JavaScript: Funzioni

Sommario

- Moduli in JavaScript
- Funzioni definite dal programmatore
- Definizione di funzioni
- Regole di scope
- Ricorsione
- Ricorsione vs. Iterazione

Obiettivi

- Capire la realizzazione del concetto di modularità mediante funzioni
- Creare nuove funzioni
- Capire il passaggio di informazioni tra funzioni
- Capire la visibilità degli identificatori

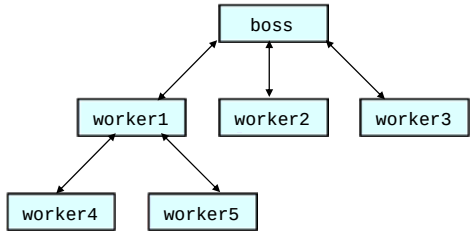
Moduli in JavaScript (1)

- Moduli in JavaScript
  - Functions
  - Methods
    - Appartengono a un oggetto
  - JavaScript comprende parecchi metodi predefiniti, che possono essere combinati con i moduli definiti dal programmatore per costruire programmi

Moduli in JavaScript (2)

- Functions
  - Sono attivate da esplicite chiamate da parte di altri moduli
  - Ricevono le informazioni necessarie attraverso i propri argomenti (parameteri)
  - Stabiliscono relazioni Client-Server
    - C'è una funzione chiamante
    - Una chiamata
    - Un valore di ritorno quando la funzione chiamata ha concluso la propria attività
    - Possono indurre vari livelli di chiamata

Moduli in JavaScript (3)



Moduli in JavaScript (4)

- Le chiamate di funzione richiedono
  - Il nome della funzione
  - Parentesi aperta
  - Lista di argomenti, ciascuno separato dagli altri da una virgola
    - Costanti, variabili o espressioni
  - Parentesi chiusa

```
total += parseFloat( inputValue );

total += parseFloat( s1 + s2 );
```


Funzioni definite dal programmatore

- Tutte le variabili dichiarate all'interno di una funzione sono dette locali
  - Non esistono all'esterno della propria funzione
- Parametri
- Favoriscono il riuso
  - È bene che siano sufficientemente brevi
  - È buona norma di programmazione usare nomi significativi

Definizione di Funzioni (1)

- Formato della definizione di una funzione

```
function function-name( parameter-list )
{
    declarations and statements
}
```


Definizione di Funzioni (2)

- Il nome della funzione può essere qualsiasi identificatore valido
- La lista di parametri indica i nomi delle variabili che saranno ricevute come argomenti della funzione
  - Il numero deve essere uguale a quello della chiamata della funzione
  - Può essere vuoto
- Dichiarazioni e istruzioni
  - Sono il corpo (body) della funzione
  - Costituiscono un blocco di codice

Definizione di Funzioni (3)

- Il controllo viene restituito dalla funzione chiamata alla chiamante
  - attraverso l'istruzione `return`
- La funzione può restituire
  - nessun valore

    `return; (si può tralasciare)`
  - oppure un valore

    `return expression;`
- Se non viene restituito alcun valore quando questo è atteso si ha un errore

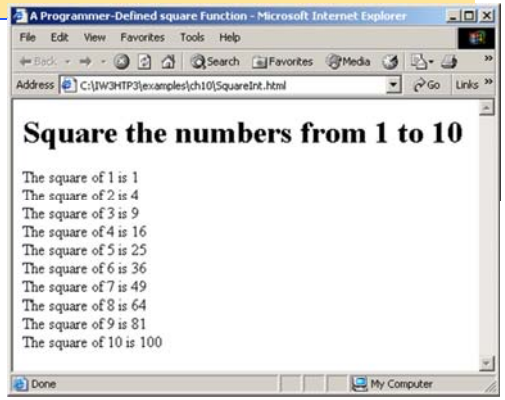

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.2: SquareInt.html -->
6  <!-- Square function           -->
7
8  <html xmlno = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>A Programmer-Defined square Function</title>
11
12       <script type = "text/javascript">
13          <!--
14          document.writeln(
15             "<h1>Square the numbers from 1 to 10</h1>" );
16
17          // square the numbers from 1 to 10
18          for ( var x = 1; x <= 10; ++x )
19             document.writeln( "The square of " + x + " is " +
20                square( x ) + "<br />" );
21
```

E' chiamata la function `square` a cui è passato il valore di x.

