

Lezione 2: tipi e funzioni predefinite in OCaml

Corso di Linguaggi di Programmazione

Vincenzo Ciancia - ISTI/CNR Pisa

vincenzo.ciancia@isti.cnr.it



Un po' di programmazione spicciola

Come si programma in pratica in OCaml?

Un file OCaml ha estensione “.ml”

Le dichiarazioni vengono lette ed eseguite ad una ad una.

```
let x = 3;;  
let z = print_string "Hello world!";;  
let _ = print_string "Hello world!";;  
print_string "Hello world!";;
```



print???

I linguaggi funzionali, in linea di principio, non hanno stato o side-effects

Q: cosa sono i side effects?

A: le modifiche allo “stato del mondo” che un programma può fare comunicando con il mondo esterno

E' sempre necessario comunicare col mondo esterno?

Side effects??

La risposta lunga, è... molto lunga, e passa per Miranda, Haskell, le *monadi*.

La risposta breve è: SI, è necessario comunicare con il mondo esterno.

La funzione “`print_string`” fa questo.

Come side effect, a noi serve soprattutto stampare, dato che usiamo OCaml solo come meta-linguaggio

Funzioni di stampa in OCaml

```
# print_string;;  
- : string -> unit = <fun>  
  
# print_int;;  
- : int -> unit = <fun>  
  
# print_string "ciao";;  
ciao- : unit = ()  
  
# print_int 3;;  
3- : unit = ()  
  
# Printf.printf "%s è una stringa, ma  
%d è un intero\n" "Questa" 3;;  
Questa è una stringa, ma 3 è un intero  
- : unit = ()
```

Il tipo “unit” ha un solo valore: ()

Serve a “non restituire niente” restituendo un valore non-informativo

Domanda:

In matematica (quindi, senza side effects!) dato un insieme S e un singolo $\{x\}$,

Quante funzioni ci sono da S a $\{x\}$?

Risposta:

C'è una sola funzione da S a $\{x\}$, quella che restituisce sempre x .

I side effects sono “fuori” dalla teoria degli insiemi, quindi fuori dal nostro semplice modello mentale matematico

Conviene separare chiaramente i side effects dalla matematica: scrivere funzioni senza interazione, poi chiamarle dal “top-level”, cioè dalla sequenza di istruzioni principale.

Invocazione di un programma

Un programma OCaml è una sequenza di dichiarazioni e/o statement in un file .ml

Esecuzione: `ocaml nomefile.ml`

Compilazione: `ocamlc nomefile.ml -o nomefile(.exe)`

Esecuzione del codice compilato: `./nomefile(.exe)`

Caveat

Particolarità del parser di ocaml possono causare strani errori nelle istruzioni del programma principale. Per evitarli usare la forma:

```
let main = ISTRUZIONI
```

invece di usare semplicemente

```
ISTRUZIONI
```

Non si vedono i side effects nei tipi delle funzioni

```
# let test () = ();;
val test : unit -> unit = <fun>
# test;;
- : unit -> unit = <fun>
# print_newline;;
- : unit -> unit = <fun>
# test ();;
- : unit = ()
# print_newline ();;

- : unit = ()
```

Una funzione di stampa “comoda”

Per comodità, invece di `print_int`, `print_string`, `print_newline` e quant'altro, useremo la versione OCaml della `printf` del C.

Una funzione dal tipo complicato

Costruita “col cacciavite” e non definibile dall'utente

Formalmente, una funzione di libreria, in pratica, più parte del linguaggio che delle librerie standard

Printf.printf

La funzione da usare è chiamata “printf” e si trova nel *modulo* Printf.

Non sapendo cosa sono i moduli, all'inizio del programma scriviamo

```
Open Printf;;  
# printf "ciao\n";;  
ciao  
- : unit = ()
```

Sintassi alternativa

```
# Printf.printf "ciao\n";;
```

Quando si usa solo “printf”, funziona solo se si è usato “open Printf” prima.

