



FONDAMENTI DI INFORMATICA

Alma Artis

Francesca Pratesi (ISTI, CNR)

Struttura di un programma e funzioni

- $\neg\neg p = p$
- $p \wedge (q \vee r) = (p \wedge q) \vee (p \wedge r)$
- $p \vee (q \wedge r) = (p \vee q) \wedge (p \vee r)$
- $(p \Rightarrow q) = (\neg q \Rightarrow \neg p)$
- $\neg(p \wedge q) = \neg p \vee \neg q$
- $\neg(p \vee q) = \neg p \wedge \neg q$



$$\neg\neg p = p$$

p	!p	!(!p)
T	F	T
F	T	F

– $(p \Rightarrow q) = (\neg q \Rightarrow \neg p)$

p	q	$p \Rightarrow q$	$\neg q$	$\neg p$	$\neg q \Rightarrow \neg p$
T	T	T	F	F	T
T	F	F	T	F	F
F	T	T	F	T	T
F	F	T	T	T	T

$$P \wedge (Q \vee R) = (P \wedge Q) \vee (P \wedge R)$$

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge q$	$p \wedge r$	$(p \wedge q) \vee (p \wedge r)$
T	T	T	T	T	T	T	T
T	F	T	T	T	F	T	T
F	T	T	T	F	F	F	F
F	F	T	T	F	F	F	F
T	T	F	T	T	T	F	T
T	F	F	F	F	F	F	F
F	T	F	T	F	F	F	F
F	F	F	F	F	F	F	F

ESERCIZI

- Scrivere un programma che inizializzi due variabili x ed y (assegnando valori a piacere) e visualizzi il risultato dell'operazione x^y .
- Scrivere un programma che inizializzi una variabile x con un valore numerico a piacere e, interpretandolo come l'area di un cerchio, visualizzi l'area del cerchio il cui diametro è il doppio di quello iniziale. Suggerimento: si calcoli prima il raggio del cerchio con area data da x , poi si prosegue con il ragionamento. La radice si fa con `Math.sqrt()`
- Scrivere un programma che inizializzi due variabili N ed M con valori numerici scelti a piacere. Il programma deve successivamente calcolare e visualizzare:
 - se N è minore o uguale di M
 - se N è compreso tra 0 ed M
 - se N è divisibile per M
 - in tutti e quattro i casi, prevedere una stampa informativa opportuna
- Scrivere un programma che calcoli il massimo e il minimo tra due variabili N ed M , usando due metodi risolutivi diversi



LIBRI E RIFERIMENTI

- Capitolo 2

Eloquent Javascript – Second Edition

Marijn Haverbeke

Licensed under CC license.

Available here: <http://eloquentjavascript.net/>



STRUTTURA DI UN PROGRAMMA JAVASCRIPT

ENVIROMENT

- L'insieme delle variabili e i loro rispettivi valori ad un certo istante sono chiamate environment
- L'environment rappresenta lo stato interno del calcolatore
- Quando il programma viene avviato, l'environment non è vuoto ma contiene le variabili che sono parte dello standard del linguaggio
- Alcune di queste variabili permettono di interagire con il sistema esterno



FUNZIONI

- Una funzione è un frammento di programma incapsulata in un valore
- Queste funzioni possono essere applicati per eseguire quella porzione di codice
- Nell'environment di default di ogni browser è presente la funzione `alert`, che permette di mostrare una finestra di dialogo con un messaggio

```
alert("Good morning!");
```



- L'esecuzione di una funzione è detta **invocazione** o **chiamata**
- Una funzione può essere chiamata scrivendo due parentesi alla fine del nome della variabile che la contiene
- Eventuali valori inclusi tra parentesi, chiamati **argomenti**, sono disponibili per il codice della funzione stessa



VALORI DI RITORNO

- Le funzioni possono produrre degli effetti sullo schermo (come la funzione `console.log`)
- Altre invece non hanno effetti esterni visibili ma restituiscono un valore di ritorno
- Ad esempio la funzione `Math.max` prende due numeri e restituisce il maggiore tra i due

```
console.log(Math.max(2, 4));
```

```
// → 4
```