```
22          // The following square function's body is executed
23          // only when the function is explicitly called.
24
25          // square function definition
26          function square( y )
27          {
28             return y * y;
29          }
30          // -->
31       </script>
32
33    </head><body></body>
34 </html>
```

La variabile y assume il valore della variabile x.

L'istruzione `return` restituisce il valore di `y * y` alla funzione chiamante.




```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.3: maximum.html -->
6  <!-- Maximum function        -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Finding the Maximum of Three Values</title>
11
12       <script type = "text/javascript">
13          <!--
14          var input1 =
15             window.prompt( "Enter first number", "0" );
16          var input2 =
17             window.prompt( "Enter second number", "0" );
18          var input3 =
19             window.prompt( "Enter third number", "0" );
20
21          var value1 = parseFloat( input1 );
22          var value2 = parseFloat( input2 );
23          var value3 = parseFloat( input3 );
```

Prompt per l'inserimento dei tre interi.

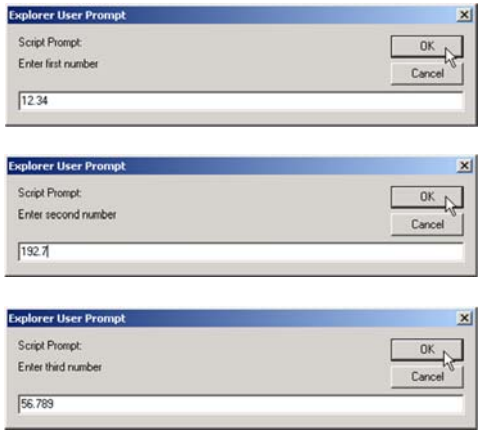
```
24
25          var maxValue = maximum( value1, value2, value3 );
26
27          document.writeln( "First number: " + value1 +
28             "<br />Second number: " + value2 +
29             "<br />Third number: " + value3 +
30             "<br />Maximum is: " + maxValue );
31
32          // maximum method definition (called from line 25)
33          function maximum( x, y, z )
34          {
35             return Math.max( x, Math.max( y, z ) );
36          }
37          // -->
38       </script>
39
40    </head>
41    <body>
42       <p>Click Refresh (or Reload) to run the script again</p>
43    </body>
44 </html>
```

Chiama la function `maximum` a cui passa il valore delle variabili `value1`, `value2` e `value3`.

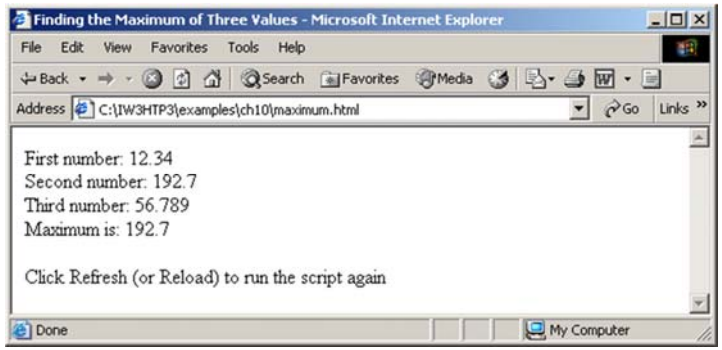
Il method `max` restituisce il maggiore di due interi.

Le variabili x, y e z assumono rispettivamente i valori delle variabili `value1`, `value2` e `value3`.

Esecuzione (1)



Esecuzione (2)



Generazione di numeri casuali

- È lo strumento che permette di introdurre elementi (pseudo) casuali
  - `Math.random`

