de biblioteca
- Input/output básico
- Sequências, como ignorar valores intermédios