Pagine dinamiche lato client: Javascript

di Roberta Molinari

Linguaggio di scripting: linguaggio interpretato destinato in genere a compiti di automazione del sistema operativo (*batch*) o delle applicazioni (*macro*), o a essere usato nella programmazione web all'interno delle pagine web

Storia e caratteristiche

- È un **linguaggio di scripting, orientato agli oggetti** e **guidato dagli eventi**.
- Fa la sua comparsa nel 1995 nel browser *Navigator* 2.0 della *Netscape* come "LiveScript", ribattezzato poi JavaScript in onore dell'azienda con cui collabora per la sua realizzazione (la *Sun* Microsistems, dal 2010 acquistata dalla *Oracle*, che stava creando *Java*).
- Microsoft realizza un linguaggio simile nel 1996 chiamandolo Jscript: oggi abbandonato.
- Netscape e Sun standardizzarono il linguaggio nel 97
- Viene usato nella creazione di siti web: esiste **lato client** e **lato server**. I suoi principali usi:
  - posizionamento dinamico degli oggetti
  - validazione campi dei moduli
  - effetti grafici

JavaScript

Caratteristiche

- Il browser visualizza il documento HTML e **interpreta** (esegue) le eventuali istruzioni scritte in Javascript.
- **È guidato dagli eventi**: se l'utente genera un evento e nel file DHTML (Dynamic HTML) esiste del codice JavaScript associato a quell'evento (event handler), questo viene eseguito
- Fornisce i **costrutti di base** di un linguaggio di programmazione: variabili, espressioni, istruzioni, …
- Ricorda C e Java, oggi è un linguaggio completo dotato di numerose librerie con nulla da invidiare ad altri linguaggi ad alto livello

JavaScript

Cosa serve

- Un **editor di testo + motore** che interpreta JavaScript lato client (es. V8 della Google, in Chrome, Chakra di Microsoft, SpiderMonkey di Mozilla su Firefox)
  - Per debbuggare si può usare *"Analizza elemento"→Debugger* in Firefox o installare l'add-on *Firebug*

JavaScript

Come inserire uno script

1. **Inserendo direttamente nel documento HTML** in un qualunque punto dell'head (non potrà fare riferimento ad oggetti del <body> perché non sono ancora stati definiti) o del body

```
<script [language="JavaScript"]>
    codice JavaScript
</script>
```

Il browser interpreta il codice quando lo incontra e lo esegue. Se c'è un errore può:

- visualizzare il documento, ma non eseguire il codice errato
- visualizzare il documento parzialmente o in bianco perché l'esecuzione dall'alto al basso è interrotta

JavaScript

Come inserire uno script

2. Caricandolo da un **file di testo esterno** con estensione .js

```
<script [language="JavaScript"]
  src="myfile.js"></script>
```

- Lo script viene eseguito dopo aver trasferito il codice esattamente nel punto in cui è inserito il tag.
- Il file .js non deve contenere tag HTML o elementi di altri linguaggi
- È il metodo consigliato se si vuole:
  - proteggere il codice sorgente
  - riutilizzare il codice in più documenti

JavaScript
Come inserire uno script

3. **Incorporandolo all'interno dei tag HTML** come valore dei nuovi attributi che sono stati introdotti per gestire gli eventi generati dall'utente

```
<div onMouseOver ="JavaScript: codice;"
   onMouseOut ="JavaScript: codice;"
   onClick ="JavaScript: codice;">
   Testo</div>
```

o cliccando su un link

```
<a href="JavaScript: codice JavaScript;">
  Clicca qui
</a>
```

JavaScript

<noscript>

<noscript> permette di presentare un contenuto alternativo quando non viene eseguito lo script, cioè se questi sono disabilitati o il browser non gli supporta.

```
<noscript>Your browser does not support JavaScript!</noscript>
```

Il tag si può trovare sia in <head> che in <body>. Se in <head> deve contenere <link>, <style> e <meta>.

È buona pratica anche commentare tutto il contenuto del tag script affinché non venga visualizzato come testo in browser che non supportano gli script

JavaScript

<noscript>

```
<script language="javascript">
    <!--
        codiceJavaScript
    -->
</script>
<noscript>
        codice HTML
</noscript>
```

I motori di ricerca e i vecchi browser testuali si comportano in modo analogo: ignorano il tag <script>, tutti i commenti, le righe di codice JavaScript e non sanno cosa voglia dire il tag <noscript>, ma riconoscono la sintassi HTML al suo interno e quindi la leggono e la inseriscono nei propri indici o la visualizzano.

JavaScript

Regole sintattiche

- Ogni istruzione inizia in una nuova riga o dopo il **;** Pertanto l'uso del ; è obbligatorio solo se si scrivono più istruzioni sulla stessa riga
- **Commenti**

```
// ci vuole lo spazio prima
/* commento su più righe*/
```

- Gli identificatori possono iniziare con **_ $ o con una lettera** e non possono iniziare con un numero, non possono contenere spazi o caratteri di punteggiatura
- Non si possono usare le parole riservate
- È **case sensitive**

JavaScript

Tipizzazione

- **Tipizzazione**: assegnazione del tipo alla variabile
  - **Static**: durante la compilazione
  - **Dynamic**: durante l'esecuzione
    - **dinamica forte**, i valori assegnati alle variabili hanno dei tipi ben definiti
    - **dinamica debole**, le variabili possono riferirsi a valori di qualsiasi tipo, che possono cambiare dinamicamente in seguito a manipolazioni esterne.
- Javascript è debolmente tipizzato: le variabili non sono tipate, possono valere qualunque cosa, anche essere una funzione

JavaScript

Tipizazzione

- JavaScript è un linguaggio **"debolmente tipato"** quindi una variabile può cambiare tipo nel corso del suo ciclo di vita. Il **tipo** delle variabili è stabilito in fase di esecuzione (**dinamic typing)**, dipende dall'ultima operazione di assegnamento eseguita

```
x=10;
x="s";          //prima x è un numero poi stringa
```

JavaScript

typeof() o typeof

La definizione stabilisce il nome della variabile, mentre il **tipo** dipende dall'ultima assegnazione.

Per verificare il tipo della variabile in un dato momento si usa `typeof()o typeof`, che può restituire la stringa: *"string", "boolean", "number", "function", "object", "undefined".*

```
x = 10;                          //typeof(x)→ "number"
x = "a";                         //typeof(x) → "string"
var x = new String();            //typeof(x )→ "object"
typeof false;                    // → "boolean"
typeof ({name: 'John', età: 34}); //→ "object"
```

JavaScript

NaN, Infinity

- **NaN**: è un numero (typeof() restituisce number) con il significato "non è un numero", che viene assegnato quando si fa un'operazione numerica che non restituiscono un numero

```
var x = 100 / "A"; // x = NaN (Not a Number)
```

Non si verifica con x==NaN ma con **`isNaN`**`(x)`

- **Infinity**: è un numero (typeof() restituisce number) che viene assegnato quando si deve assegnare un valore superiore al massimo memorizzabile o inferiore al minimo (-Infinity)

```
var x =  2 / 0;          // Infinity
var y = -2 / 0;          // -Infinity
```

Si verifica con x**==**Infinity o con **`isFinite`**`(x)`

JavaScript

Null, Undefined

- **undefined**: è il valore di una variabile dichiarata, ma non inizializzata. Convertito in NaN

```
var x;                  //Il valore è undefined
typeof(x)               //undefined
isNaN (x)               //true
```

- Le variabili definite possono essere svuotate del loro valore impostando il loro valore a **null** (indica che il valore è sconosciuto). Sono trasformate in oggetto. Significa vuoto, sconosciuto e NON puntatore nullo.

```
y = null;               //Il valore è null
typeof(y)               // object ANCHE se SBAGLIATO

null == undefined       //true uguali come VALORE
null === undefined      //false diversi come TIPO
```

JavaScript

Variabili

- Prima di essere adoperata una variabile può essere dichiarata (non necessario, il controllo è **lasco**)

  `[var|let] nome[=valore];` //dichiarazione con eventuale inizializzazione. Se non si assegna un valore la var è "undefined"

  ```
  var foo,bool=true; //inizializzata a true solo bool
  var x=y=z=3;  //inizializzate tutte a 3, ma solo x ha var
  var a="il",b='lo'; //sono due stringhe
  ```

- Una variabile non dichiarata viene automaticamente creata al primo utilizzo ed è accessibile da qualsiasi punto di uno script (è **globale**). L'utilizzo di **`var|let`** dà la possibilità di stabilire lo **scope**

JavaScript

Variabili: hoisting

Le dichiarazioni delle variabili, ovunque si verifichino, vengono elaborate prima dell'esecuzione di qualsiasi codice. Questo significa che una variabile può essere usata prima che sia dichiarata: questo si chiama **hoisting** (sollevamento), in quanto sembra che la dichiarazione della variabile venga spostata all'inizio della funzione o del codice globale. Pertanto si raccomanda di dichiarare sempre le variabili all'inizio del loro ambito. Il valore però verrà effettivamente assegnato al raggiungimento dell'istruzione di assegnazione

```
alert(x)  //undefined
var x=3;  //corretto
alert(x) //3
```

JavaScript

Variabili: scope globale

Le variabili dichiarate con **`var/let`** possono essere:

- **globali:** dichiarate fuori da qualsiasi funzione sono accessibili da qualsiasi punto dello script, anche all'interno di funzioni
- **locali**: dichiarate all'interno di una funzione accessibili soltanto all'interno del suo corpo; occupano spazio solo per il tempo di esecuzione della funzione (quindi per essere sicuri di usare le variabili locali, conviene dichiararle esplicitamente)

JavaScript

Variabili: scope con var

Dichiarare una variabile all'interno di un blocco di codice {} con **`var`** NON crea un nuovo scope per la variabile dichiarata (come in C++), resta visibile anche al di fuori del blocco.

È possibile dichiarare due volte la stessa variabile con var all'interno dello stesso blocco

Con **`var`** la variabile avrà scope di funzione e sarà utilizzabile (nel corpo della funzione) anche nelle righe che precedono la dichiarazione stessa (hoisting)

`È preferibile usare let + strict mode`

JavaScript

Variabili: scope

```
var x = 0;  // x globale
alert(typeof z); // undefined, z non esiste ancora
function a() {
  var y = 2;    // y ha scope di funzione
  alert(x +" "+ y);    // 0 2
  b();      // chiamando b() si crea z globale
  function b() {//genera ReferenceError con strict mode)
    x = 3;  //assegna 3 alla var globale x
    y = 4;  //assegna alla y di a() non crea una var globale
    z = 5;  //crea una var globale
  }
  alert(x +" "+ y +" "+ z);  // 3 4 5
}
a();                //chiama a() e quindi anche b()
alert(x +" "+ z);      // 3 5
alert(typeof y); // undefined perchè y ha scope di funzione
```

JavaScript

Variabili: scope con let

let, definita dalle specifiche di ECMAScript 6, permette di dichiarare variabili limitandone la visibilità ad un blocco di codice, ad un assegnazione, ad un' espressione in cui è usata

Inoltre una variabile dichiarata con let non sarà soggetta all'hoisting e NON è possibile dichiarare due volte la stessa variabile con let all'interno dello stesso blocco

```
var x = 10;//anche let var;
var y;
{
    let  x= 20  //nasconde quella esterna
     y = x + 1;  //21
}
y = x + y;              //10+21
```

JavaScript
Variabili: scope con var/let

```
function varTest() {
  var x = 31;
  if (true) {
    var x = 71;  // same variable!
    alert(x);  // 71
  }
  alert(x);  // 71
}
function letTest() {
  let x = 31;
  if (true) {
    let x = 72;  // different variable
    alert(x);  // 72
  }
  alert(x);  // 31
}
varTest()
letTest()
```

JavaScript

strict mode

Per rendere obbligatorio la dichiarazione delle variabili (consigliato, per supporto al controllo della correttezza del programma, tipo non modificare variabili omonime dichiarate altrove) si può inserire all'inizio del codice la stringa:

```
"use strict";
```

- Da un punto di vista sintattico, non si tratta di una istruzione vera e propria, ma di una stringa. Questo garantisce la compatibilità con le versioni precedenti (< standard 5) del linguaggio che semplicemente la ignoreranno senza generare errori.
- Una volta abilitato lo **strict mode**, ad ogni assenza di dichiarazione di variabili verrà segnalato un errore.

JavaScript

Boolean

- È considerata **vera** qualsiasi variabile con valori diversi dai seguenti:
  - false
  - 0 and -0
  - "" and '' (empty strings)
  - null (vuoto, sconosciuto e NON puntatore nullo)
  - undefined (valore non assegnato)
  - NaN (Not a Number)
  - document.all (something you will rarely encounter)

JavaScript

Variabili: tipi

- Una variabile può essere un oggetto di **tipo primitivo**: **numerico** (solo double a 64 bit), **booleano, stringa.** Questi oggetti hanno proprietà e metodi. Le variabili sono convertite automaticamente quando si utilizzano i loro metodi o proprietà.