```
var randomValue =
Math.random();
```

- Valore floating point compreso tra 0 e 1
- Math.floor
  - Arrotondamento all'intero inferiore

```
Math.floor( 1 + Math.random() * 6 )
```

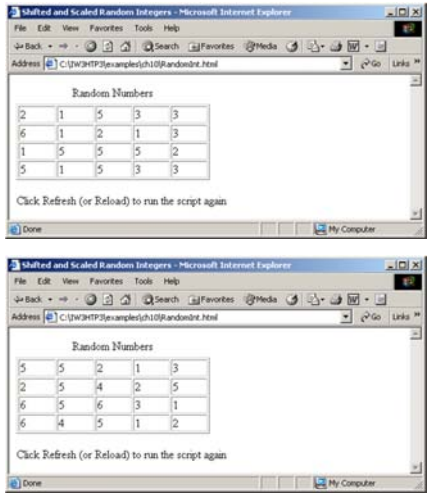


```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 10.4: RandomInt.html       -->
<!-- Demonstrating the Random method -->

<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Shifted and Scaled Random Integers</title>

      <script type = "text/javascript">
         <!--
         var value;

         document.writeln(
            "<table border = \"1\" width = \"50%\">" );
         document.writeln(
            "<caption>Random Numbers</caption><tr>" );

         for ( var i = 1; i <= 20; i++ ) {
            value = Math.floor( 1 + Math.random() * 6 );
            document.writeln( "<td>" + value + "</td>" );

            // write end and start <tr> tags when
            // i is a multiple of 5 and not 20
            if ( i % 5 == 0 && i != 20 )
               document.writeln( "</tr><tr>" );
         }

         document.writeln( "</tr></table>" );
         // -->
      </script>

   </head>
   <body>
      <p>Click Refresh (or Reload) to run the script again</p>
   </body>
</html>
```

Il `for` crea 20 celle (4 righe x 5 colonne).

Ogni cella è valorizzata con un numero casuale generato dal method `random`.

Esecuzione




```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.5: RollDie.html -->
6  <!-- Rolling a Six-Sided Die -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Roll a Six-Sided Die 6000 Times</title>
11
12       <script type = "text/javascript">
13          <!--
14          var frequency1 = 0, frequency2 = 0,
15              frequency3 = 0, frequency4 = 0,
16              frequency5 = 0, frequency6 = 0, face;
17
18          // summarize results
19          for ( var roll = 1; roll <= 6000; ++roll ) {
20             face = Math.floor( 1 + Math.random() * 6 );
21
```

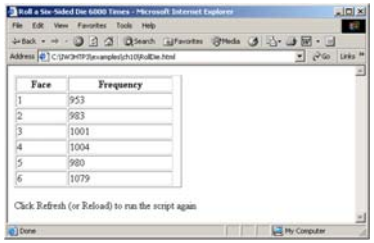
Usa il method `random` per generare un numero casuale tra `1` e `6`.

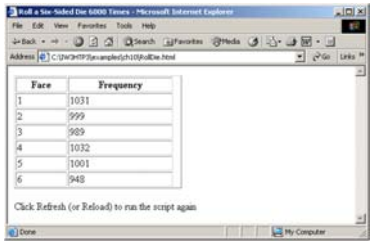
```
22             switch ( face ) {
23                case 1:
24                   ++frequency1;
25                   break;
26                case 2:
27                   ++frequency2;
28                   break;
29                case 3:
30                   ++frequency3;
31                   break;
32                case 4:
33                   ++frequency4;
34                   break;
35                case 5:
36                   ++frequency5;
37                   break;
38                case 6:
39                   ++frequency6;
40                   break;
41             }
42          }
43
```