- Una funzione che produce un valore è una espressione e, quindi, può essere utilizzata in un contesto più complesso

```
console.log(Math.min(2, 4) + 100);
```

```
// → 102
```



LA FUNZIONE `console.log`

- Questa funzione è disponibile in molti browser moderni
- Permette di mostrare a video il valore di espressioni
- Nei browser, il testo prodotto dalla funzione è mostrato all'interno della **Javascript console**
- Di solito questa parte del browser è nascosta.
 - In molti sistemi si apre premendo il tasto **F12**
 - Nei sistemi basati su Mac Os, si apre con **Command-Option-I**
 - Altrimenti si può cercare nel menu del browser una voce del tipo “web console” o “developer tools”



FUNZIONI `prompt` E `confirm`

- Queste due funzioni permettono di interagire con l'utente tramite finestre di dialogo
 - `confirm` permette una scelta tra “Ok” o “Cancel”: ritorna il valore `true` se l'utente preme OK. Altrimenti ritorna `false`
 - `confirm` solitamente è usato (e funziona) nelle pagine web
 - `prompt` può essere utilizzata per chiedere una domanda “aperta” all'utente. La funzione ritorna il valore inserito dall'utente
 - Replit.com accetta il `prompt` solo se preceduta dall'istruzione

```
const prompt = require('prompt-sync')();
```
- Queste due funzioni non sono molto utilizzate nelle applicazioni più diffuse
- Possono essere utili in un contesto didattico



ESERCIZIO

- Scrivere un programma che riceva in input la lunghezza del lato di un quadrato e calcoli il suo perimetro e la sua area



ESERCIZIO: AREA E PERIMETRO DEL QUADRATO

```
var a = prompt("Dammi la lunghezza del lato");  
  
var area = a * a;  
var perimetro = a * 4;  
  
console.log("L'area del quadrato è " + area);  
console.log("Il perimetro del quadrato è " + perimetro);
```



ESERCIZIO

- Scrivere un programma che riceva in input la lunghezza dei lati di un rettangolo e calcoli il suo perimetro e la sua area



ESERCIZIO: AREA E PERIMETRO DEL RETTANGOLO

Converte il parametro in un numero (o in NaN)

```
var a = Number(prompt("Dammi la lunghezza del lato  
maggiore"));  
var b = Number(prompt("Dammi la lunghezza del lato  
minore"));  
  
var area = a * b;  
var perimetro = (a + b) * 2;  
  
console.log("L'area del rettangolo è " + area);  
console.log("Il perimetro del rettangolo è " + perimetro);
```

CONTROLLARE L'INPUT

```
var numero = Number(prompt("Inserisci un numero"));  
if (!isNaN(numero))  
    alert("Il numero e' la radice quadrata di " +  
          numero * numero);
```



CONTROLLARE L'INPUT (2)

```
var numero = Number(prompt("Inserisci un numero"));  
if (!isNaN(numero))  
    alert("Il numero e' la radice quadrata di " +  
          numero * numero);  
else  
    alert("Hey. Perche' non hai inserito un numero?");
```



ESEMPIO DI SERIE DI CONDIZIONALI

Solitamente prompt accetta anche un testo di default

```
var num = Number(prompt("Pick a number", "0"));
```

```
if (num < 10)
```

```
    alert("Small");
```

```
else if (num < 100)
```

```
    alert("Medium");
```

```
else
```

```
    alert("Large");
```

non è così in
tutti gli
interpreti, per
esempio non è
così con
Programiz

ESERCIZIO

- Scrivere un programma che prenda in input dall'utente due numeri e ritorni il massimo tra i due



ESERCIZIO: MASSIMO TRA DUE NUMERI

```
var a = Number(prompt("Dammi il primo numero"));
var b = Number(prompt("Dammi il primo numero"));

if(a > b)
    console.log("Il massimo è " + a);
else
    console.log("Il massimo è " + b);
```





INTRODUZIONE ALLE FUNZIONI

È POSSIBILE DEFINIRE LE PROPRIE FUNZIONI

- Una funzione serve a rappresentare un algoritmo (parametrizzato) in un dato linguaggio di programmazione
- Le funzioni sono utili per:
 - strutturare i programmi più grandi
 - ridurre le ripetizioni di codice
 - associare nomi a sotto-programmi
 - isolare sotto-programmi all'interno di programmi più grandi
 - focalizzarsi su un problema alla volta