- Se una variabile è dichiarata con `new`, allora è un **oggetto** corrispondente al tipo primitivo

```
var x = new Number(1.2);
var y = new Boolean(true);
var s = new String("ciao");
```

`typeof()` restituisce sempre **`object`**

JavaScript

Variabili: tipi stringa

- Per le stringhe si può usare:
  - indifferentemente ' o "
  - ` Backticks (ALT+96) ci permettono di incorporare variabili ed espressioni in una stringa inserendole in $ {...}

```
let name = "John";
alert( `Hello, ${name}!` ); //Hello, John!
alert( `the result is ${1 + 2}` );
    // the result is 3
```

- NON esiste il tipo char
- Per interpretare in modo corretto i caratteri "," e \ in una stringa, usare \" o \' o \\

JavaScript

Casting implicito

- Conversione di tipo **implicita**

`X=2+"3"` x è stringa = "23", basta una stringa

`X="2"*"3"` x è un numero = 6, l'opertore converte

`X=2*"3A"` errore

JavaScript

Casting esplicito

- Conversione di tipo **esplicita**

```
VarStr=String(VarNum);
```
da numero a stringa

```
VarNum=parseInt(VarStr[,BaseNumerica]);
```
da stringa a numero senza decimale (basta che la stringa inizi con un numero),

```
VarNum=parseFloat(VarStr);
```
da stringa a float

```
VarNum=eval(VarStr);
```
cerca di convertire in numerico una qualunque espressione (tipo "2+2") se non ci riesce restituisce `NaN`

```
VarNum=Number(VarStr);//restituisce un numero
                      NON un oggetto


parseInt("39 gradi") //39
Number ("39 gradi") //NaN
```

JavaScript
Conversione tra tipi: implicita

- Da tipo →NUMBER

Tipo	Valore numerico
undefined	NaN se x è undefined e faccio x+4 ottengo NaN
null	0 se x=null e faccio x+4 ottengo 4
booleano	1 se true, 0 se false
stringa	intero, decimale, zero o NaN in base alla specifica stringa

- Da tipo →BOOLEAN

Tipo	Valore booleano
undefined	false
null	false
numero	false se 0 o NaN, true in tutti gli altri casi
stringa	false se stringa vuota, true in tutti gli altri casi

JavaScript
Conversione tra tipi: implicita

- Da tipo →STRING

Tipo	Valore stringa
undefined	"undefined"
null	"null"
booleano	"true" se true "false" se false
numero	"NaN" se NaN, "Infinity" se Infinity la stringa che rappresenta il numero negli altri casi

JavaScript
Conversione tra tipi: implicita

- Operatore +: *se almeno uno dei due operandi è una stringa, allora viene effettuata una concatenazione di stringhe, altrimenti viene eseguita una addizione.*

```
x= 10 + "3"          //x="103"
x=true + null        //x=1
```

i valori *true* e *null* vengono convertiti in numeri

- Se l'operatore è un numerico si converte in numero

```
x= 10 * "3"          //x=30
```

- Per gli operatori relazionali >, >=, < e <=: *se nessuno dei due operandi è un numero, allora viene eseguito un confronto tra stringhe, altrimenti viene eseguito un confronto tra numeri.*

JavaScript
Conversione tra tipi: implicita

- Per == e != vale: *se entrambi gli operatori sono stringhe allora viene effettuato un confronto tra stringhe, altrimenti si esegue un confronto tra numeri; unica eccezione è*

  `null == undefined`

  che è vera per definizione

- **operatori di uguaglianza e disuguaglianza stretta** (=== e!==). Questi operatori confrontano gli operandi senza effettuare alcuna conversione. Quindi *due espressioni vengono considerate uguali soltanto se sono dello stesso tipo ed rappresentano effettivamente lo stesso valore*. Nel caso di due oggetti restituisce sempre false a meno che puntino alla stessa area

JavaScript
Conversione tra tipi: esplicita

```
s= String(10);                  //s="10"
n= parseInt("10 gradi");        //n=10
n= parseInt("11 gradi",2);      //n=11₂ ovvero 3
```

//il secondo parametro rappresenta la base numerica

La funzione **`eval()`** prende come argomento una stringa e la valuta o la esegue come se fosse codice JavaScript

```
x='a';y='ab';z=(eval('x<y'));          //z=true
s1= "2+2"; z=eval(s1);                 //z=4
s2= new String("2+2"); z=eval(s2);     //z="2+2"
x=eval(s2.valueOf());                  //x=4
x=10; y=20; z=eval("x+y+40")           //z=70
```

Qual è il risultato delle espressioni seguenti?

```
"" + 1 + 0
"" - 1 + 0
true + false
6 / "3"
"2" * "3"
4 + 5 + "px"
"$" + 4 + 5
"4" - 2
"4px" - 2
7 / 0
" -9 " + 5
" -9 " - 5
null + 1
undefined + 1
```

JavaScript

Istruzioni

- I **blocchi** di istruzioni

```
{
  x=2;
  y=6;
}
```

- **Assegnazione** x=2

```
x=2;
y=(z=0);      //sia y che z varranno 0
```

- **Assegnazione condizionale**

```
x =((Cond) ? valVero : valFalso);
```

Esempio

```
x=((y>0)?2:3);  //se y>0,x=2 se no x=3
```

JavaScript

Istruzioni

- **Operatori su stringhe** (per string, null, caratteri speciali, oggetti)

  `s=s+"ciao";` concatenazione anche in forma compatta s+="ciao"

  Caratteri speciali
  - `\f` avanzamento pagina
  - `\n` inizio riga
  - `\t` tabulazione
  - `\\` back slash
  - `\u00E8` `\u` seguito dal codice Unicode (nell'esempio è)

- **Operatori polimorfi**

  `+ += == != =` se operano tra tipi diversi si ha prima una conversione e poi si esegue l'operazione. Si da precedenza ai tipi string

JavaScript

Istruzioni

- **Operatori numerici** (per int, float e bool)
  - + * / %(MOD) ++ --

```
x/=y;                 equivale x=x/y restituisce un float
x=parseInt(x/y);      equivale     x=x DIV y
Math.floor(0.7);      arrotonda per difetto →0
Math.ceil(0.2);       arrotonda per eccesso→1
Math.round(0.7);      arrotonda
Math.random();  restituisce un numero tra [0..1)
Math.sqr();
Math.min(x,y);  restituisce il minore tra i due
Math.max(x,y); restituisce il maggiore tra i due
+"2"          + unario converte in numero →2
2**3          2^3
```

Le operazioni errate sui numeri non bloccano mai l'esecuzione (assegnano NaN al risultato)

JavaScript

Istruzioni

Come in C:

- `-= *=, /= …`
- **L'assegnamento restituisce il valore assegnato**

```
let a = 1;
let b = 2;
let c = 3 - (a = b + 1);
alert( a ); // 3
alert( c ); // 0
```

JavaScript

Istruzioni

- **Operatori logici**

  || && ^ ! //OR AND XOR NOT

- **Operatori relazionali**

  <,><= ==, !=, <= (anche per stringhe)

  === , !== (ug. e dis stretta senza conversione di tipi: due espressioni vengono considerate uguali soltanto se sono dello stesso tipo e rappresentano effettivamente lo stesso valore.)

```
X=5;
X=='5';   //true
X==='5' //false
```

Attenzione!

```
NaN == undefined                    //false
null == undefined                   //true unico caso anche per
isNaN(undefined)                    //TRUE
NaN == null                                 //false
NaN === undefined === null          //false
```

JavaScript

Istruzioni condizionali

- IF

```
if (cond)
   bloccoVero;
 [else if
   bloccoFalso;]
 [else    bloccoAltro;]
```

- Case

```
switch (espressione)
{case cost1:blocco1;
   [break;]
 [case cost2:blocco2;]
   [break;]
 [default: blocco;]
 }
```

JavaScript

Cicli

- While

```
while (cond)
   Blocco;
```

- Do while (repeat che cicla per vero)

```
do
   Blocco;
while (cond);
```

- **`break;`** //esce dal blocco
- **`continue;`** //ricomincia il blocco di un ciclo

JavaScript

For

- For

```
for(esprIniz;cond;passo;)
   blocco;
```

si esegue `esprIniz`, si verifica `cond`, se è vera si esegue `blocco`. si esegue `passo` e si ricomincia verificando `cond` …

Per lavorare più comodamente con gli array JavaScript prevede due varianti del for:

- `for...in` per gli oggetti e i vettori
- `for...of` per le collezioni come: Array, String,…

JavaScript

For … in

- **`For in`** scorre le proprietà di un oggetto (in un vettore sono gli indici)

```
var quantita = [12, 34, 45, 7, 19];
var totale = 0;
for (var indice in quantita) {
  totale = totale +  quantita[indice];
}
```

Non bisogna specificare la lunghezza dell'array nè l'istruzione di modifica della condizione. JavaScript rileva che la variabile `quantita` è un array ed assegna ad ogni iterazione alla variabile `indice` il valore dell'indice corrente.

JavaScript

For … of

- **For of** scorre i valori presenti in oggetti iterabili ovvero collezioni come: Array, Map, Set, String,…

```
var quantita = [12, 34, 45, 7, 19];
var totale = 0;
for (var valore of quantita) {
    totale = totale +  valore;
}
```

- Ad ogni iterazione JavaScript assegna alla variabile valore il contenuto di ciascun elemento dell'array.
- Fa parte delle specifiche di ECMAScript 6 e potrebbe non essere disponibile in engine JavaScript meno recenti.

JavaScript

With

- With

```
with (Math){
    floor(random()*quanti)+aPartireDa;
}
```

Attenzione non usare dentro il `with (Math)` variabili con lo stesso nome delle funzioni di `Math`(es min, max,…) se no non funziona

JavaScript

Costanti

Dallo standard 6 è possibile dichiarare costanti

```
const PIGRECO=3.14;
```

Per le versioni precedenti si deve utilizzare la dichiarazione con assegnamento e ricordandosi di non modificare nelle istruzioni il valore

```
var PIGRECO=3.14;
```

Finestre di dialogo: possono essere **modali** (bloccanti) se non permettono la prosecuzione dell'esecuzione del resto del codice fino alla loro chiusura, oppure **non modali** (non bloccanti)

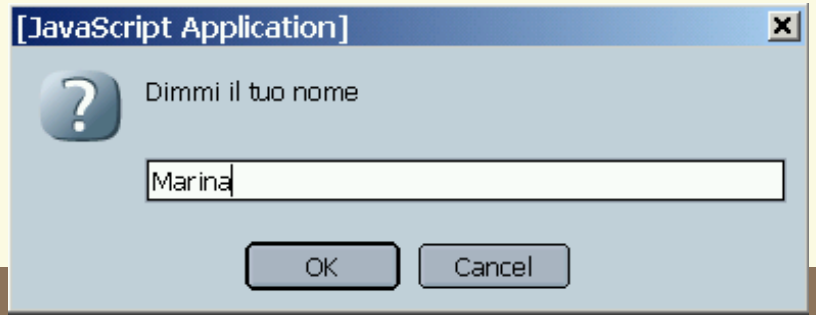
Istruzioni di Input

```
x=window.prompt("Dimmi il tuo nome","");
```

Anche solo **`prompt`**(), il secondo parametro è il valore di default. Apre una finestra modale. Restituisce:

- se si introduce qualcosa e si preme OK, restituisce sempre una stringa anche se si introducono numeri
- **`null`** se preme ANNULLA
- **`undefined`** se non si inserisce nulla e non c'è un valore di default

Si può quindi valutare var==undefined oppure ==null

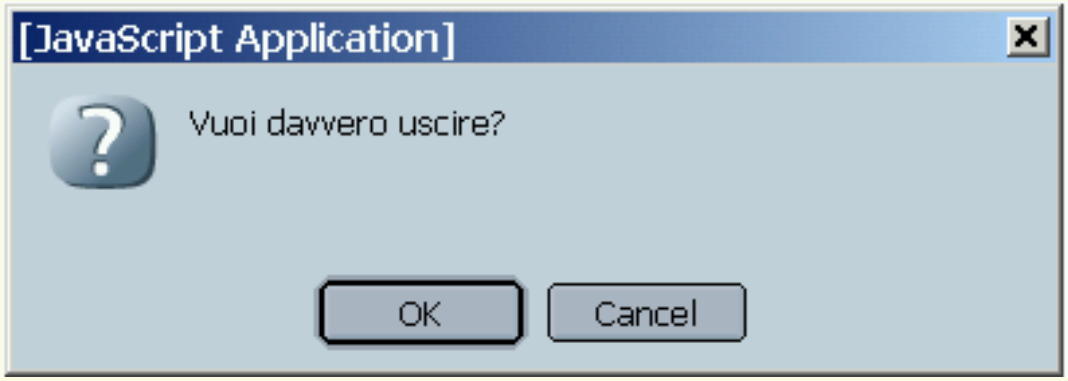


JavaScript

Istruzioni di Input

```
window.confirm("Vuoi davvero uscire?")
```

apre una finestra modale che restituisce true (OK) o false (CANCEL)