```
44          document.writeln( "<table border = \"1\"" +
45             "width = \"50%\">" );
46          document.writeln( "<thead><th>Face</th>" +
47             "<th>Frequency<th></thead>" );
48          document.writeln( "<tbody><tr><td>1</td><td>" +
49             frequency1 + "</td></tr>" );
50          document.writeln( "<tr><td>2</td><td>" + frequency2 +
51             "</td></tr>" );
52          document.writeln( "<tr><td>3</td><td>" + frequency3 +
53             "</td></tr>" );
54          document.writeln( "<tr><td>4</td><td>" + frequency4 +
55             "</td></tr>" );
56          document.writeln( "<tr><td>5</td><td>" + frequency5 +
57             "</td></tr>" );
58          document.writeln( "<tr><td>6</td><td>" + frequency6 +
59             "</td></tr></tbody></table>" );
60          // -->
61       </script>
62
63    </head>
64    <body>
65       <p>Click Refresh (or Reload) to run the script again</p>
66    </body>
67 </html>
```

Il risultato dei lanci del dado è riportato in una tabella.

Esecuzione





Esempio: Game of Chance (1)

- Craps: gioco con 2 dadi il cui obiettivo è di ottenere il valore 2
  - Click **Roll Dice** (“lancia i dadi”)
  - I campi testuali mostrano i valori ottenuti dal lancio dei dadi e il punto ottenuto
  - La status bar mostra i risultati

Esempio: Game of Chance (2)

- Usa form XHTML
  - Permette più input per volta
  - Attributo `action` vuoto
  - L’attributo `name` permette agli script di interagire con il form
- Manipolazione di eventi programmazione event-driven
  - Assegna una function a un evento
  - `Onclick`

Esempio: Game of Chance (3)

- Costante
  - Variabile il cui valore non è mai modificato durante l’esecuzione del programma

Esempio: Game of Chance (4)

- Modifica di proprietà
  - Si accede al valore di una proprietà con la notazione dot (.)


```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig. 10.6: Craps.html -->
6  <!-- Craps Program         -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Program that Simulates the Game of Craps</title>
11
12       <script type = "text/javascript">
13          <!--
14          // variables used to test the state of the game
15          var WON = 0, LOST = 1, CONTINUE_ROLLING = 2;
16
17          // other variables used in program
18          var firstRoll = true,            // true if first roll
19              sumOfDice = 0,               // sum of the dice
20              myPoint = 0, // point if no win/loss on first roll
21              gameStatus = CONTINUE_ROLLING; // game not over yet
22
```

```
23          // process one roll of the dice
24          function play()
25          {
26             if ( firstRoll ) {          // first roll of the dice
27                sumOfDice = rollDice();
28
29                switch ( sumOfDice ) {
30                   case 7: case 11:          // win on first roll
31                      gameStatus = WON;
32                      // clear point field
33                      document.craps.point.value = "";
34                      break;
35                   case 2: case 3: case 12: // lose on first roll
36                      gameStatus = LOST;
37                      // clear point field
38                      document.craps.point.value = "";
39                      break;
40                   default:                 // remember point
41                      gameStatus = CONTINUE_ROLLING;
42                      myPoint = sumOfDice;
43                      document.craps.point.value = myPoint;
44                      firstRoll = false;
45                }
46             }
```

If the value of `firstRoll` is `true`, then function `rollDice` is called.

If function `rollDice` returns a value of `7` or `11`, the player wins and the `break` statement causes program control proceeds to the first line after the `switch` structure.

If function `rollDice` returns a `2`, `3` or `12`, the player loses and the `break` statement causes control to proceed to first line after the switch structure.