DICHIARAZIONE (O DEFINIZIONE) DI FUNZIONI

- Similmente alla dichiarazione di variabili, una **dichiarazione di funzione** è un comando che definisce un **identificatore a cui è associata una funzione**
- La definizione comprende un'intestazione e un blocco di comandi



DICHIARAZIONE DI FUNZIONI - ESEMPIO

/ dichiarazione della funzione */*

```
function stampa_ciao(){  
    console.log('Ciao!');  
}
```

← Intestazione della
funzione

corpo della funzione:
blocco di istruzioni (in
questo caso le parentesi
graffe sono sempre
obbligatorie!)

INVOCAZIONE O CHIAMATA DI FUNZIONI

- Una volta definita una funzione, è possibile usarla
- Chiamare una funzione vuol dire invocarne il nome ed eseguirne il corpo
- È possibile invocare una funzione in ogni punto del programma
 - Anche più volte
 - Ogni invocazione avrà un proprio stato come input



CHIAMATA DI FUNZIONI - ESEMPIO

```
/* dichiarazione della funzione */
```

```
function stampa_ciao(){  
    console.log('Ciao!');  
}
```

```
/* chiamata della funzione */
```

```
stampa_ciao();
```




L'ESECUZIONE DEL PROGRAMMA

/ dichiarazione della funzione */*

```
function stampa_ciao(){  
    console.log('Ciao!');  
}
```

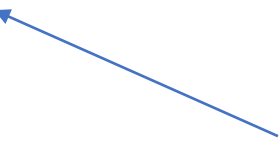
Con la dichiarazione di questa funzione non stiamo eseguendo il codice del corpo (la nostra stampa)



/ chiamata della funzione */*

```
stampa_ciao();
```

L'esecuzione dell'istruzione di stampa avviene solo quando la funzione viene chiamata



L'ESECUZIONE DEL PROGRAMMA (2)

```
/* dichiarazione della funzione */
```

```
function stampa_ciao(){  
    console.log('Ciao!');  
}
```

Fin qui stiamo solo
informando l'interprete
che esiste questa
funzione


```
/* chiamata della funzione */
```

```
stampa_ciao();
```

L'esecuzione vera e propria del
programma parte da qua; in
effetti, questo è l'inizio del
corpo del programma

LO STATO

```
/* dichiarazione della funzione */  
function stampa_ciao(){  
    console.log('Ciao!');  
}  
  
/* chiamata della funzione */  
stampa_ciao();
```



Dichiarare una funzione modifica lo stato del programma: da questo momento l'interprete sa che esiste una funzione `stampa_ciao`. Ogni volta che questo nome verrà invocato, sappiamo a cosa ci riferiamo

PARAMETRI DI UNA FUNZIONE

- Quando invochiamo una funzione dobbiamo indicare a quali dati applicare la funzione, cioè i valori dei parametri
 - Es:
 - `console.log('ciao!');`
 - `Math.floor(2.5);`
- Una funzione può non aver bisogno di parametri



PARAMETRI FORMALI

- L'intestazione della funzione definisce la lista dei suoi parametri formali (di solito per brevità si chiamano semplicemente parametri)
- I parametri formali sono indicati tra parentesi tonde dopo l'identificatore associato alla funzione



PARAMETRI FORMALI - ESEMPIO

identificatore della funzione

```
function stampa_somma(n,m){  
    console.log(n+m);  
}
```

parametri formali



INVOCAZIONE DI FUNZIONE E PARAMETRI

- Nella chiamata di funzione occorre **specificare un valore per tutti i parametri** della funzione stessa
- A ciascun parametro formale è assegnato un valore che dipende dall'espressione utilizzata nel punto di chiamata
- I valori che vengono assegnati a parametri formali durante l'invocazione di una funzione sono detti **parametri attuali**
- I parametri attuali servono per specificare quale valore devono assumere i parametri formali quando la funzione viene eseguita
 - **il valore dei parametri attuali determina l'esatta esecuzione del codice della funzione**