Printf.printf, al contrario, funziona sempre, ma è più lungo da scrivere e da leggere

Placeholders

La funzione `printf` è comoda perchè ci permette di inserire variabili e risultati di espressioni al posto di particolari sequenze di caratteri chiamate *placeholder*

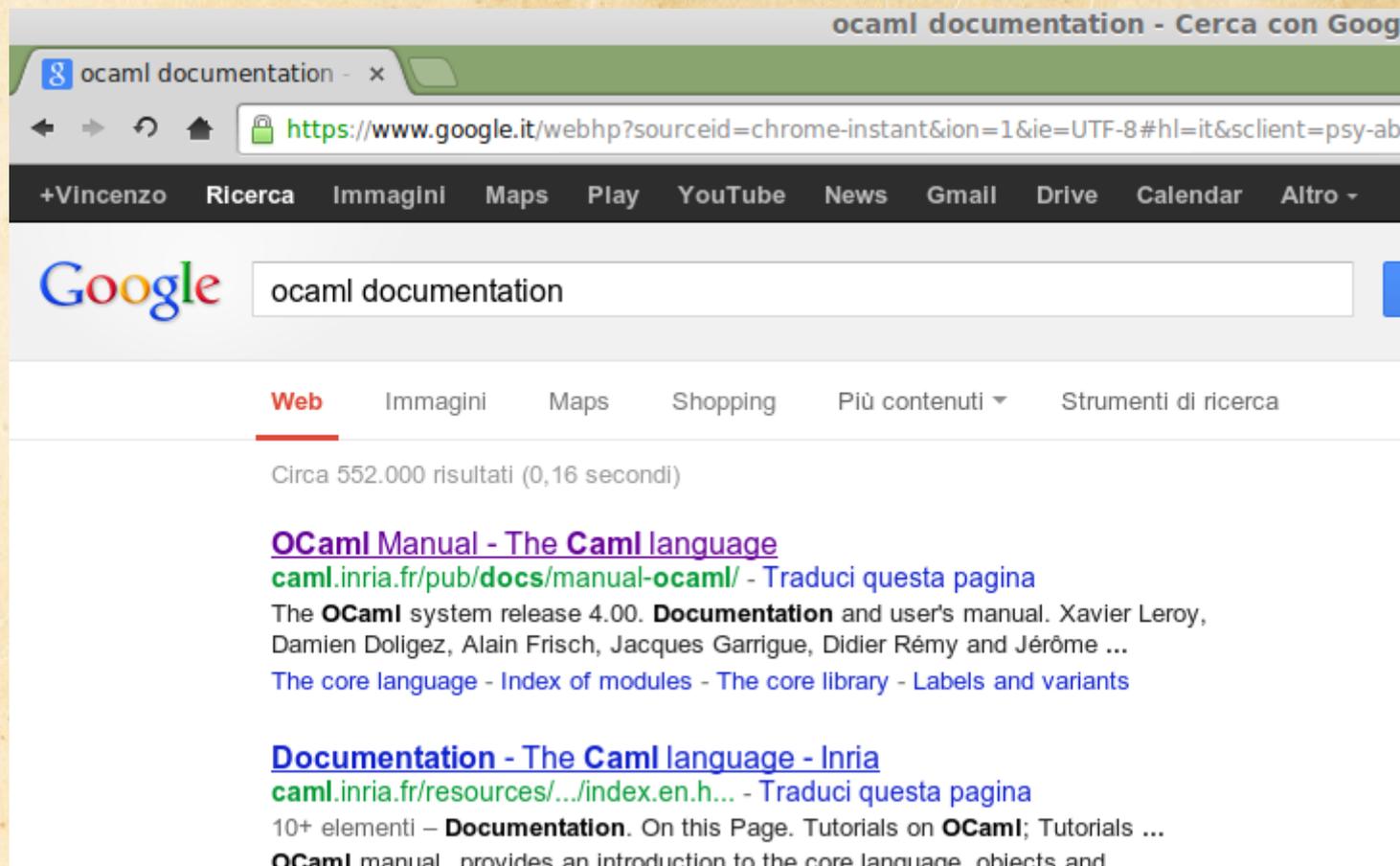
```
# printf "La somma di %s e quattro ha %i  
cifre e fa %i\n" "tre" 1 (3+4);;
```

La somma di tre e quattro ha 1 cifre e fa
7

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

Dove trovare documentazione su printf?

Nel manuale di OCaml!