```
47             else {
48                sumOfDice = rollDice();
49
50                if ( sumOfDice == myPoint ) // win by making point
51                   gameStatus = WON;
52                else
53                   if ( sumOfDice == 7 )    // lose by rolling 7
54                      gameStatus = LOST;
55             }
56
57             if ( gameStatus == CONTINUE_ROLLING )
58                window.status = "Roll again";
59             else {
60                if ( gameStatus == WON )
61                   window.status = "Player wins. " +
62                      "Click Roll Dice to play again.";
63                else
64                   window.status = "Player loses. " +
65                      "Click Roll Dice to play again.";
66
67                firstRoll = true;
68             }
69          }
70
```

If the value of `firstRoll` is `false`, function `rollDice` is called to see if the point has been reached.

If the values returned by function `rollDice` equals `7`, the player loses.

If the value returned by function `rollDice` equals the value of variable `myPoint`, the player wins because the point has been reached.

`window` method `status` displays a message in the status bar of the browser.

```
71         // roll the dice
72         function rollDice()
73         {
74            var die1, die2, workSum;
75
76            die1 = Math.floor( 1 + Math.random() * 6 );
77            die2 = Math.floor( 1 + Math.random() * 6 );
78            workSum = die1 + die2;
79
80            document.craps.firstDie.value = die1;
81            document.craps.secondDie.value = die2;
82            document.craps.sum.value = workSum;
83
84            return workSum;
85         }
86         // -->
87      </script>
88
89   </head>
```

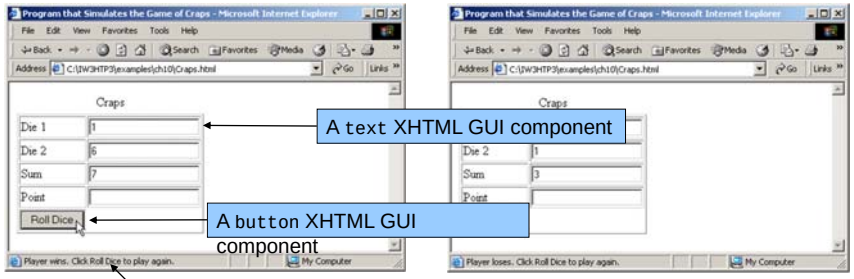
Function `rollDice` is called to simulate the rolling of two dice on the craps table.

Methods `random` and `floor` are used to generate the values for the two dice.

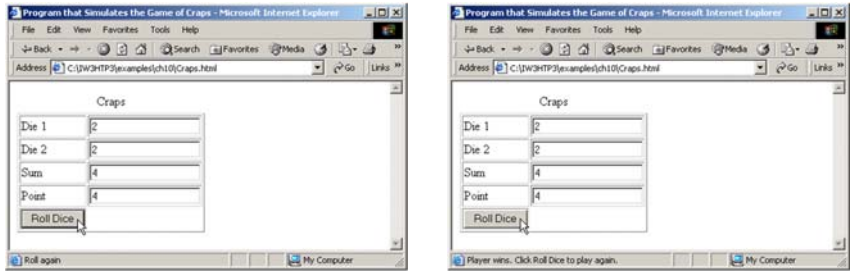
Referencing the names of form elements in the XHTML document, the values of the dice are placed in their respective form fields.