JavaScript

Istruzioni di Output

```
window.alert("Hello world");
```

apre una finestra modale che restituisce undefined

```
window.status ("Hello world");
```

viene scritto nella barra di stato

```
console.log("Hello world");
```

viene scritto a console durante il debug

JavaScript

Istruzioni di Output

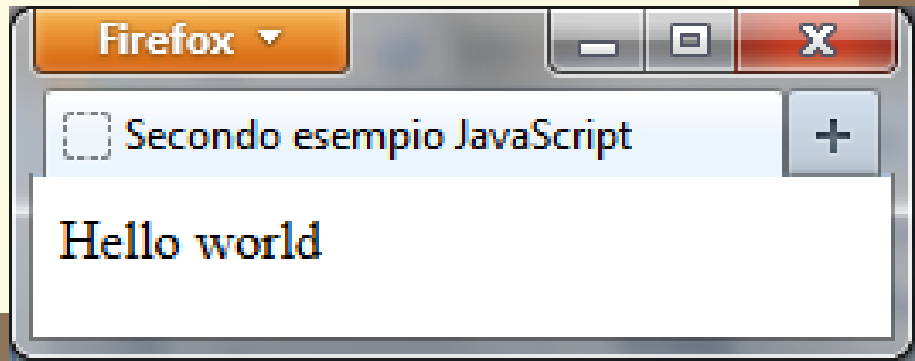
`document.write("Hello world");`

viene scritto nel flusso del documento HTML al momento dell'esecuzione

`.writeln` mette un a capo nel documento (non mette un
! Usa <pre>)

Con **`.open()`** e **`.close()`**

```
document.open();
document.write("Hello World");
document.close();
```



JavaScript

try..catch

Serve per gestire un errore. La clausola finally viene eseguita in ogni caso, anche se c'è return

```
try {
       foo()
} catch(err) {
    alert(err.message) ;
    return 10 ;
} finally {
    alert("tutto ok") ;
}
```

Per generare un errore si utilizza l'istruzione

```
throw "Errore"      //sarà assegnato a err
...
catch(err) {
    alert(err) ;   //visualizza "Errore"
```

JavaScript

Funzioni

Funzioni si possono dichiarare così (non si dichiara il tipo restituito, avrà uno scope)

```
function nome ([par1,…,parN]){
    [var local1,…]
    …
    [return (espressione);]    //anche array
}
```

Si richiamera così: `nome()`

- Le funzioni possono essere richiamate prima della loro definizione, per il meccanismo dell'hoisting
- Se definite in una funzione sono visibili solo dentro essa

JavaScript

Funzioni

- Sono tutte funzioni, le procedure non restituiscono risultato Return è facoltativo. Se non c'è restituisce undefined
- Typeof su una funzione restituisce "function"
- La chiamata può avvenire anche prima della dichiarazione (non per le anonime) e con un numero anche diverso di parametri (quelli mancanti saranno undefined e quelli in eccesso non saranno considerati)
- Normalmente si dichiarano nell'HEAD
- Il passaggio dei parametri relativi a tipi di dato primitivi avviene sempre per valore per gli altri tipi di oggetti avviene sempre per riferimento.

JavaScript
Funzioni: con valori di default

- Dal ECMAScript 6 viene introdotta la possibilità di specificare dei valori di default:

```
function somma(x = 0, y = 0) {
    var z = x + y;     return z;
}
```

Così, se al momento della chiamata non viene passato un argomento, ad esso viene assegnato il valore di default specificato, invece del valore undefined. Quindi, ad esempio, la chiamata somma() senza argomenti restituirà il valore 0 anzichè NaN.

```
somma(3) //x=3 e y=0
somma (undefined, 5) //x=0, y=5
```

JavaScript
Funzioni: con valori di default

- Senza i valori di default quando NON vengono passati dei parametri questi sono undefined

```
function myFunction(x, y) {
      if (y === undefined) {
            y = 0;
      }
      return x + y;
  }

myFuncion(3) //x=3 y è undefined
  restituisce NaN
```

JavaScript

Espressione Funzione

Ad una variabile si può assegnare un'espressione funzione (con **`let`** avrà uno scope se no è globale).

```
[const|let] a = [function] ([par1,…,parN]){
      [let local1,…]
      …
      [return (espressione);]
   }; //è consigliato mettere il ;
```

Si richiamerà così: `a()`

JavaScript

Espressione Funzione

- Ad a viene passato tutto "il codice della funzione".

```
function sayHi() {
     alert( "Hello" ); }

alert( sayHi ); //shows the function code
```

- Non possono essere richiamate prima della loro dichiarazione

```
sayHi("John"); // error!
let sayHi = function(name) {
     alert( `Hello, ${name}` ); };
```

JavaScript

Funzioni di callback

```
function ask(question, yes, no) {
     if (confirm(question)) yes()
     else no(); }
function showOk() {
     alert( "You agreed." ); }
function showCancel() {
     alert( "You canceled." ); }

ask("Do you agree?", showOk, showCancel);
```

I parametri `showOk`, `showCancel` sono dette funzioni di **callback**, in quanto passiamo una funzione come parametro e ci aspettiamo che sarà eseguita in seguito

JavaScript

Funzione anonime

```
function ask(question, yes, no) {
if (confirm(question)) yes()
else no(); }
ask( "Do you agree?",
     function(){alert("You agreed.");},
     function(){alert("You canceled.");}
     );
```

Non sono richiamabili da nessuna altra parte perchè non hanno nome

JavaScript

Funzione freccia

```
let func =
      (arg1, arg2, ...argN) => expression
```

- Si crea una funzione che ha gli argomenti dihiarati nelle tonde e restituisce il valore dell'espressione a destra della freccia=>. Corrisponde a:

```
let func =
      function(arg1, arg2, ...argN){
           return expression; };
```

JavaScript

Funzione freccia

- Se ha un solo argomento le () si possono omettere

```
let double = n => n * 2;
```

- Se non ne ha si mette ()

```
let sayHi = () => alert("Hello!");
```

- Se contiene più di un'istruzione si usano {} e l'istruzione return

```
let sum = (a, b) => {
      let result = a + b;
      return result;
      };
alert( sum(1, 2) ); // 3
```

JavaScript

Funzione freccia

Esempio di uso come funzioni di callback anonime

```
let age = prompt("What is your age?",18);
let welcome = (age < 18) ?
      () => alert('Hello') :
      () => alert("Greetings!");
welcome();
```

JavaScript

Costruttore di funzioni

Si possono definire delle funzioni attraverso il costruttore di funzioni **Function**(), da evitare

```
var myFunction = new Function("a", "b",
  "return a * b");
var x = myFunction(4, 3);
```

Che è del tutto equivalente a scrivere

```
var myFunction = function (a, b) {return
  a * b};
var x = myFunction(4, 3);
```

JavaScript
Funzioni: esempio di scope

```
foo()        //per l'hoisting
function foo(){
   function bar(){ //locale anche con var bar =
       console.log("Hello")}
   bar()
}
bar()        //non visibile

//foo1()     //non viene fatto l'hoisting
foo1= function(){
   bar1 =function(){     //globale
       console.log("Hello 1")}
   bar1()
}
foo1()
bar1()              //visibile
```

JavaScript

Funzionis self-invocking

- Una funzione può dichiararsi e contemporaneamente chiamarsi se termina con ()
- Nel caso di espressioni funzione. Non si può più chiamare con foo()

```
var foo=function () {
    var x = "Hello!!";}();
```

- Nel caso di una funzione anonima

```
(function () {
    var x = "Hello!!";    // I will invoke myself
})();
```

- Non si può fare con la dichiarazione classica

JavaScript

Funzioni: scope chain

- *Gerarchia di scope* o **scope chain**: una funzione può accedere allo scope locale, allo scope globale ed allo scope accessibile dalla funzione in cui è stata definita (funzione esterna), il quale può essere a sua volta il risultato della combinazione del proprio scope locale con lo scope della sua funzione esterna e così via.
- in JavaScript *l'accesso allo scope della sua funzione esterna è consentito anche dopo che questa ha terminato la sua esecuzione*.

JavaScript

Funzioni: scope chain

```
var saluto = "Buongiorno"; var visualizzaSaluto;
function saluta(persona) {
   var nc = persona.nome + " " + persona.cognome;
   return function() { console.log(saluto + " " + nc);
                       };
}
visualizzaSaluto =
      saluta({nome: "Mario", cognome: "Rossi"});
visualizzaSaluto();
```

- In questo caso la funzione `saluta()` non visualizza direttamente la stringa ma restituisce una funzione che assolve questo compito. Pertanto, quando la funzione restituita viene invocata, la funzione `saluta()` (la sua funzione esterna) ha terminato la sua esecuzione e quindi il suo contesto di esecuzione non esiste più. Nonostante ciò è ancora possibile accedere alla variabile `nc` presente nel suo scope locale.

JavaScript

Funzioni: scope chain

```
<p>Counting with a local variable.</p>
<button type="button" onclick="myFunction()"> Count!
</button>
<p id="demo">0</p>
<script>
var add = (function () {
    var counter = 0;
    return function () {return counter += 1;}
})();
function myFunction(){
    document.getElementById("demo").innerHTML = add();
}
</script>
```

Esempio di Javascript **Closure:** una funzione ha accesso allo scope genitore, anche dopo la chiusura della funzione genitore.

JavaScript
Funzioni: array arguments

- Si può non definire alcun argomento nella definizione di una funzione ed accedere ai valori passati in fase di chiamata tramite un array predefinito: **arguments**

  Ad esempio, possiamo sommare un numero indefinito di valori con chiamate somma(1,2) o somma(1,2,3)…

```
function somma() {
    var z = 0;     var i;
        for (i in arguments) {
        z = z + arguments[i];
    }
     return z;
}
```

JavaScript
Funzioni: rest parameter

```
function eseguiOperazione(x, ...y) {
      var z = 0;
      switch (x) {
          case "somma":
            for (i in y) {
                z = z + y[i];
            } break;
          case "moltiplica":
            for (i in y) {
                z = z * y[i];
            } break;
          default:  z = NaN;  break;
    }
    return z;
}
```

JavaScript
Funzioni: rest parameter

- L'argomento x che rappresenta il nome dell'operazione da eseguire e y preceduto da tre punti che rappresenta il resto dei valori da passare alla funzione, che saranno disponibili sotto forma di vettore
- L'approccio è simile all'array predefinito arguments, ma mentre questo cattura tutti gli argomenti della funzione, il rest parameter cattura soltanto gli argomenti in più rispetto a quelli specificati singolarmente.

JavaScript
Funzioni: spread operator

- La stessa notazione del rest parameter può essere utilizzata nelle chiamate a funzioni che prevedono diversi argomenti. In questo caso si parla di **spread operator**, cioè di un operatore che sparge i valori contenuti in un array sugli argomenti di una funzione, come nel seguente esempio:

```
var addendi = [8, 23, 19, 72, 3, 39];
somma(...addendi);
```

- La chiamata con lo spread operator è equivalente alla seguente chiamata:

```
somma(8, 23, 19, 72, 3, 39);
```

JavaScript

Oggetti

Javascript gestisce **oggetti**:

- **integrati** (o interni) :gli oggetti di base
  - Array
  - Date
  - String
  - Math
  - Number
  - Boolean
  - Function
- **riflessi dal browser o browser dipendenti** **BOM:** forniti dall'ambiente del browser (*window, document, location, status, history, navigator*)
- **riflessi dall'HTML** **DOM:** sono gli elementi del documento HTML
- **definiti dall'utente**

Oggetti integrati

di Roberta Molinari

JavaScript

Oggetti integrati

Javascript fornisce i seguenti **oggetti integrati**:

- Array
- Date
- String
- Math
- Number
- Boolean
- Function

- È sconsigliato istanziare oggetti Array, String, Number, Boolean (per es. new String()), invece se ne potranno comunque usare i metodi e gli attributi

Funzioni e proprietà globali o di sistema: non appartengono a nessun oggetto specifico, quindi si indicano senza "."

JavaScript

Number

- **Number** (sono sempre floating point a 64 bit)
  - operazioni aritmetiche +,-,*,/,% , x++, --x, +=
  - Funzioni globali
    - **`isNaN(v)`** verifica se v non è un numero
    - **`isFinite(v)`, `isInteger(v)`** verifica se v è finito, intero
    - **`parseFloat(str)`, `parseInt(str)`** Cercano all'inizio di str un numero decimale o un intero. Se non lo trovano restituiscono NaN.

      `parseFloat(1.3)`→`1.3` `parseInt(1.3)`→`1`
    - **`Number(ogg)`** Converte oggetto in un numero, se possibile. Se no ritorna NaN o =0 se non c'è ogg (es. con "")
  - Proprietà globali: **`NaN`, `±Infinity`, `undefined`** (indica che a una variabile non è stato assegnato un valore).