The screenshot shows a Google search interface. The search bar contains the text "ocaml documentation". Below the search bar, the "Web" tab is selected, and the search results are displayed. The first result is titled "OCaml Manual - The Caml language" and includes the URL "caml.inria.fr/pub/docs/manual-ocaml/". The second result is titled "Documentation - The Caml language - Inria" and includes the URL "caml.inria.fr/resources/.../index.en.h...".

ocaml documentation - Cerca con Google

ocaml documentation - x

https://www.google.it/webhp?sourceid=chrome-instant&ion=1&ie=UTF-8#hl=it&sclient=psy-ab

+Vincenzo Ricerca Immagini Maps Play YouTube News Gmail Drive Calendar Altro -

Google ocaml documentation

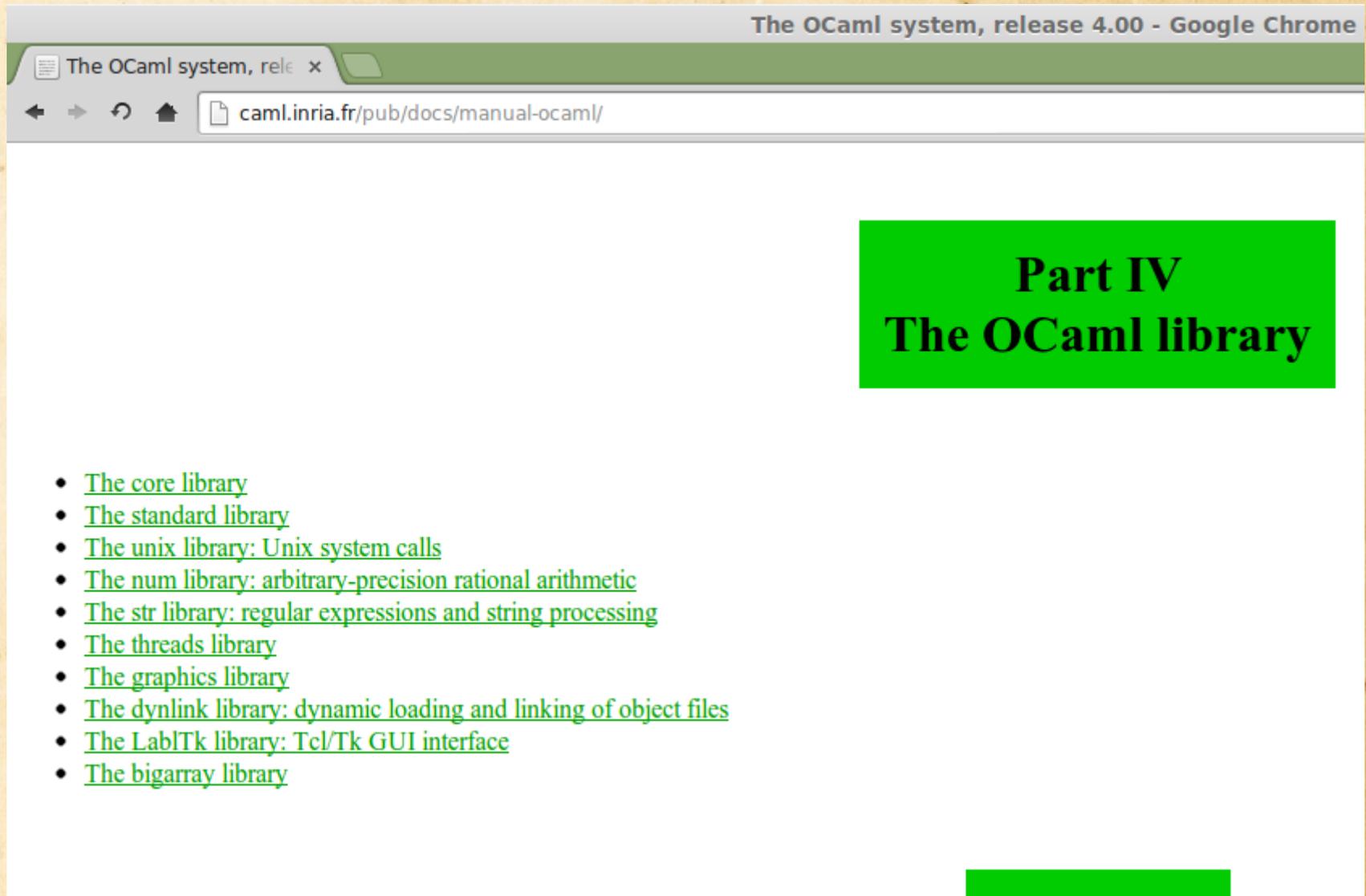
Web Immagini Maps Shopping Più contenuti ▾ Strumenti di ricerca

Circa 552.000 risultati (0,16 secondi)