```
90   <body>
91      <form name = "craps" action = "">
92         <table border = "1">
93         <caption>Craps</caption>
94         <tr><td>Die 1</td>
95            <td><input name = "firstDie" type = "text" />
96            </td></tr>
97         <tr><td>Die 2</td>
98            <td><input name = "secondDie" type = "text" />
99            </td></tr>
100        <tr><td>Sum</td>
101           <td><input name = "sum" type = "text" />
102           </td></tr>
103        <tr><td>Point</td>
104           <td><input name = "point" type = "text" />
105           </td></tr>
106        <tr><td><input type = "button" value = "Roll Dice"
107           onclick = "play()" /></td></tr>
108        </table>
109     </form>
110  </body>
111 </html>
```

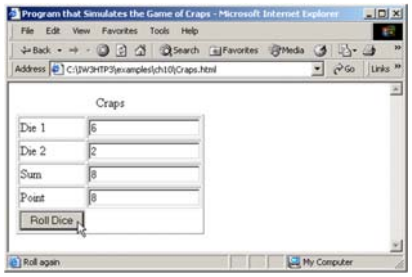
Esempio: Game of Chance Esecuzione (1)

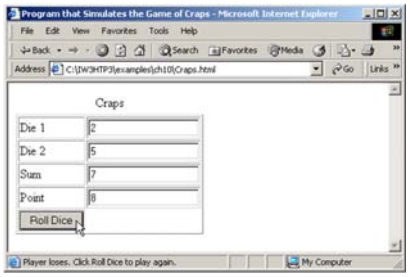


Esempio: Game of Chance Esecuzione (2)



Esempio: Game of Chance Esecuzione (3)





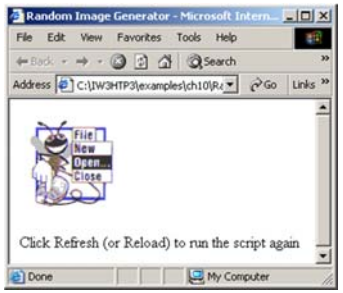
Altro Esempio: Generatore Casuale di Immagini

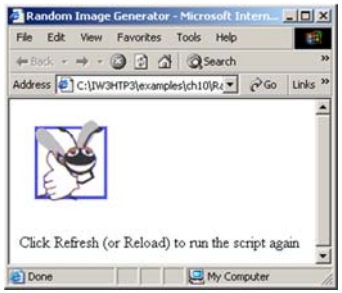
- Selezione casuale di un immagine
  - Le immagini hanno come nomi valori interi (`1.gif`, `2.gif`, …, `7.gif`)
  - Sono generati numeri casuali in un intervallo appropriato
  - È aggiornata la proprietà `src`


```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 10.7: RandomPicture.html     -->
6  <!-- Randomly displays one of 7 images -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Random Image Generator</title>
11
12       <script type = "text/javascript">
13          <!--
14          document.write ( "<img src = \"" +
15             Math.floor( 1 + Math.random() * 7 ) +
16             ".gif\" width = \"105\" height = \"100\" />" );
17          // -->
18       </script>
19
20    </head>
21
22    <body>
23       <p>Click Refresh (or Reload) to run the script again</p>
24    </body>
25 </html>
```

Inserting a random number into the image's `src` property with `document.write` and `Math.random`

Altro Esempio: Generatore Casuale di Immagini Esecuzione





Regole di Visibilità (1)

- Per Scope (Visibilità) si intende
  - Quella parte di un programma in cui un identificatore (di varibile, di costante, di modulo, …) può essere refernziato
- All'interno di una funzione lo scope è locale
  - Gli identificatori esistono solo tra le parentesi graffe aperta e chiusa
  - Le variabili locali nascondono le variabili globali

Regole di Visibilità (2)

- Dimostrazione dello Scope
  - La variabile globale `x` è inizializzata a `1`
  - `start` inizializza la variabile locale `x` a `5`
  - `functionA` inizializza la variabile locale `x` a `25`
  - `functionB` non ha la variabile locale `x`


```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.8: scoping.html    -->
6  <!-- Local and Global Variables -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>A Scoping Example</title>
11
12       <script type = "text/javascript">
13          <!--
14          var x = 1;      // global variable
15
16          function start()
17          {
18             var x = 5;   // variable local to function start
19
20             document.writeln( "local x in start is " + x );
21
22             functionA(); // functionA has local x
23             functionB(); // functionB uses global variable x
24             functionA(); // functionA reinitializes local x
25             functionB(); // global variable x retains its value
```

La variabile `x` è inizializzata a `1`.

La function `start` cambia il valore di `x` a `5`.