PASSAGGIO DEI PARAMETRI

- L'associazione tra parametri formali e i valori dei parametri attuali avviene secondo il sistema posizionale
 - ad ogni parametro formale è associato il valore del parametro attuale in posizione corrispondente nella lista
 - il primo parametro attuale è legato al primo parametro formale
 - il secondo parametro attuale è legato al secondo parametro formale
 - e così via
- Questa associazione avviene al momento dell'esecuzione



PASSAGGIO DEI PARAMETRI - ESEMPIO

```
// dichiarazione funzioni  
function stampa_somma(n,m){  
    console.log(n+m);  
}  
// corpo del programma  
stampa_somma(2,3);  
stampa_somma(19,25);
```



PASSAGGIO DEI PARAMETRI - ESEMPIO

// dichiarazione funzioni

```
function stampa_somma(n,m){  
    console.log(n+m);  
}
```

Viene definita una funzione `stampa_somma`, che prende due parametri (interi)

```
}
```

// corpo del programma

```
stampa_somma(2,3);
```

```
stampa_somma(19,25);
```

Viene invocata la funzione `stampa_somma`, a cui vengono passati i valori 2 e 3 (parametri attuali)
→ viene stampato 5

Viene invocata la funzione `stampa_somma`, a cui vengono passati i valori 19 e 25 (parametri attuali)
→ viene stampato 44

PASSAGGIO DEI PARAMETRI - ESEMPIO

// dichiarazione funzioni

```
function stampa_somma(n,m){
```

dichiarazione

```
    console.log(n+m);
```

```
}
```

// corpo del programma

```
stampa_somma(2,3);
```

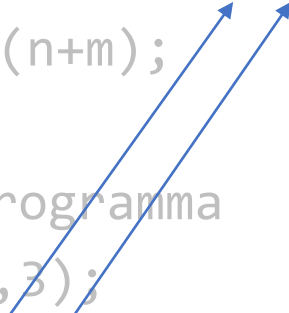
invocazione

```
stampa_somma(19,25);
```



PASSAGGIO DEI PARAMETRI - ESEMPIO

```
// dichiarazione funzioni  
function stampa_somma(n,m){  
    console.log(n+m);  
}  
// corpo del programma  
stampa_somma(2,3);  
stampa_somma(19,25);
```

Two blue arrows originate from the parameter pairs (2,3) and (19,25) in the function calls and point towards the parameters n and m in the function definition, illustrating the flow of data during function invocation.

dichiarazione

invocazione

PASSAGGIO DEI PARAMETRI (2)

- In una invocazione di funzione il numero di parametri attuali è uguale al numero dei parametri formali (indicati nella dichiarazione)
 - se il numero dei parametri attuali è minore del numero di parametri formali, ai parametri formali restanti è associato il valore «undefined»
 - se il numero dei parametri attuali è maggiore del numero dei parametri formali, i valori dei parametri attuali in eccesso vengono ignorati
- Ci sono però delle funzioni speciali, che consentono di essere parametriche anche nel numero dei parametri (es. `Math.min()`)



PASSAGGIO DEI PARAMETRI (2) - ESEMPIO

// dichiarazione funzioni

```
function saluta(nome){  
    console.log('ciao '+nome);  
}
```

// corpo del programma

```
saluta('Francesca'); ← Stampa «ciao Francesca»  
saluta('Chiara', 'Filippo'); ← Stampa «ciao Chiara»  
saluta(); ← Stampa «ciao »
```



PASSAGGIO DEI PARAMETRI (3)

- I parametri attuali possono essere anche espressioni più complesse oppure variabili
 - Il valore che viene passato alla funzione è quello contenuto nello stato al momento dell'invocazione



PASSAGGIO DEI PARAMETRI (3) - ESEMPIO

```
// dichiarazione funzioni
function stampa_somma(n,m){
    console.log(n+m);
}
// corpo del programma
var addendo1 = 2, addendo2 = 3;
stampa_somma(addendo1,addendo2);
addendo1 = 19;
addendo2 = 50;
stampa_somma(addendo1,addendo2/2);
```



VALORE DI RITORNO

- Una funzione può restituire un valore al programma principale al termine della propria esecuzione
- Per indicare il valore di ritorno si utilizza il comando `return`
 - Quando l'esecuzione incontra la parola chiave `return`, la funzione termina **immediatamente!**
- Il `return` può essere seguito dal **singolo** valore da restituire
 - La funzione può anche non restituire nessun valore
 - Se il `return` è seguito da un'espressione, la funzione termina e restituisce il valore dell'espressione
 - Il programma principale ha accesso al valore