[OCaml Manual - The Caml language](#)
caml.inria.fr/pub/docs/manual-ocaml/ - Traduci questa pagina
The **OCaml** system release 4.00. **Documentation** and user's manual. Xavier Leroy, Damien Doligez, Alain Frisch, Jacques Garrigue, Didier Rémy and Jérôme ...
[The core language](#) - [Index of modules](#) - [The core library](#) - [Labels and variants](#)

[Documentation - The Caml language - Inria](#)
caml.inria.fr/resources/.../index.en.h... - Traduci questa pagina
10+ elementi - **Documentation**. On this Page. Tutorials on **OCaml**; Tutorials ...
OCaml manual provides an introduction to the core language, objects and

Nella standard library, nel modulo printf:



The screenshot shows a Google Chrome browser window with the title "The OCaml system, release 4.00 - Google Chrome". The address bar contains the URL "caml.inria.fr/pub/docs/manual-ocaml/". The main content area displays a green box with the text "Part IV The OCaml library". Below this, a list of library components is shown, each preceded by a bullet point and underlined.

**Part IV
The OCaml library**

- [The core library](#)
- [The standard library](#)
- [The unix library: Unix system calls](#)
- [The num library: arbitrary-precision rational arithmetic](#)
- [The str library: regular expressions and string processing](#)
- [The threads library](#)
- [The graphics library](#)
- [The dynlink library: dynamic loading and linking of object files](#)
- [The LablTk library: Tcl/Tk GUI interface](#)
- [The bigarray library](#)

Controllo di sequenza

Se un programma ha dei side effects, diventa importante l'**ordine di valutazione** delle istruzioni.

Per questo nel linguaggio ocaml c'è l'operatore di sequenza “;” che consente di eseguire due o più istruzioni di seguito

```
let main =  
  Printf.printf "Prima riga\n";  
  Printf.printf "Seconda riga\n";  
  Printf.printf "Terza riga \n";;
```

Definizioni per casi e *pattern matching*

Pattern matching

```
# type int_or_string = I of  
int | S of string;;
```

```
type int_or_string = I of  
int | S of string
```

```
# let is_int x =  
    match x with  
        I i_x -> true  
        | S s_x -> false;;
```

```
val is_int : int_or_string  
-> bool = <fun>
```

```
# is_int (S "c");;
```

```
- : bool = false
```

```
# is_int (I 1);;
```

```
- : bool = true
```

Molte sintassi alternative

```
# let is_int x =  
  match x with  
    | I i_x -> true  
    | S s_x -> false;;
```

```
# let is_int = function  
  | I x_i -> true  
  | S x_s -> false;;
```

```
# let is_int = fun x ->  
  match x with  
    | I x_i -> true  
    | S x_s -> false;;
```

Cosa vuol dire “match”?

match x with $C(x_1, x_2, \dots, x_n)$ ->
ESPRESSIONE(x_1, x_2, \dots, x_n, x)

Trova la più semplice istanziatura delle variabili x_1, \dots, x_n che rende x uguale a $C(x_1, \dots, x_n)$

Se esiste!

Cosa succede se non esiste?

Provate a definire

```
let is_int x =  
  match x with  
    I x_i -> true;;
```

Cosa succede se si chiede all'interprete:

```
# is_int (S "ciao");;
```

Cosa vuol dire “fun”?

Le funzioni in OCaml sono valori, così come gli interi e le stringhe.

In OCaml posso scrivere la stringa “ciao” e l'intero 3, senza dovergli dare un nome

Posso scrivere anche una funzione “immediatamente” senza doverle dare un nome?

```
# (fun x -> x + 1) ( 2 );;  
- : int = 3
```

Notazione, notazione...

Ecco spiegata la seguente notazione, che è solo un'associazione nome/valore:

```
let is_int =  
  (fun x ->  
    match x with  
      | I x_i -> true  
      | S x_s -> false);;
```

La notazione “function x -> ...” combina match e fun in un solo costrutto:

```
# let is_int = (function  
  | I x_i -> true  
  | S x_s -> false);;
```

Pattern matching su coppie

```
# let proj1 x =  
    match x with  
        (x1,x2) -> x1;;  
val proj1 : 'a * 'b -> 'a = <fun>  
  
# proj1 (1,2);;  
- : int = 1  
  
# proj1 ("ciao",3);;  
- : string = "ciao"
```

Funzioni ricorsive

```
# type nat = Zero | Succ of nat;;

# let rec to_int x =
  match x with
    Zero -> 0
  | Succ y -> 1 + (to_int y);;

# to_int (Succ(Succ(Zero)));;
- : int = 2
```

Un po' di matematica spicciola

Definizioni induttive

Esercizio: prendete carta e penna

Definite per induzione l'insieme S delle sequenze finite di interi

Domanda: come si fa a definire l'insieme S delle sequenze finite di elementi presi da un insieme I ?
Come chiamereste tale insieme?