```
26
27             document.writeln(
28                "<p>local x in start is " + x + "</p>" );
29          }
30
31          function functionA()
32          {
33             var x = 25;  // initialized each time
34                          // functionA is called
35
36             document.writeln( "<p>local x in functionA is " +
37                               x + " after entering functionA" );
38             ++x;
39             document.writeln( "<br />local x in functionA is " +
40                x + " before exiting functionA" + "</p>" );
41          }
42
```

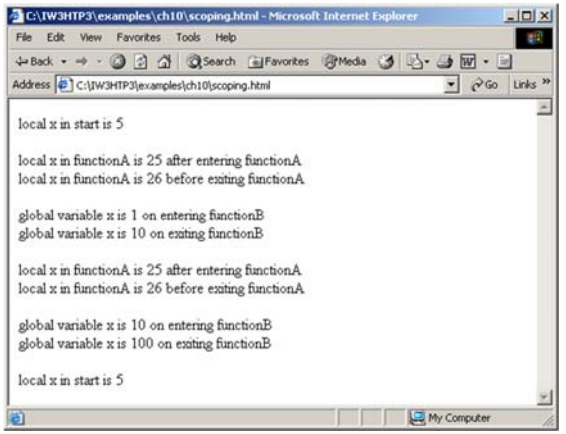
La function `functionA` cambia il valore di `x` a `25`.

Si incrementa il valore di `x`.

```
43         function functionB()
44         {
45            document.writeln( "<p>global variable x is " + x +
46               " on entering functionB" );
47            x *= 10;
48            document.writeln( "<br />global variable x is " +
49               x + " on exiting functionB"  + "</p>" );
50         }
51         // -->
52      </script>
53
54   </head>
55   <body onload = "start()"></body>
56 </html>
```

La function `functionB` moltiplica il valore di `x` per `10`.

Esecuzione



Funzioni Globali (1)

- Oggetto `Global`
  - Sempre disponibile
  - 7 metodi
  - Non è necessario riferire esplicitamente `Global` prima della chiamata al metodo

Funzioni Globali (2)

Global function	Description
`escape`	This function takes a string argument and returns a string in which all spaces, punctuation, accent characters and any other character that is not in the ASCII character set (see Appendix D, ASCII Character Set) are encoded in a hexadecimal format (see Appendix E, Number Systems) that can be represented on all platforms.
`eval`	This function takes a string argument representing JavaScript code to execute. The JavaScript interpreter evaluates the code and executes it when the `eval` function is called. This function allows JavaScript code to be stored as strings and executed dynamically.
`isFinite`	This function takes a numeric argument and returns `true` if the value of the argument is not `NaN`, `Number.POSITIVE_INFINITY` or `Number.NEGATIVE_INFINITY`; otherwise, the function returns `false`.
`isNaN`	This function takes a numeric argument and returns `true` if the value of the argument is not a number; otherwise, it returns `false`. The function is commonly used with the return value of `parseInt` or `parseFloat` to determine whether the result is a proper numeric value.

Fig. 10.9 JavaScript global functions.

Funzioni Globali (3)

Global function	Description
`parseFloat`	This function takes a string argument and attempts to convert the beginning of the string into a floating-point value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (e.g., `parseFloat( "abc123.45" )` returns `NaN`, and `parseFloat( "123.45abc" )` returns the value `123.45`).
`parseInt`	This function takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (e.g., `parseInt( "abc123" )` returns `NaN`, and `parseInt( "123abc" )` returns the integer value `123`). This function takes an optional second argument, from 2 to 36, specifying the **radix** (or **base**) of the number. Base 2 indicates that the first argument string is in **binary** format, base 8 indicates that the first argument string is in **octal** format and base 16 indicates that the first argument string is in **hexadecimal** format. See see Appendex E, Number Systems, for more information on binary, octal and hexadecimal numbers.
`unescape`	This function takes a string as its argument and returns a string in which all characters previously encoded with `escape` are decoded.

Fig. 10.9 JavaScript global functions.

Ricorsione (1)

- Una funzione ricorsiva chiama sé stessa
  - Passo di ricorsione o chiamata ricorsiva
  - Parte dell'istruzione di `return`
- Deve esistere un caso di base che permette di interrompere la ricorsione
  - È il caso più semplice del problema
  - Restituisce un valore anziché chiamare la funzione
- Ogni chiamata ricorsiva semplifica l'input

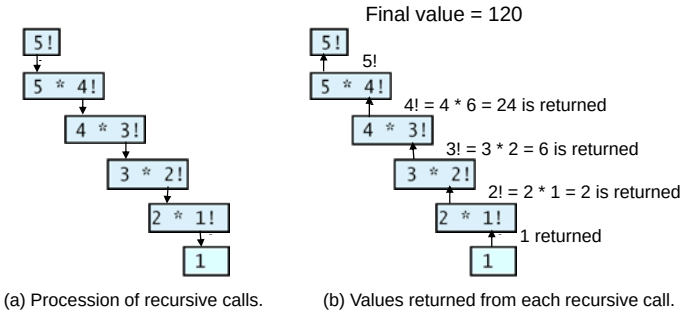
Ricorsione (2)

- Fattoriale
  - $n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$
- Iterative approach:

```
var factorial = 1;

for ( var counter = number; counter >= 1; --
counter )
   factorial *= counter;
```

  - Ogni fattore è minore di 1 rispetto al precedente
    - Si ferma a 1: caso base
    - Candidato ideale per una soluzione ricorsiva

Ricorsione (3)



(a) Procession of recursive calls. (b) Values returned from each recursive call.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.11: FactorialTest.html -->
6  <!-- Recursive factorial example    -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Recursive Factorial Function</title>
11
12       <script language = "javascript">
13          document.writeln( "<h1>Factorials of 1 to 10</h1>" );
14
15             "<table border = '1' width = '100%'>" );
16
17          for ( var i = 0; i <= 10; i++ )
18             document.writeln( "<tr><td>" + i + "!</td><td>" +
19                         factorial( i ) + "</td></tr>" );
20
21          document.writeln( "</table>" );
22
```

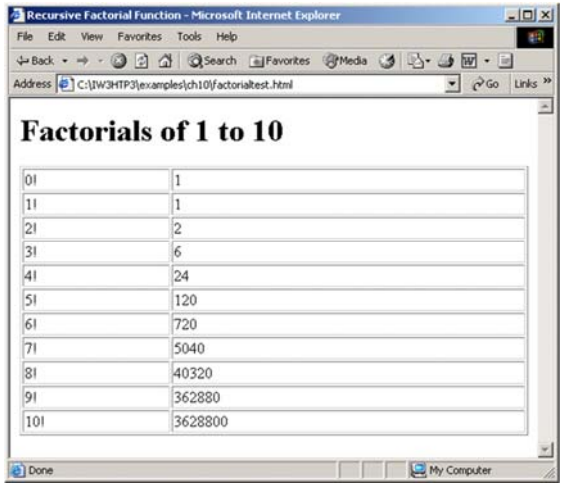
Chiamata alla function `factorial` a cui è passato il valore di `i`.


```
23          // Recursive definition of function factorial
24          function factorial( number )
25          {
26             if ( number <= 1 )  // base case
27                return 1;
28             else
29                return number * factorial( number - 1 );
30          }
31       </script>
32    </head><body></body>
33 </html>
```

La variabile `number` assume il valore della variabile `i`.

Chiamata alla function `factorial` a cui è passato il valore attuale di `number` meno `1`.

Esecuzione



Ricorsione vs. Iterazione (1)

- Iterazione
  - Il risultato è raggiunto attraverso l'uso esplicito di strutture di ripetizione
  - Termina quando fallisce la condizione che governa il loop
  - Spesso è più veloce della ricorsione

Ricorsione vs. Iterazione (2)

- Ricorsione
  - Continue chiamate alla funzione da parte della stessa funzione
  - Termina quando si raggiunge il caso base
  - Lentezza indotta dall'overhead di chiamate a funzione
    - Ogni chiamata genera una nuova copia delle variabili locali
  - Più facile da leggere e debuggare per problemi con soluzioni intrinsecamente ricorsive