JavaScript
Costanti statiche Number

Costante	Valore restituito
Number.MAX_VALUE	Numero più grande che può essere rappresentato in JavaScript. Equivale a circa 1,79E+308.
Number.MIN_VALUE	Il numero più vicino a zero che può essere rappresentato in JavaScript. Equivale a circa 5,00E-324.
Number.NaN	Valore che non è un numero, ma il suo typeof è Number. Appare come risultato di operazioni che non restituiscono un numero come 0/0 Nei confronti di uguaglianza, **NaN** non corrisponde ad alcun valore, incluso se stesso. Per verificare se un valore è equivalente a **NaN**, utilizzare la funzione isNaN.
Number.NEGATIVE_INFINITY	Valore minore del massimo numero negativo che possa essere rappresentato in JavaScript. In JavaScript i valori **NEGATIVE_INFINITY** vengono visualizzati come `-infinity`.
Number.POSITIVE_INFINITY	Valore maggiore del massimo numero negativo che possa essere rappresentato in JavaScript, o risultato di 1/0 In JavaScript i valori **POSITIVE_INFINITY** vengono visualizzati come `infinity`.

JavaScript

Oggetto Math

- **Proprietà** dell'oggetto **Math**
  - **`E`** costante di Eulero
  - **`PI`** pi greco
- **Metodi**
  - **`abs(val)`** valore assoluto
  - **`round(val)`** arrotondamento
  - **`ceil(val)`** arrotondamento per eccesso
  - **`floor(val)`** arrotonda per difetto
  - **`random()`** numero casuale tra 0 e 1
  - **`pow(base, exp)`** elevamento a potenza
  - **`sqrt(val)`** radice quadrata

JavaScript

Boolean

- **Boolean** possibili valori `true, false`

```
var pagato = true;
var consegnato = false;
```

  - Operatori logici: `&&, ||, !`

JavaScript

String

- **String**

```
var nome = "Mario"; //anche 'Mario'
var empty = ""; //vuota Number("")→0
var empty2 = new String(); //vuota
var str2 = new String("Altra");
```

- Con `new String()` typeof restituisce Object
- Assegnazione con =
- Si possono usare sia " che '
- Confronto con == !=
- Funzione globale **`String(oggetto)`** converte oggetto nella stringa che rappresenta il suo valore,

JavaScript

String

- Concatenazione di stringhe con **+**

  `nome = nome + "!"` //anche nome +="!"

- Dalla versione 6 si può usare l'apice inverso ` (Alt+96 in Windows, AltGr + ' in Linux) e l'inserimento tramite ${}

```
let myPet = 'armadillo'
alert(`I own a pet ${myPet}.`)
```

JavaScript

String

- **Proprietà** degli oggetti String
  - **`length`** lunghezza della stringa
- **Metodi** degli oggetti String
  - **`charAt(pos)`** equivale a s[pos] , che non si deve fare
  - **`charCodeAt(pos)`** restituisce il codice Unicode di pos
  - **`subString(start, end)`** restituisce i caratteri ∈[start..end) (primo 0)
  - **`toUpperCase()/toLowerCase()`** restituiscono la stringa modificata, ma non modificano l'oggetto a cui sono applicati
  - **`vett=stringa.split(",")`** usa il carattere specificato come separatore per individuare gli elementi del vettore
  - **`s=str1.concat(" ", str2)`** restituisce la stringa ottenuta dalla concatenazione di str1+" "+str2

JavaScript

String

- **Metodi** degli oggetti String
  - **`indexOf/lastIndexOf(str[,pos])`** posizione della prima/ultima occorrenza della string **`str`** cercata a partire dalla posizione pos facoltativo. -1 se non la trova
  - **`search(str)`** come la precedente, ma non può specificare da dove iniziare la ricerca, invece può usare espressioni regolari da cercare. Es s.search(/mamma/i);
  - **`replace(old, new)`** restituisce la stringa ottenuta sostituendo la prima occorrenza di `old` con `new`. `old` può essere un espressione regolare con i parametri `/i`(insensitive) `/g`(sostituzione globale)

JavaScript

String encode()

- **`encodeURIComponent()`** è pensata per codificare i valori di eventuali parametri passati in un URI. Codifica una stringa lasciando inalterate cifre, lettere e i caratteri +-*/._@ e rimpiazzando tutti gli altri caratteri con la codifica esadecimale preceduta da un carattere percentuale (%).
- **`encodeURI()`** come la precedente, ma esclude dalla codifica i caratteri , /?:@&=+$#

```
var param= encodeURIComponent("Che cos'è?");
var encodedURI = encodeURI("http://www.html.it/ab.php?x=") + param;
```

- Le funzioni **`decodeURIComponent`**(), **`decodeURI`**() esegueno il procedimento contrario

JavaScript

Date

- Una variabile di tipo **Date** rappresenta un istante temporale (data ed ora) rappresentante i millisecondi passati dal 1-1-1970 (<0 per antecedenti)

```
var adesso = new Date(); //data e ora corrente
var natale2012 = new Date("Dec 12,25");
var fineAnno2013 = new Date(2013,11,31,23,59,00);
var fineAnno2014 = new Date("2014-12-31T23:59");
```

- I mesi e i giorni della settimana partono da 0
- Se aggiungo 1 ai giorni o ai mesi tiene conto degli anni bisestili e aggiorna l'anno (dicembre + 1). Lo stesso con le ore e i minuti.
- Si possono fare operazioni aritmetiche (considerando il risultato in millisec) e confronti <,<=,…

JavaScript

Date

- **Metodi** degli oggetti Date
  - `getXXX ()` restituisce il valore della caratteristica XXX della data (es. getFullYear()).
  - `setXXX (val)` imposta il valore della caratteristica XXX della data (es. setFullYear(2013,1,1));
  - `toString()` restituisce la data come stringa formattata

Caratteristica	Significato
Date	Giorno del mese
Day	Giorno della settimana
FullYear	Anno
Minutes	Minuti

Caratteristica	Significato
Hours	Ore
Month	Mese
Seconds	Secondi
Time	millsec dal 1-1-70

JavaScript

Array

- È un **insieme di elementi di tipo arbitrario** con primo indice 0 (array *denso*). È un particolare oggetto i cui campi sono reperibili tramite indice (typeof restituisce object)

```
var v = [];
var a = [1, 2, 3];
var b = [10, "ciao", True];
```

- Si può usare il costruttore Array, ma si preferisce il primo modo

```
var v = new Array(); //vuoto meglio var v=[];
var w = new Array("Qui", "Quo", "Qua");
var c= new Array(3); //array di 3 celle undef.
var d= new Array(3,4); //array di 2 celle con 3 e 4
```

JavaScript

Array

- Non è necessario specificare la dimensione
- typeof restituisce object, per cui per sapere se è un , array, dalla versione 5 si può utilizzare
  - `Array.isArray(vet);`
  - `vet istanceof Array`
- Si può visualizzare in toto

```
alert(v)  //compare una lista separata da ,
```

- Se usati come parametri sono passati per referenza e come par formali si dichiara solo il nome

```
function f(V,N)
```

- Possono essere restituiti come valore di una funzione da assegnare ad un vettore, anche se non ha senso essendo oggetti fare v1=v2 (diventano puntatori alla stessa area).

JavaScript

Array

- Se si assegna un valore ad un elemento non presente questo viene creato

```
var v = [];
v[0] = 1;
v[3] = 5; //1 e 2 sono undefined
```

- Se si accede al valore di un elemento non presente si ottiene un valore indefinito

```
var v = [];
var a = v[0]; //undefined
```

- Per aggiungere elementi in fondo

```
newLeng = v.push(6);//equivale a v[v.length]=6;
```

JavaScript

Array

- **Proprietà** degli oggetti Array
  - **`length`** lunghezza del vettore (è di RW), se si cambia il suo valore si cambia la dimensione del vettore
- **Metodi**
  - **`sort(fConf)`** per ordinare il vettore secondo la funzione **`fConf`** (invia i valori a **`fConf`** e ordina i valori in base al valore restituito <0,=0,>0). Se non c'è **`fConf`** è ordinato alfabeticam. (secondo ASCII 1< A< a) Per ordinare in base ai numeri
    `v.sort(function(a, b){return a - b})`
  - **`reverse()`** inverte l'ordine degli elementi

JavaScript

Array

- **Metodi**
  - **`join(c)`** unisce i valori in una stringa concatenata con il carattere c
  - **`primoEl=v.shift()`** elimina e restituisce il primo elemento
  - **`newLeng=v.unShift(newPrimoEl)`** aggiunge un nuovo elemento al posto 0 e restituisce la nuova lunghezza
  - **`newLeng=push(val)`** e **`val=pop()`** come i precedenti, ma con l'ultimo elemento
  - **`delete vett[0]`** assegna undefined al primo elemento (la lunghezza non viene modificata)

JavaScript

Array

- **Metodi**
- **`splice(pos, n[, el1,…,eln])`** a partire dalla posizione **`pos`**, elimina **`n`** elementi e inserisce **`el1`**,...

```
var fruits = ["Banana", "Orange", "Apple"];
fruits.splice(1, 0, "Lemon", "Kiwi");
```

Senza gli ultimi parametri elimina solo gli elementi

```
fruits.splice(0, 1); //elimina il primo elemento
```

- **`slice(da, a)`** crea un nuovo array prelevando gli elementi dalla posizione **`da`** fino ad **`a`** esclusa (se non c'è a fino all'ultimo elemento)

```
x=fruits.slice(1, 3);
y=fruits.slice(2);
```

JavaScript

Array

- **Metodi**
- **`concat(arr1 [, arr2,…])`** concatena gli array

```
var myChildren = myGirls.concat(myBoys);
```

- **`forEach(callback)`** richiama la funzione di callback passandogli come parametro tutti gli elementi dell'array (può avere 1 o 3 parametri formali)

```
var txt = "";
var numbers = [45, 4, 9, 16, 25];
numbers.forEach(myFunction);
function myFunction(value, index, array) {
  txt = txt + value + "<br>"; }
```

oppure

```
function myFunction(value) {
  txt = txt + value + "<br>"; }
```

JavaScript

Array associativi

- In alcuni linguaggi, si possono definire array del tipo *chiave-valore* (hashmap) o **array associativi**. Javascript NON li gestisce, si può fare, perché vengono definiti degli oggetti e NON array (lenght=0 sempre)
- Chiave con ", campo indifferente ' "

```
var array = new Array();
array["js"] = 'JavaScript';
//ridefinito come un'oggetto standard
array["css"]='Cascading Style Sheet';
```

- Non si possono visualizzare in toto con una alert

JavaScript

Array associativi

- JavaScript gestisce i suoi elementi come proprietà di un oggetto, pertanto la proprietà length è sempre=0, non si può usare il for of e si può usare la notazione per le proprietà (es. **`array.js`**)

```
for (i in array) {
    document.write(i + '=' + array[i])
}
```

- Si possono serializzare (trasformare in stringa) con `stringify(vet)`
- Per aggiungere campi e valore

```
vet.push({"campo": val})
```

JavaScript

Array multidimensionali

- Non si possono definire array bidimensionali, ma un array può essere un elemento di un array. Questo consente di definire **array multidimensionali o matrici**.

```
mat=new Array(N);
for (k=0;k<N;k++) mat[k]= new Array(M);
```

Oppure

```
var mat = [[4,3,1],[8,2,7],[8,6, 1]];
```

- Per accedere al numero 8 indicheremo le sue coordinate nel seguente modo :

```
var otto = mat[1][0]; //[riga][colonna]
```

JavaScript

Oggetti: ToString

Tutti gli oggetti JavaScript sono basati su Object e condividono alcuni metodi: **toString()** e **valueOf()**.

- `toString()` restituisce una versione in stringa dell'oggetto:

```
var x = new Object(32);
x.toString(); //restituisce "32"
```

- Nel caso di oggetto non riconducibile ad un tipo di dato primitivo sarà restituita la stringa [object Object]:

```
var persona = new Object({nome: "Mario",
  cognome: "Rossi"});
persona.toString(); //restituisce"[object Object]"
```

JavaScript

Oggetti: ValueOf

- Il metodo **valueOf()** restituisce il corrispondente valore del tipo di dato primitivo associato all'oggetto:

```
var x = new Object(32);
x.valueOf();                    //restituisce 32
```

Oggetto	Valore restituito
Array	Restituisce l'istanza della matrice.
Boolean	Valore Boolean.
Date	Il valore memorizzato dell'ora espresso in millisecondi a partire dalla mezzanotte dell'1 gennaio 1970 in formato UTC.
Funzione	Funzione stessa.
Number	Valore numerico.
Oggetto non associato a un tipo primitivo	Oggetto stesso. È l'impostazione predefinita.
Stringa	Valore stringa.

JavaScript
Oggetti: toString e valueOf

- È da sottolineare come JavaScript chiami implicitamente questi metodi quando è necessario effettuare delle conversioni o quando in un'espressione è richiesto il valore primitivo dell'oggetto.