VALORE DI RITORNO - ESEMPIO

```
// dichiarazione funzioni
function calcola_somma(n,m){
    return (n+m);
}
// corpo del programma
var somma;
calcola_somma(1,1); ← Calcola la somma ma non stiamo stampando nulla
console.log(calcola_somma(2,3)); ← Stampa 5
somma = calcola_somma(19,25);
console.log('La somma è '+somma); } ← Calcola la somma, la salva e la stampa
```

IL COMANDO RETURN

- Una funzione che restituisce un valore può essere invocata come parte di un'espressione

```
var base = 12;  
var altezza = 10;  
var perimetro = 2 * calcola_somma(base, altezza)
```



ESERCIZIO

- Definire una funzione con tre parametri (numerici): x , a e b .
- La funzione deve restituire il valore booleano `true` se x è compreso tra a e b , `false` altrimenti



ESERCIZIO: VALORE COMPRESO IN UN INTERVALLO

```
function compreso(x, a, b){  
    return ((x >=a) && (x<=b));  
}
```

```
//corpo del programma  
var numero=1, inizio=0, fine=6;  
console.log(compreso(1,0,7));  
console.log(compreso(numero,inizio,fine));  
console.log(compreso(Math.random(),inizio,0.8));
```



IMPORTANTE

- I nomi dei parametri formali e dei parametri attuali **possono** essere diversi

```
function calcola_somma(n,m){  
    return (n+m);  
}  
// corpo del programma  
var addendo1=2, addendo2=3, n=5, m=7;  
console.log(calcola_somma(addendo1,addendo2));  
console.log(calcola_somma(n,m));
```



AMBIENTE E VISIBILITÀ DELLE VARIABILI

- Un ambiente è costituito da un insieme di variabili e di funzioni
 - in un ambiente non possono esserci due variabili o due funzioni con lo stesso nome (identificatore)
- Un ambiente è modificato:
 - dalle dichiarazioni: aggiungono una variabile o una funzione all'ambiente
 - dai comandi che modificano i valori delle variabili
- Ogni variabile è visibile nei punti del programma in cui fa parte dell'ambiente
 - Anche i blocchi racchiusi tra { } definiscono ambienti diversi



VARIABILI GLOBALI E LOCALI

- Una variabile globale è visibile in tutto il programma
 - A meno che non vengano mascherate
- Una variabile locale è visibile solo in alcuni punti del programma
- Per i nostri scopi un parametro formale si comporta come una variabile locale
 - NB: il parametro formale non deve essere dichiarato dalla keyword var



VARIABILI GLOBALI E LOCALI - ESEMPIO

```
function funzione_prova(numero){  
    var nome = 'Francesca';  
    var testo = '';  
    var b = 5;  
    if (numero>0) testo = nome;  
    else testo = ''+b;  
    return testo;  
}
```

```
var a, b, x;  
a = 12;  
b = 45;  
x = 2**3;  
console.log(funzione_prova(a));
```




VARIABILI GLOBALI E LOCALI - ESEMPIO

```
function funzione_prova(numero){  
  var nome = 'Francesca';  
  var testo = '';  
  var b = 5;  
  if (numero>0) testo = nome;  
  else testo = ''+b;  
  return testo;  
}
```

variabili globali

```
var a, b, x;  
a = 12;  
b = 45;  
x = 2**3;  
console.log(funzione_prova(a));
```



VARIABILI GLOBALI E LOCALI - ESEMPIO

```
function funzione_prova(numero){  
  var nome ← 'Francesca';  
  var testo ← '';  
  var b = 5;  
  if (numero>0) testo = nome;  
  else testo = ''+b;  
  return testo;  
}  
  
var a, b, x;  
a = 12;  
b = 45;  
x = 2**3;  
console.log(funzione_prova(a));
```

variabili locali

variabili globali

SHADOWING O NAME MASKING

- Una variabile globale può essere nascosta da una variabile locale che ha lo stesso identificatore
 - una variabile nascosta non è accessibile



NAME MASKING - ESEMPIO

```
function funzione1(){  
  var x, y;  
  x = 12;  
  y = 25;  
}  
function funzione2(){  
  var x, a;  
  x = 32;  
  a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;
```



NAME MASKING - ESEMPIO

```
function funzione1(){  
  var x, y;  
  x = 12;  
  y = 25;  
}  
function funzione2(){  
  var x, a;  
  x = 32;  
  a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;
```

In funzione1 sono accessibili le variabili:

- locali: x e y
- globali: a e b

Le variabili globali x e y non sono visibili!