Definizioni induttive

Esercizio: definite tre funzioni per induzione sull'insieme S

- 1) $\text{length} : S \rightarrow \text{Nat}$ (quanti elementi?)
- 2) $\text{sum} : S \rightarrow \text{Int}$ (somma degli elementi?)
- 3) $\text{concat} : S * S \rightarrow S$ (concatenazione)

Almeno 2 soluzioni per S

Soluzione 1: S è il minimo insieme che contiene le sequenze vuote, gli interi, ed è chiuso per concatenazione di due sequenze

$\text{Empty} \in S ; i \in \text{Int} \Rightarrow i \in S ; s_1, s_2 \in S \Rightarrow \text{Concat}(s_1, s_2) \in S$

Soluzione 2: S è il minimo insieme che contiene le sequenze vuote, ed è chiuso per concatenazione di un intero e una sequenza.

$\text{Empty} \in S ; s \in S , i \in \text{Int} \Rightarrow \text{Insert}(i, s) \in S$

Almeno 2 soluzioni per S

Nella soluzione 1, abbiamo bisogno di assiomi aggiuntivi per dire ad esempio che

$$\text{concat}([1,2],[3]) = \text{concat}([1],[2,3])$$

Nella soluzione 2, non serve nessun assioma!

Le “costruzioni libere” (= libere da assiomi) come la 2 sono rappresentabili in OCaml, le altre non in maniera automatica.

Definizioni induttive

Esercizio: definire le seguenti funzioni per induzione sull'insieme S

- 1) $\text{length} : S \rightarrow \text{Nat}$ (quanti elementi?)
- 2) $\text{sum} : S \rightarrow \text{Int}$ (somma degli elementi?)
- 3) $\text{concat} : S * S \rightarrow S$ (concatenazione?)
- 4) $\text{head} : S \rightarrow \text{Int}$ (primo elemento?)
- 5) $\text{tail} : S \rightarrow S$ (tutto tranne il primo elemento?)

Passiamo a OCaml

Esercizio:

Definire in Ocaml il tipo s delle sequenze di interi, che sono o la sequenza vuota `Empty`, o la sequenza `Insert(i,s)` dove i è un intero e s una sequenza di interi

Definire la funzione $\text{length} : s \rightarrow \text{int}$

Calcolare

```
length (Insert(3,Insert(4,Empty)))
```

Definire strutture dati in OCaml

Soluzione: length

```
# type s = Empty | Insert of (int * s);;
type s = Empty | Insert of (int * s)

# let rec length x =
  match x with
  | Empty -> 0
  | Insert(n,y) -> 1 + (length y);;
val length : s -> int = <fun>

# length (Insert(3,Insert(4,Empty)));;
- : int = 2
```

Definire le funzioni...

sum : s -> int

concat : s * s -> s

Sum

```
# let rec sum x =  
  match x with  
    Empty -> 0  
  | Insert (i,y) -> i + (sum y);;  
  
sum (Insert(3,Insert(5,Empty)));;
```

Concat

```
# let rec concat (x,y) =
  match x with
    Empty -> y
  | Insert(i,z) -> Insert(i,concat (z,y));;

val concat : s * s -> s = <fun>

# concat (Empty,Empty);;
- : s = Empty

# concat (Empty,Insert(3,Insert(4,Empty)));;
- : s = Insert (3, Insert (4, Empty))

# concat (Insert(2,Insert(1,Empty)),Insert(3,Insert(4,Empty)));;
- : s = Insert (2, Insert (1, Insert (3, Insert (4, Empty))))

# concat (Insert(2,Insert(1,Empty)),Empty);;
- : s = Insert (2, Insert (1, Empty))
```

Esercizio

Definire la funzione “prodotto” che moltiplica due sequenze di interi

Q: cosa fare quando la lunghezza delle due sequenze non è uguale?

Generalizzare

Definire una funzione

`applica2 : (s * s) → (int * int →
int) → s`