Oggetti riflessi

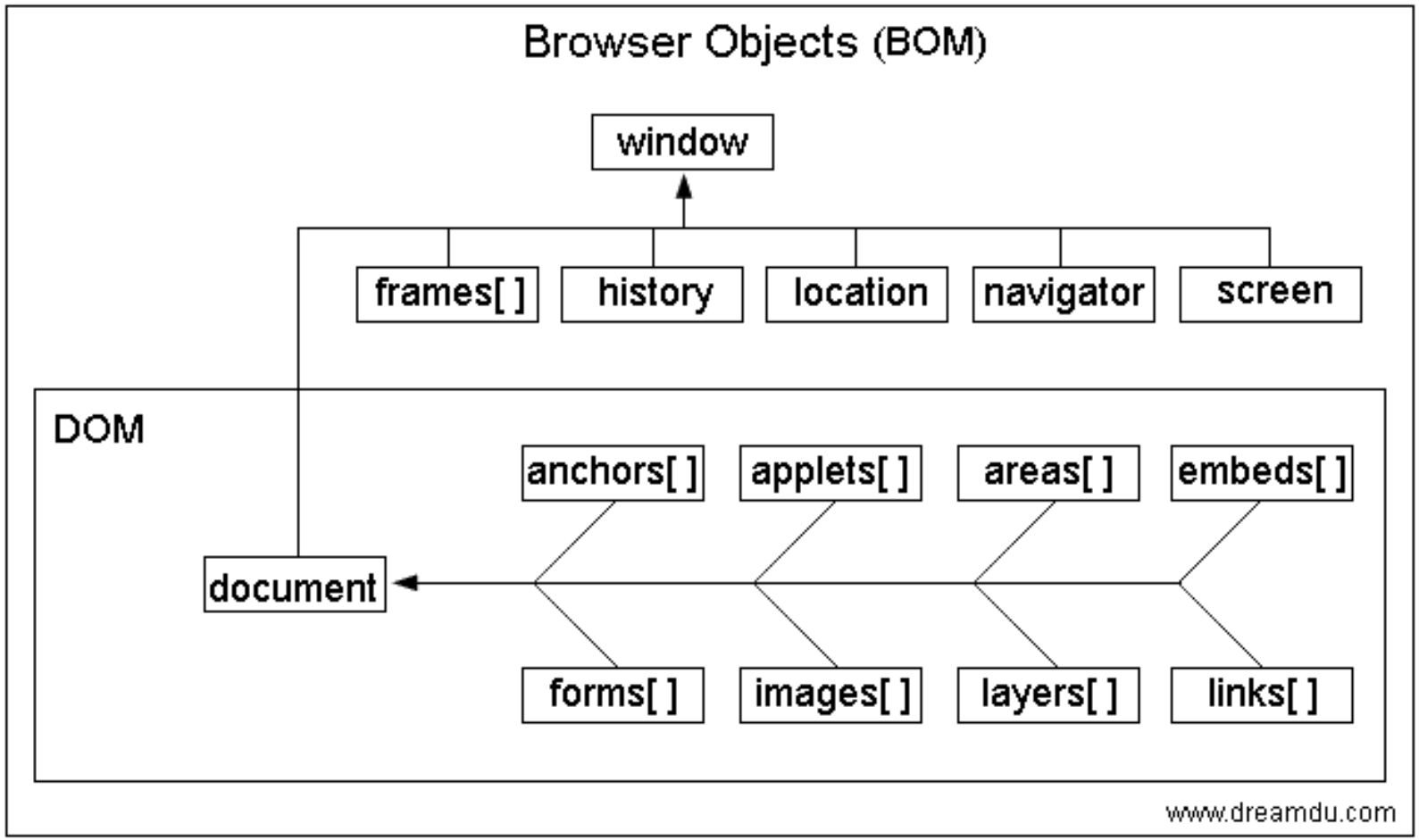
di Roberta Molinari

Javascript

Oggetti riflessi

- Javascript gestisce gli oggetti riflessi dal browser secondo 2 modelli a oggetti
  - **BOM**: il Browser Object Model fornisce l'accesso alle varie caratteristiche e all'ambiente del browser: finestra, schermo, cronologia,…
  - **DOM**: il Document Object Model fornisce l'accesso agli elementi che compongono il contenuto della finestra, ovvero il documento con le varie componenti HTML
- Javascript scompone una pagina web in una serie di oggetti in relazione gerarchica tra loro, ciascuno dotato di proprietà e metodi.

BOM e DOM

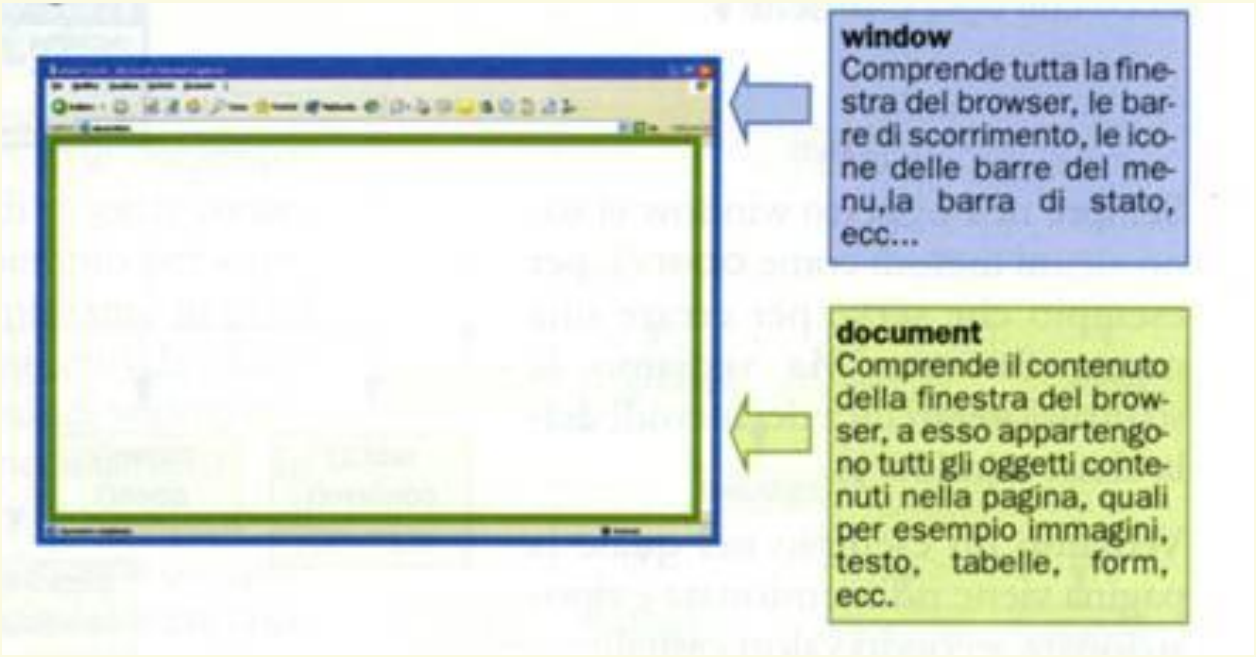


frames, forms, images,… sono vettori il cui primo elemento ha indice 0

Javascript

BOM

- **Window**: la finestra principale del browser, l'oggetto di livello massimo (sinonimo: self). Possono esserci più oggetti window contemporaneamente attivi



Javascript

BOM: window

Window: la finestra principale del browser

- Proprietà
  - ***status*** testo nella barra di stato (IE)
- Metodi
  - ***open() close()***
  - ***alert() prompt() confirm()***
  - ***setTimeot() clearTimeout()***
  - ***setInterval() clearInterval()***
- Eventi
  - ***onload***
  - ***onresize***
  - ***onunload***

Javascript

window.open()

w1=open("url","nome"[,"impostazioni"])

apre una nuova finestra e ***w1*** è il suo riferimento. Se mancano i primi parametri inserire ("","Nuova Finestra",..)

- *nome* può essere:
  - il nome della finestra (Nota: il nome non specifica il titolo della nuova finestra),
  - _blank (valore predefinito),
  - _parent e _top (per i frame)
  - _self per cui si va al nuovo URL che sostituisce la pagina corrente
- Le specifiche della finestra vanno separate da una virgola e sono illustrate nella tabella che segue. Quelle mancanti sono considerate false

Es. "toolbar=no, width=40, height=40, scrollbars=0, menubar=no"

Javascript window.open() specifiche

channelmode=yes|no|1|0nerhner	Whether or not to display the window in theater mode. Default is no. IE only
directories=yes|no|1|0	Obsolete. Whether or not to add directory buttons. Default is yes. IE only
fullscreen=yes|no|1|0	Whether or not to display the browser in full-screen mode. Default is no. A window in full-screen mode must also be in theater mode. IE only
height=pixels	The height of the window. Min. value is 100
left=pixels	The left position of the window. Negative values not allowed
location=yes|no|1|0	Whether or not to display the address field. Opera only
menubar=yes|no|1|0	Whether or not to display the menu bar
resizable=yes|no|1|0	Whether or not the window is resizable. IE only
scrollbars=yes|no|1|0	Whether or not to display scroll bars. IE, Firefox & Opera only
status=yes|no|1|0	Whether or not to add a status bar. IE
titlebar=yes|no|1|0	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box
toolbar=yes|no|1|0	Whether or not to display the browser toolbar. IE and Firefox only
top=pixels	The top position of the window. Negative values not allowed
width=pixels	The width of the window. Min. value is 100

Javascript

window.opener

- La proprietà **opener** restituisce un riferimento alla finestra che ha creato la finestra corrente con il metodo **window.open()**. In questo modo è possibile restituire valori alla finestra di origine (madre).

```
window.opener.close() //chiude la madre

opener.document.getElementById("txtMadre").
 innerHTML="testo dalla figlia";
//txtMadre deve essere dichiarato prima
```

JavaScript setTimeout() clearTimeout()

```
window.setTimeout ("funz()", millisec)
```

- Passati `millisec` millisecondi, si esegue la funzione `funz` 1 sola volta. Il primo parametro può essere:
  - il nome di una funzione `setTimeout(funz,30)`
  - la stringa contenente le istruzioni da eseguire (chiamata alla funzione o istruzioni estese)
    `setTimeout("alert('!')",30)`
  - dichiarazione della funzione
    `setTimeout(function(){alert('ciao')},3000)`
- Se si vuole evitare l'esecuzione si deve assegnare ad una variabile la funzione e poi eseguire una **`.clearTimeout`** prima che passi il tempo

```
t=window.setTimeout("funz()", millisec);
…
window.clearTimeout(t);
```

JavaScript setInterval() clearInterval()

```
window.setInterval("funz()", millisec)
```

- Come `setTimeout`, ma l'esecuzione viene eseguita ogni `millisec` millisecondi. Per il primo parametro valgono le stesse regole di `setTimeout`
- Se si vuole interromperne l'esecuzione si deve assegnare ad una variabile la funzione e poi eseguire una **`.clearInterval`**

```
t=window.setInterval("funz()", millisec);
…
window.clearInterval(t);
```

Javascript

BOM

- **Location**: contiene proprietà relative alla posizione del documento corrente come il suo URL
- **History:** la lista degli URL visitati nella sessione attuale
- **Navigator:** contiene alcuni dettagli sul browser in uso (es. nome e versione).
- **Screen**: contiene informazioni relative allo schermo del device utilizzato dall'utente.

Javascript

BOM: history

History: la lista degli URL visitati nella sessione attuale

- Proprietà
  - ***length*** numero di URL nella lista
  - ***current*** =history[0]
  - ***next previous***
- Metodi
  - ***go(n)*** va all'n-simo URL in history
  - ***go(str)*** carica il primo URL cha ha al suo interno una sottostringa =str
  - ***back()*** =history(-1)
  - ***forward()*** =history(1)

Javascript

BOM: location

Location: indirizzo documento corrente

`Protocol://Host:Port/Pathname/NomeFile`

`window.location.href=URL` si va a quella pagina modificando l'history

- Proprietà
  - ***Protocol hostname port pathname***
  - *href* equivale all'URL
- Metodi
  - ***Reload()***
  - ***Replace(url)*** come *.href* senza modificare l'history

Javascript

BOM: navigator

Navigator: per IE

- Proprietà (di sola lettura)
  - ***appcodName*** codice nome browser
  - ***appName*** nome browser
  - ***Appversion*** versione del browser
  - ***plugins*** array di plug-in istallati
  - ***mimeTypes*** array di MIME supportati
  - ***cookieEnabled*** ritorna se i cookie sono abilitati
  - ***geolocation*** ritorna l'oggetto con la posizione del client (ne chiede il permesso all'utente). Da >IE8

Navigator.geolocation .getCurrentPosition()

Il metodo

`.getCurrentPosition(showPosition, showError)`

ha due parametri che sono le funzini da attivare se tutto va bene, e se invece c'è un errore.

Il metodo restituisce un oggetto che ha le seguenti proprietà (le prime due sono sempre disponibili):

- **`.coords.latitude/longitude/altitude/speed`**
- **`.timestamp`** data/ora del rilevamento

A partire da Chrome 50, l'API di geolocalizzazione funziona solo su contesti protetti come HTTPS. Se il sito è ospitato su un'origine non sicura (come HTTP), le richieste per ottenere la posizione degli utenti non funzioneranno più.