NAME MASKING - ESEMPIO

```
function funzione1(){  
  var x, y;  
  x = 12;  
  y = 25;  
}  
function funzione2(){  
  var x, a;  
  x = 32;  
  a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;
```

In funzione2 sono accessibili le variabili:

- locali: x e a
- globali: b e y

Le variabili globali a e x non sono visibili!



NAME MASKING - ESEMPIO

```
function funzione1(){  
  var x, y;  
  x = 12;  
  y = 25;  
}  
function funzione2(){  
  var x, a;  
  x = 32;  
  a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;
```

Nel corpo del programma sono accessibili le variabili:

- globali: a, b, x e y

Le variabili locali a funzione1 e funzione2 non sono visibili!



LO STATO E LE VARIABILI DI AMBIENTE

- Come già detto, lo stato viene generato a partire dalle funzioni e dalle variabili accessibili in un particolare punto del programma



LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(){  
    var x, a;  
    x = 32;  
    a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```



LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```

← {{funzione1, function(){...}}}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```

← {(funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
    var x, y;
    x = 12;
    y = 25;
}
function funzione2(x){
    var a;
    x = 32;
    a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
funzione1();
```


{(a,undefined), (b,undefined), (x,undefined), (y,undefined), (funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}
```

// corpo del programma

```
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```

 `{{(a,12), (b,undefined), (x,undefined), (y,undefined), (funzione1, function(){...}), (funzione2, function(x){...})}}`

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}
```

// corpo del programma

```
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```


 {(a,12), (b,45), (x,undefined), (y,undefined), (funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
    var x, y;
    x = 12;
    y = 25;
}
function funzione2(x){
    var a;
    x = 32;
    a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
function1();
```

Environment state after the last line of code:

```
{(a,12), (b,45), (x,42), (y,undefined), (funzione1, function(){...}), (funzione2, function(x){...})}
```



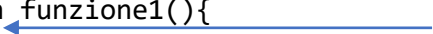
LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
    var x, y;
    x = 12;
    y = 25;
}
function funzione2(x){
    var a;
    x = 32;
    a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
function1();
```

→ {(a,12), (b,45), (x,42), (y,8), (funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
  var x, y;
  x = 12;
  y = 25;
}
function funzione2(x){
  var a;
  x = 32;
  a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
funzione1();
```



LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;                                     {(x,undefined), (y,undefined), (a,12), (b,45), (funzione1, function(x){...}), (funzione2,  
    x = 12;                                       function(){...})}  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
funzione1();
```

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
  var x, y;
  x = 12;
  y = 25;
}
function funzione2(x){
  var a;
  x = 32;
  a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
function1();
```

{(x,12), (y,undefined), (a,12), (b,45), (funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){
    var x, y;
    x = 12;
    y = 25;
}
function funzione2(x){
    var a;
    x = 32;
    a = 42;
}
// corpo del programma
var a, b, x, y;
a = 12;
b = 45;
x = 42;
y = 2**3;
function1();
```

← {(x,12), (y,25), (a,12), (b,45), (funzione1, function(){...}), (funzione2, function(x){...})}

LO STATO E LE VARIABILI DI AMBIENTE - ESEMPIO

```
function funzione1(){  
    var x, y;  
    x = 12;  
    y = 25;  
}  
function funzione2(x){  
    var a;  
    x = 32;  
    a = 42;  
}  
// corpo del programma  
var a, b, x, y;  
a = 12;  
b = 45;  
x = 42;  
y = 2**3;  
function1();
```

← {(a,12), (b,45), (x,42), (y,8), (funzione1, function(){...}), (funzione2, function(x){...})}