Navigator.geolocation

```
var x = document.getElementById("latLog");
function getLocation() {
    if (navigator.geolocation) { //se supportato
        navigator.geolocation
       .getCurrentPosition(showPosition, showError);
    } else {
        x.innerHTML = "Geolocation is not supported by
this browser.";
    }
}
function showPosition(position) {
    x.innerHTML = "Latitude: " +
position.coords.latitude +
    "<br>Longitude: " + position.coords.longitude;
}
```

Navigator.geolocation

```
function showError(error) {
    switch(error.code) {
        case error.PERMISSION_DENIED:
            x.innerHTML = "User denied the request for
Geolocation."
            break;
        case error.POSITION_UNAVAILABLE:
            x.innerHTML = "Location information is
unavailable."
            break;
        case error.TIMEOUT:
            x.innerHTML = "The request to get user
location timed out."
            break;
        case error.UNKNOWN_ERROR:
            x.innerHTML = "An unknown error occurred."
            break;
    }
```

Navigator.geolocation

google API

Per poterlo usare bisogna avere un achive google

```
function showPosition(position) {
    var latlon = position.coords.latitude + "," +
position.coords.longitude;

    var img_url =
"https://maps.googleapis.com/maps/api/staticmap?center
=
    "+latlon+"&zoom=14&size=400x300&sensor=false
      &key=YOUR_:KEY";

    document.getElementById("mapholder").innerHTML =
"<img src='"+img_url+"'>";
}
```

Navigator.geolocation

google API

- I valori del parametro `zoom` hanno i seguenti significati:
  - 1: World
  - 3: Landmass/continent
  - 10: City
  - 15: Streets
  - 20: Buildings

Navigator.geolocation watchPosition()

- **`watchPosition()`** - Restituisce la posizione corrente dell'utente e continua a aggiornare la posizione mentre l'utente si sposta (come il GPS in un'automobile).
- **`clearWatch()`**- Interrompe l'aggiornamento

- Anche questo metodo ha due parametri funzione con lo stesso significato (funzione se tutto ok, funzione se errore)

Javascript

BOM: screen

Screen:

- Proprietà (di sola lettura)
  - ***availWidth e availHeight*** - dimensioni dello schermo esclusa la taskbar;
  - ***colorDepth*** - profondità del colore (bits x pixel);
  - ***pixelDepth*** - risoluzione colore dello schermo (bits x pixel);
  - ***width e height*** - dimensioni dello schermo;

Javascript

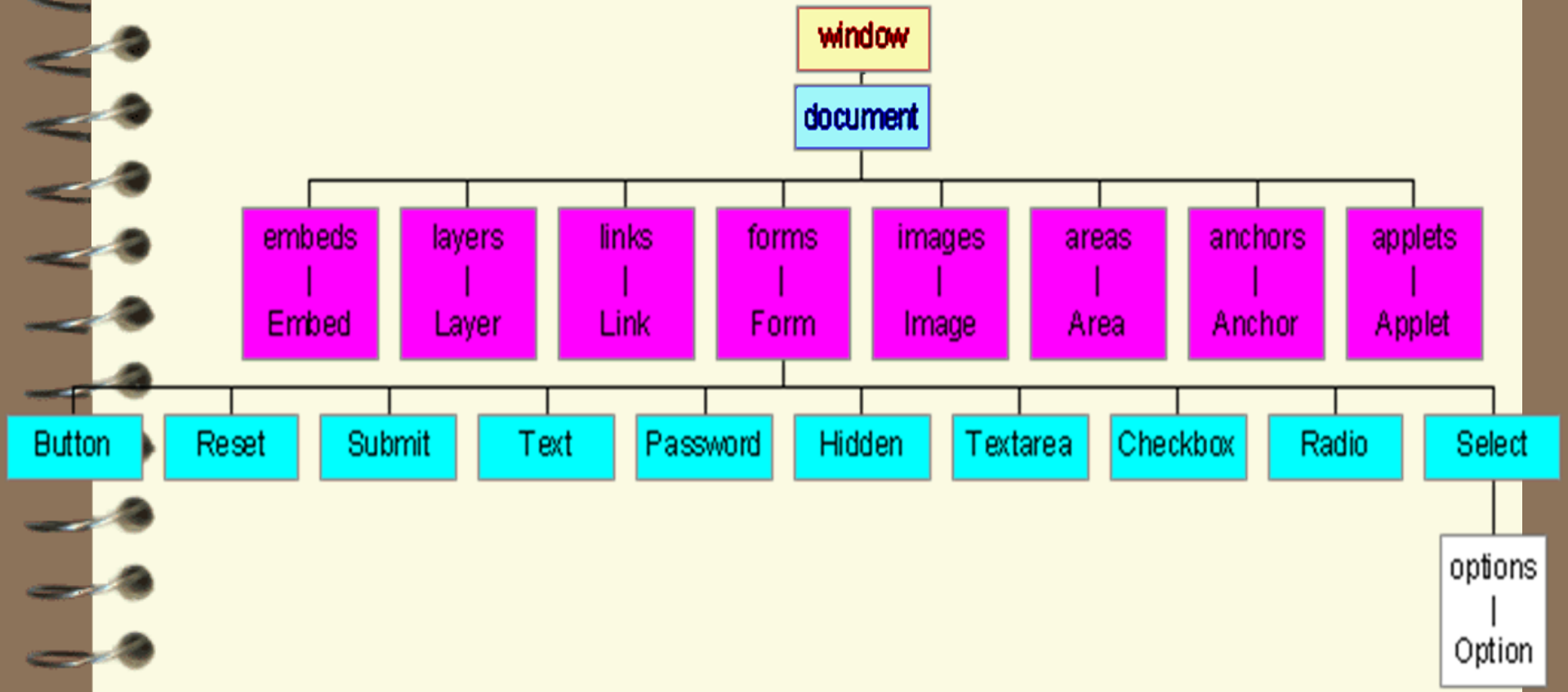
DOM

Il **DOM Document Object Model** è una standard di rappresentazione e di interazione con documenti strutturati (HTML o XML) visti come un insieme di oggetti.

Una pagina HTML in questo modello è rappresentata come un albero, in cui ogni elemento è genitore e/o figlio di un altro, tranne <HTML> che è l'elemento radice che racchiude tutti gli altri. Il DOM può essere pensato come la rappresentazione costruita dal browser in seguito al parsing del markup originale.

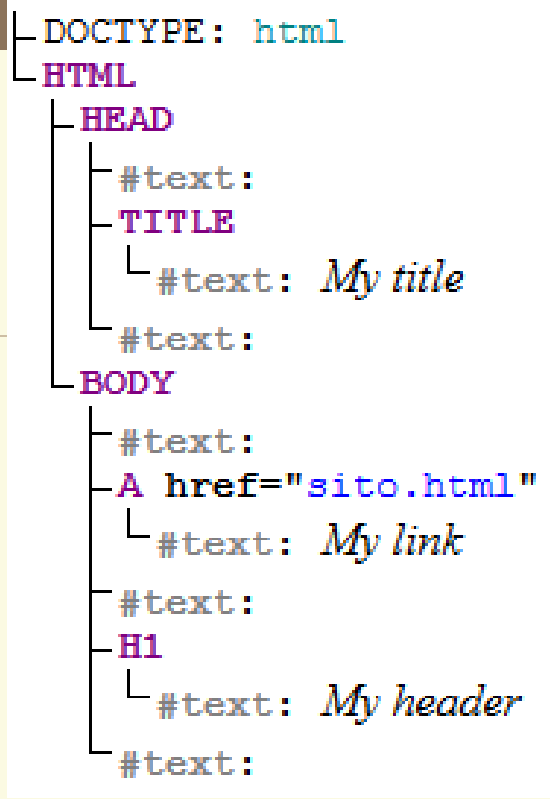
Gli oggetti nell'albero DOM potranno essere indirizzati e modificati usando metodi sugli oggetti. Pertanto mentre il contenuto del markup è statico, il contenuto del DOM è dinamico, in quanto può essere modificato dall'utente o da altri meccanismi

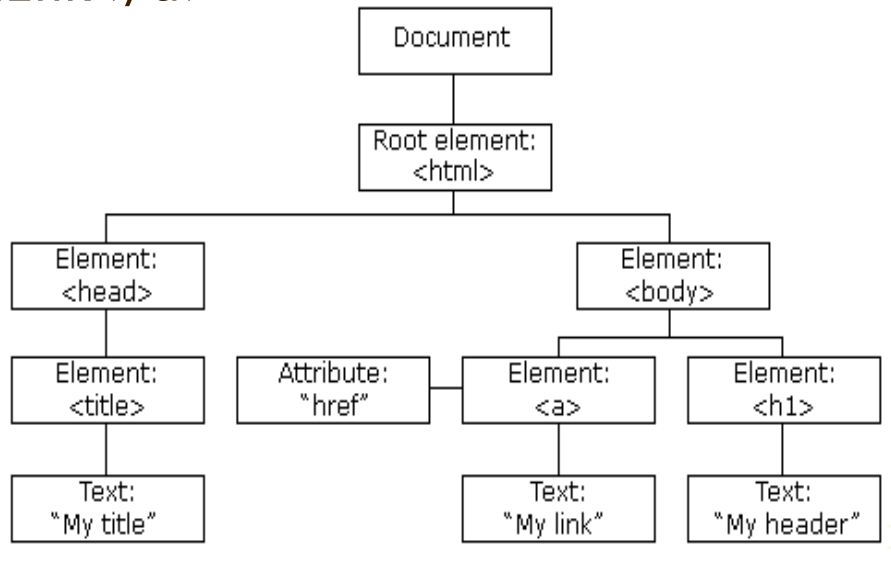
DOM


window
document
embeds | Embed
layers | Layer
links | Link
forms | Form
images | Image
areas | Area
anchors | Anchor
applets | Applet
Button
Reset
Submit
Text
Password
Hidden
Textarea
Checkbox
Radio
Select
options | Option

DOM

```
<!DOCTYPE HTML>
<html>
<head>
      <title>My title</title>
<head>
<body >
<a href="sito.html"> My link</a>
<h1>My header</h1>
</body>
</html>
```





Javascript

DOM: document

- **Document:** Parte del documento tra <body> e </body>, inserito nella finestra principale del browser. È la finestra attiva.

Tutti gli oggetti contenuti nel documento sono visti come sue proprietà, poiché risiedono al suo interno. Questi sono individuabili con vettori con indice che parte da 0 e segue l'ordine con il quale i tag (es. <IMG> per le immagini) vengono trovati nella pagina in una scansione dall'alto verso il basso (es. ogni elemento del vettore images è un'immagine presente nella pagina).

Javascript

Document : proprietà

- **`.alinkColor`**, **`.linkColor`**, **`.vlinkColor`** colore link
- **`.bgColor`**, **`.fgColor`** colore sfondo, testo
- **`.domain`** dominio del server in cui è caricato il documento
- **`.lastModified`** data ultima modifica
- **`.referrer`** se il documento corrente è stato raggiunto da un altro ne restituisce l'URL
- **`.title`** il titolo del documento
- **`.URL`** URL completo del documento

Javascript

Document : proprietà

- `.firstChild, .lastChild` primo/ultimo nodo dell'elemento
- `.parentNode` nodo padre dell'elemento
- `.id` valore dell'id

Javascript

Document: metodi

- **`getElementById("id")`** restituisce il riferimento all'elemento caratterizzato univocamente da quell'ID
- **`getElementsByTagName("tag")`** ritorna un array di tutti gli elementi del tag specificato
- **`getElementsByName("name")`** ritorna un array di tutti gli elementi con quel Name
- **`querySelectorAll("selCSS")`** ritorna la lista di tutti i selletori CSS che si chiamano selCSS (tipo "p b" o "p.class" o "#id")

Javascript

Document: metodi

- **`open()`** apre e cancella il contenuto attuale
- **`close()`** chiude e rende visibili le modifiche apportate
- **`write(s)` `writeln(s)`** scrive la stringa di testo **`s`** nel documento HTML (writeln le termina con '\n'). Se non c'è stata una open aggiunge in fondo

Javascript

Document: metodi

Per creare dinamicamente nuovo nodi nel DOM

- **`createTextNode(stringa)`** restituisce un text Node con il contenuto = stringa
- **`createElement(nomeTag)`** restituisce un elemento con il nome tag specificato (viene assegnato alla proprietà nodeName dell'elemento)
- **`appendChild(el)`** inserisce l'elemento `el` dopo l'ultimo figlio del nodo a cui è applicato elemento
- **`insertBefore`**`(newnode, existingNode)` inserisce prima dell'`existingNode` il `newNode` al nodo a cui è applicato. Se `existingNode=null` lo mette in coda

Javascript

Document: metodi

Per eliminare dinamicamente nodi nel DOM

- **`Nodo.removeChild(nodo)`** restituisce il nodo eliminato o NULL se il nodo non esiste
- **`hasChildNodes()`** verifica se un nodo ha dei figli

```
// Get the <ul> element with id="myList"
var list =
document.getElementById("myList");

// As long as <ul> has a child node, remove it
while (list.hasChildNodes()) {
    list.removeChild(list.firstChild);
}
```

Javascript

Document: metodi

```
var btn
= document.createElement("BUTTON");
btn.setAttribute("onclick","window.locati
on.href='http://www.google.it'")
var t =document.createTextNode("Google");
btn.appendChild(t)    //Append the text to <button>
//si potrebbe usare innerHTML
document.body.appendChild(btn)
                      //Append <button> to<body>
```

Javascript

Document: metodi

ATTENZIONE!

Per i radio, il testo che compare dopo non è nel suo innerHTML.

```
opt = document.createElement("input")
opt.type="radio"; opt.name= "sesso" ; opt.value="M"
document.body.appendChild(opt)
testo = document.createTextNode("Maschio")
document.body.appendChild(testo)

opt = document.createElement("input")
opt.type="radio"; opt.name= "sesso" ; opt.value="F"
document.body.appendChild(opt)
testo = document.createTextNode("Femmina")
document.body.appendChild(testo)
```

Javascript

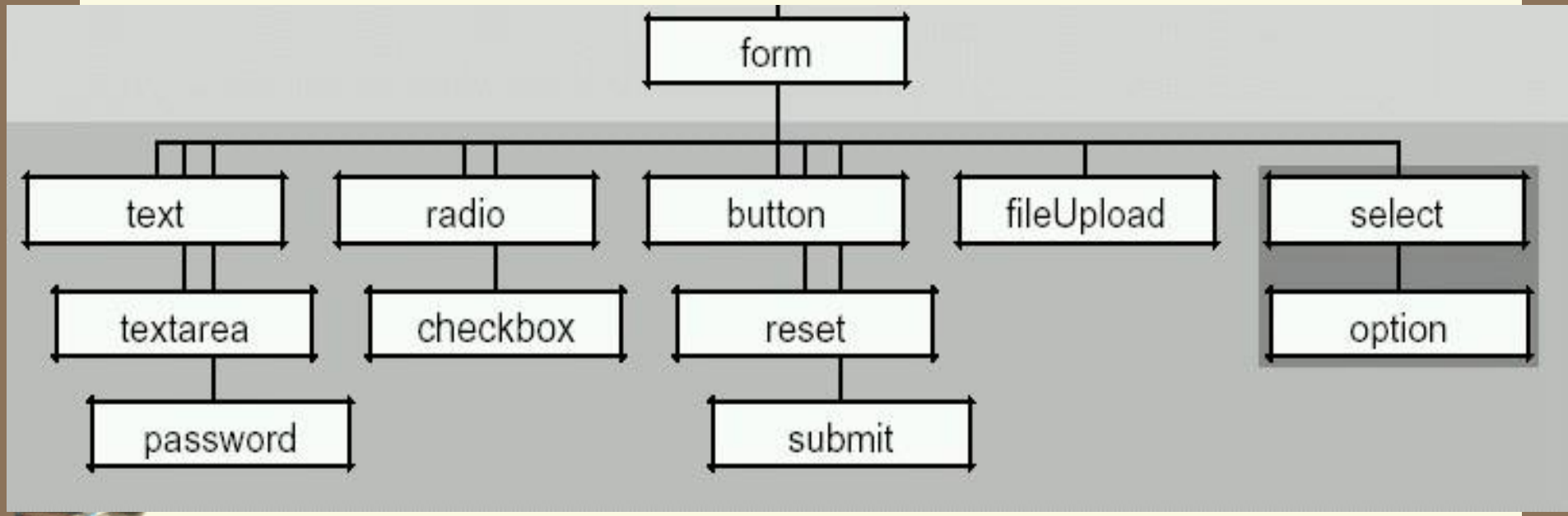
Oggetti riflessi HTML

- **document.images**
- **document.links**
- **document.anchors**
- **document.cookies**
- **document.layers** i livelli presenti
- **document.applets**
- **document.embeds** gli oggetti multimediali
- **document.forms**

Javascript

document.forms

- a ogni form viene associato un vettore di nome **elements** che contiene tutti gli oggetti contenuti nel form stesso.



Javascript
Riferimento agli oggetti riflessi

- Per fare riferimento alla prima immagine, di nome/id *imgImmagine*, presente nella pagina
  - `window.document.images[0]` //rif. Assoluto
  - `document.images["imgImmagine"]`
  - `document.imgImmagine`
  - `imgImmagine` //rif. relativo
- Per fare riferimento al primo elemento del form *frmDati*, che è una casella di testo di nome/id *txtNomeCognome*:
  - `window.document.forms["frmDati"].elements[0]`
  - `document.frmDati.elements[0]`
  - `frmDati.txtNomeCognome`

Javascript
Riferimento agli oggetti riflessi

- Si possono usare i metodi **getElement** per riferirsi agli oggetti del documento: per riferirsi all'immagine di nome *imgImmagine* e id univoco *img1* si fa

```
document.getElementById("img1")
document.getElementsByName("imgImmagine")
```

  - Per riferirsi a tutte le immagini

```
x=document.getElementsByTagName("img");
x.item(k). …
x[k]. …
```

Nota: il "nomeId" è case sensitive

Javascript

Proprietà .innerHTML

È possibile modificare il contenuto di un nodo (quello che compare tra il tag di apertura e il relativo tag di chiusura) attraverso la modifica della sua proprietà `.innerHTML`

```
document.getElementById(id).innerHTML
                = codice_HTML;
```

dove `codice_HTML` è una stringa contenente il codice HTML da inserire all'interno del nodo.

Attenzione a dove si vuole inserire il nuovo codice HTML, in quanto il metodo innerHTML sostituisce tutto il contenuto del nodo al quale viene applicato, con il nuovo codice specificato.

```
<div id="testo"> <i>vecchio testo</i> </div>
 document.getElementById("testo").innerHTML =
      "<b>nuovo testo</b>";
```

Javascript
Riferimento agli oggetti riflessi

- Per modificare un attibuto

  `.setAttribute("nome","val")`

- Per recuperare un attibuto

  `.getAttribute("nome");`

- Per eliminare un attibuto

  `.removeAttribute("nome");`

- Per verificare se possiede quell'attributo

  `.hasAttribute("nome");`

```
var x = document.getElementById("myAnchor")
if (x.hasAttribute("target")) {
    x.setAttribute("target", "_self");
}
```

Nota: il "nome" NON è case sensitive

Javascript
Riferimento agli oggetti riflessi

- Per modificare un attributo si può fare anche direttamente

`.nomeAttributo`

```
var x = document.getElementById("myAnchor")
x.target="_self"; var y= x.target;
```

Nota: il "nome" dell'attributo è case sensitive

- Per modificare un attributo dello style

`.style.nomeAttributo`

```
var x = document.getElementById("myAnchor")
x.target="_self"
var y= x.target
```

Javascript

Abbreviazioni

- Uso di **this** per riferirsi all'oggetto corrente

```
<input type ='text' name='txtNome'
onblur='numero(this.value)'>
```

- Uso di **with** per non ripetere il riferimento completo all'oggetto

```
with (oggetto) {comandi}
```

- Definire una variabile che contenga il riferimento ad un oggetto

```
var OggImmagine = document.imgImmagine;
OggImmagine.width += i;
```

Javascript

Form

- Proprietà
  - **`name, action, enctype, method, target`**
  - **`elements`** vettore degli elementi del form
  - **`length`** numero di elementi compresi hidden e pulsanti
- Metodi
  - **`reset(), submit()`**
- Gestore Eventi
  - **`onClick, onSubmit, onReset`**

Javascript

Button

Include anche i pulsanti speciali *submit*, *reset*

- Proprietà
  - **`name`**, **`type`** (button, reset o submit), **`value`**
- Metodi
  - **`click()`**
- Gestore Eventi
  - **`onClick`**, **`ondblclick`**

Javascript

Text

- Proprietà
  - **`name`**, **`type`** (password), **`value`**, **`length`**, **`defaultValue`**
- Metodi
  - **`focus()`**, **`blur()`**, **`select()`**
- Gestore Eventi
  - **`onFocus`** imposta lo stato attivo
  - **`onBlur`** elimina lo stato attivo
  - **`onSelect`** seleziona il testo nella text

Javascript

Textarea

- Proprietà
  - `name, type, value, length, cols, rows, defaultValue`
- Metodi
  - `focus(), blur(), select()`
- Gestore Eventi
  - `onFocus, onBlur, onSelect, onChange`

Javascript

Checkbox

- Proprietà
  - `name, type, value, checked`
- Metodi
  - `click()`
- Gestore Eventi
  - `onClick`

Se hanno un solo nome si genera un array

Javascript

Radio

- Proprietà
  - **`name, type, value, checked, defaultchecked`**
  - **`Lenght`** (numero di pulsanti presenti)
- Metodi
  - **`click()`**
- Gestore Eventi
  - **`onClick`**

Ha un solo nome si genera un array

Javascript

Select

- Proprietà
  - **`name, value, checked, text`** (tra option)
  - **`type (select-one, select-multiple),`**
  - **`lenght`** (numero di option)
  - **`options`** (oggetti option contenute)
  - **`selectedIndex`** (indice opzione selezionata, il primo se multiple)
- Metodi
  - **`blur(), focus()`**
- Gestore Eventi
  - **`onBlur, onChange, onFocus`**

Javascript

Select.options

- Proprietà
  - **`lenght`** (numero di option)
  - **`selectedIndex`** (imposta o restituisce l'indice dell'opzione selezionata (il primo=0), il primo se multiple)
- Metodi
  - **`add(option[,index])`** Aggiunge la option in fondo o nella posizione eventualmente indicata
  - **`remove(index)`** elimina la option con indice index

```
var c = document.createElement("option");
c.text = "Prova";
mySelect.options.add(c, 1);
mySelect.options.remove(1);
```

JavaScript
Schema oggetti riflessi HTML

Oggetto	Proprietà	Metodi	Eventi
button	name – type – value - disabled	click - dblclick	onClick - ondblclick
text	name – type – value – size – defaultValue – value.length - disabled	focus – blur - select	onFocus – onBlur – onSelect – onChange
textarea	name – value – size – cols – rows - disabled	focus – blur - select	onFocus – onBlur – onSelect – onChange
checkbox	name – type - length value – checked (T|F) - defaultChecked – disabled (di un [k])	click	onClick
radio	name – type – length value - checked (T|F) - defaultChecked – disabled (di un [k])	click	onClick
select	name – value – type(select-one; select-multiple) - length – selectedIndex – options - disabled	focus – blur	onFocus – onBlur – onChange
options (array)	index – text (di un options[k]) length – selected – selectedindex (dell'array options) - disabled		

JavaScript
Schema oggetti riflessi HTML

- In realtà i metodi e gli eventi sono utilizzabili da tutti gli oggetti, ma quelli in elenco sono i più sensati
- `type` è una proprietà di sola lettura
- `blur()` fa perdere il focus e non lo assegna a nessuno
- `disabled` rende non "toccabile" dall'utente
- `default` restituisce il valore o le impostazioni di checked iniziali
- `Check|Radio[k].value` è quello impostato con value
- `Check|Radio.length`= quanti oggetti check o radio hanno lo stesso nome. Per riferirsi ai vari radio si usa `form. Check|Radio[k]`
- Nella select comunque qualcosa è selezionato e il suo indice è reperibile con `select.selectedIndex` o `select.options.selectedIndex`, mentre il suo value (che è sempre di tipo string) è reperibile con `select.value` o con `select.options[select.selectedIndex].value`
- `Select.options[k].text`= testo che segue il tag option

Javascript

Esempio controllo input

```
function checkData(){
 var err = ""; name=document.getElementById("username");
 if (name.value=='' || name.value.length<2)
       err += " Manca nome\n";
 news=document.getElementsByName("news")
 if (!news[0].checked && !news[1].checked)
       err += "Seleziona…\n";
 if   (document.getElementById("combo").selectedIndex==0)
       err+="seleziona…"
 if (tel.value=="" || isNaN(tel.value))
//devono essere in quest'ordine perechè isNaN("")=false
   err+="Formato numero di telefono sbagliato"

if (err=="") document.forms["myForm"].submit();
 else alert (err);
}
```

Javascript
Esempio controllo input regex

Nel caso di caselle di testo, si possono usare le regex per verificare la correttezza del formato del testo.

Si dichiara una costante REGEX con la stringa di confronto e poi si usa il metodo `REGEX.test(testoDaVerificare)`

```
const REGEX = /^[a-zA-Z ]+$/ "
if( txtNome.value=="" || !REGEX.test(txtNome.value)){
    err += "Formato nome scorretto\n";
}
```

allows YYYY/M/D and periods instead of slashes	/^\d{4}[\/.]\d{1,2}[\/.]\d{1,2}$/
YYYY-MM-DD and YYYY-M-D	/^\d{4}\-\d{1,2}\-\d{1,2}$/
YYYY-MM-DD	/(\d{4})-(\d{2})-(\d{2})/
nomi e cognomi	/^[a-zA-Z]+$/

Javascript
Esempio mostra nascondi

```
function mostra() {
document.getElementById("content").style.display="block";
}
function nascondi() {
document.getElementById("content").style.display="none";
}

<div id="content">
      Questo è il div da mostrare o nascondere
</div>
<input type="button" value="Mostra" onclick="mostra()"/>
<input type="button" value="Nascondi"
onclick="nascondi()"/>
```

Javascript
modificare stili CSS

Una delle caratteristiche più utili di Javascript, e del DOM in particolare, la possibilità di aggiungere e modificare stili CSS degli elementi di pagina

```
var el=document.getElementById("bigtext");
el.style.fontSize="50px";
```

- essendo **float** una parola riservata in Js: gli equivalenti saranno quindi **cssFloat** per browser quali Opera, Mozilla e Safari, mentre **styleFloat** per Internet Explorer. Basterà impostarle entrambe per ottenere il risultato voluto in maniera cross-browser:

```
var el=document.getElementById("box");
el.style.styleFloat="left";
el.style.cssFloat="left";
```

Javascript

modificare stili CSS

Se le dichiarazioni CSS da aggiungere mediante Javascript dovessero essere diverse c'è un'alternativa che si rivela più leggera in termini di peso: intervenire sull'attributo HTML **style** degli elementi di pagina, proprio come se si impostasse uno stile in linea, attraverso il metodo **setAttribute**.

```
function so_applyStyleString(obj,str){
if(document.all && !window.opera)
    obj.style.setAttribute("cssText",str);
else
    obj.setAttribute("style",str);
}


var s="float:right;width:10em;border:1px dotted
#CCC;padding:5px"
var divs=document.getElementsByTagName("div");
for(i=0;i<d.length;i++){
    if(divs[i].className=="pullquote")
        so_applyStyleString(divs[i],s);
}
```

Javascript

modificare stili CSS

La proprietà **className** consente di accedere sia in lettura che in scrittura alle classi CSS attributie sia nel markup che da Javascript stesso.

```
var el=document.getElementById("menu");
el.className="open";
```

Per non perdere le classi precedenti

```
el.className + =" open";
```

Javascript

modificare stili CSS

Attraverso la proprietà **style** degli elementi DOM è possibile accedere a proprietà CSS in Javascript. La variabile **contentColor** conterrà effettivamente un valore non nullo *solo se la proprietà CSS è stata in precedenza impostata mediante Javascript.*

```
var el=document.getElementById("content");
var contentColor=el.style.backgroundColor;
```

document.defaultView.getComputedStyle del DOM consente di accedere a valori CSS non precedentemente impostati mediante Javascript.

```
function getStyleProp(x,prop){
if(x.currentStyle)          //ci sono differenze con IE
    return(x.currentStyle[prop]);
if(document.defaultView.getComputedStyle)
    return(document.defaultView.getComputedStyle(x,")[pro
p]);
return(null);
}
```

Javascript

modificare stili CSS

- Una buona pratica in soluzioni basate su CSS e Javascript è tenere il CSS funzionale e/o presentazionale della soluzione arricchita in un *CSS esterno e separato* rispetto a quello che riguarda layout e presentazione della pagina senza Javascript. Detto ciò, i modi di procedere sono sostanzialmente tre.
- Il primo e più semplice è linkare nella sezione head sia il javascript sia il CSS:
- <script type=”text/javascript” src=”nifty.js”></script> <link rel=”stylesheet” type=”text/css” href=”nifty.css”>
- Questa soluzione ha un piccolo svantaggio, ovvero far scaricare all’utente il CSS necessario anche se Javascript non può girare. Gli utenti che navigano con Javascript disabilitato o con browser non DOM-compatibili saranno una percentuale che oscilla tra il 5 e il 10%, ma servire un CSS di cui non potranno beneficiare uno spreco di byte sia per l’utente che per il server.
- Il secondo e il terzo approccio linkano il CSS solo nel caso in cui Javascript sia abilitato. Ma come fare? Semplice, basta includere il CSS tramite il javascript stesso: ecco così la seconda soluzione. All’interno del

Oggetti definiti dall'utente

di Roberta Molinari

JavaScript
Oggetti definiti dall'utente

- Javascript è un linguaggio **object based** (si possono usare gli oggetti), ma non object oriented (non ci sono classi, un oggetto non è istanza di una classe o una vera ereditarietà).
- **Un oggetto è un contenitore di proprietà**, cioè di elementi caratterizzati da un nome. Una **sorta di array associativo** che è possibile costruire e modificare dinamicamente
- Si possono creare oggetti in vari modi
  - con la **notazione letterale** {nome:val,...}
  - definendo una **funzione costruttore**

JavaScript

Oggetti chiave: valore

- Per creare un oggetto con la notazione letterale si definiscono le sue proprietà come **chiave: valore**

```
var person = { //anche su una riga sola
    firstName : "John",
    lastName : "Doe",
    age : 50
};
```

- Una proprietà può assumere qualsiasi valore, compreso un altro oggetto (come le strutture in C)

```
var persona = {
   nome: "Mario",
   cognome: "Rossi",
   indirizzo: {via: "Via Garibaldi", num: 15}
};
```

JavaScript
Oggetti: funzioni costruttore

- Un costruttore è una normale funzione invocata con l'operatore **new**

```
function Persona(){  //typeof function
    this.nome=n;
    this.cognome=c;
}
```

this è una parola chiave (non una variabile) che indica l'oggetto che possiede il codice. La parola `this` dichiara una proprietà public, se non c'è è una variabile

```
var mr = new Persona();
mr.nome ="Mario"; mr.cognome = "Rossi";
```

JavaScript
Oggetti: funzioni costruttore

- Possiamo prevedere dei parametri nel costruttore

```
function Persona(n,c){
    this.nome=(n);
    this.cognome=(c);
}
```

- ATTENZIONE! se nella creazione dell'oggetto omettiamo `new`, quello che otterremo non sarà la creazione di un oggetto ma l'esecuzione della funzione, con risultati imprevedibili.
- Con `new` typeof → object
- Senza `new` typeof → undefined

JavaScript
Oggetti: funzioni costruttore

- Possiamo prevedere dei valori di default in modo diverso da `function Persona(n="",c="")`

```
function Persona(n,c){
    this.nome=(n || "");
    this.cognome=(c || "");
}
```

e in questo caso nella creazione si può fare in 2 modi

```
var rossi =
    new Persona(undefined,"Rossi"); //nome==""
var rossi =
    new Persona(null, "Rossi"); //nome==="",
//non possibile nell'altro modo
```

JavaScript

Oggetti: new

- Per creare un oggetto sfruttando l'oggetto Object, che è alla base di qualsiasi oggetto JavaScript, si usa **new Object()**

```
var persona = new Object({
    nome: "Mario",
    cognome: "Rossi"
});
```

- Per creare **oggetti vuoti** (senza proprietà)

```
var auto = new Object(); //typeof object
```

Equivale a

```
var auto ={};
```

JavaScript

Oggetti: proprietà

- I nomi delle proprietà non hanno le restrizioni dei nomi delle variabili, ma se le infrangono vanno tra ""

```
{"primo-nome": "Mario",
 "secondo.nome": "Rossi"}
```

- Per accedere alle proprietà degli oggetti si usa la **dot notation** o la notazione degli **array associativi** (obbligatoria se i nomi non seguono le regole dei nomi delle variabili)

```
a = oggetto.nome_proprietà
a = oggetto ["nome_proprietà"]
x = nome_proprietà"; a= oggetto[x]
```

JavaScript

Oggetti: proprietà

- Per ottenere tutte le proprietà si utilizza il for-in

```
for (campo in p1){
        alert( p1[campo])     //NON p1.campo
}
```

- Si possono aggiungere proprietà pubbliche dinamicamente che apparterranno solo all'oggetto

```
persona.hobby="scacchi"
```

- Si possono anche eliminare dinamicamente le proprietà ai singoli oggetti (Dopo l'eliminazione della proprietà, ogni tentativo di accesso ad essa restituirà il valore undefined)

```
delete persona.hobby
```

JavaScript

Oggetti: metodi

- Un metodo non è altro che una funzione assegnata ad una proprietà. Si può aggiungerlo direttamente all'oggetto

```
var persona={
      nome= "Mario", cognome = "Rossi",
      visualizza()= function(){
      return this.nome+this.cognome
      }
```

- Si possono aggiungere dinamicamente metodi pubblici

```
pers1.fun=function(){}
```
//solo a pers1

JavaScript

Oggetti: metodi

- Si può aggiungerlo nel costruttore

```
function  Persona=function(n,c){
    this.nome=n;
    this.cognome=c;
    this.visualizza()= function(){
        return this.nome+this.cognome
    }
```

- Se dichiaro al suo interno una funzione senza il this, è una funzione NON un metodo
- Per richiamarlo

```
var pers1=new Persona(…)
pers1.visualizza()
```

JavaScript

Oggetti: metodi

- I metodi si possono anche dichiarare così
  1. Si definisce separatamente la funzione
  2. Si assegna alla classe nello stesso modo che una proprietà

```
function visNomeCogn() {
    return this.nome+this.cognome}
Persona=function(n,c){
    this.nome=n;   this.cognome=c;
    this.visualizza=visNomeCognome //senza()
}
```

non assegniamo alla proprietà il risultato della chiamata alla funzione, ma la funzione stessa tramite il suo nome. La proprietà **visualizza**, dal momento che contiene una funzione, è di fatto un metodo.

JavaScript

Oggetti: prototipi

- Per aggiungere una proprietà/metodo a tutte le istanze si usa **`prototype`** (simula l'ereditarietà)

```
Persona.prototype.hobby="scacchi"
Persona.prototype.fun=function(){…}
```

- La nuova proprietà non è direttamente agganciata a ciascun oggetto, ma accessibile come se fosse una sua proprietà. Il **prototipo** di un oggetto è una sorta di riferimento ad un altro oggetto. Gli oggetti che creiamo tramite la semplice applicazione della notazione letterale hanno come prototipo Object, tramite costruttore hanno l'oggetto prototype del costruttore.

JavaScript
Oggetti: Ereditarietà prototipale

- Il meccanismo su cui si basa l'**ereditierietà prototipale** è se una proprietà non si trova in un oggetto viene cercata nel suo prototipo.
- Il prototipo di un oggetto può a sua volta avere un altro prototipo. In questo caso la ricerca di una proprietà o di un metodo risale la catena dei prototipi fino ad arrivare all'oggetto Object, il prototipo base di tutti gli oggetti.
- Anche gli oggetti predefiniti di JavaScript hanno un prototipo di riferimento, perciò si possono modificare.
- Se si cancella la proprietà di un oggetto, non si cancella dal suo prototipo, ma se la cencelli dal prototipo la cancelli da tutti gli oggetti

JavaScript

Funzioni come oggetti

- In Javascript, anche se il typeof su funzioni restituisce function, sono descrivibili come oggetti con metodi e proprietà
- `arguments.length`: proprietà = al numero di argomenti
- `toString()`: applicato alla funzione restituisce la funzione come stringa

```
function myFunction(a, b) {
    return a * b;
}
var txt = myFunction.toString();
```

JavaScript

ricapitolando

```
var x1 = {};      // new object
var x2 = "";      // new primitive string
var x3 = 0;       // new primitive number
var x4 = false;   // new primitive boolean
var x5 = [];      // new array object
var x6 = /()/     // new regexp object
var x7 = function(){};
                  // new function object
```

Gli eventi in Javascript

di Roberta Molinari

JavaScript

Gli eventi

Per gestire un evento si deve invocare il suo gestore (handler):

1. Assegnando l'handler (possono anche essere istruzioni) direttamente all'attributo associato all'evento nel tag HTML:

```
<IMG onClick="nomeFunz()">
```

2. Associando la funzione di callback alle relative proprietà dell'oggetto:

```
window.onResize=nomeFunz;          //senza parentesi
```

3. Tramite `addEventListener` sull'oggetto DOM (così si ha maggiore flessibilità)

```
body.addEventListener("onLoad",nomeFunz)
```

JavaScript

Gli eventi

Se si utilizza il primo metodo e la funzione restituisce true, viene anche eseguita l'operazione associata di default all'evento, se restituisce false non viene eseguita:

```
<a href="#" onClick="nomeFunz()">
```

"#" indica di posizionarsi ad inizio pagina, se `nomeFunz()` restituisce false, non verrà eseguita

```
<input type="submit"
   onClick="return confirm('Confermi?')>
```

JavaScript

Gli eventi

addEventListener("evento", funz)

Si può usare in questo modo

```
document.addEventListener("click", myFunction);

function myFunction() {
    document.getElementById("demo").innerHTML =
"Hello World";
}
```

o in questo

```
document.addEventListener("click", function(){
    document.getElementById("demo").innerHTML =
            "Hello World";
});
```

- Per eliminare un handler **.removeEventListener**("",f)

JavaScript Gli eventi

Evento	Tag	Descrizione
abort	<img>	L'utente fallisce il caricamento di un'immagine.
blur	<body> <frameset> <frame> <input type = "text"> <textarea> <select>	Un documento perde il focus dell'input. Un frame perde il focus dell'input. Un frame perde il focus dell'input. Un campo di testo perde il focus dell'input. Un'area di testo perde il focus dell'input. Un elemento di selezione perde il focus dell'input.
change	<input type = "text"> <textarea> <select>	Un campo di testo viene modificato e perde il focus dell'input. Un'area di testo viene modificata e perde il focus dell'input. Un elemento di selezione viene modificato e perde il focus dell'input.
click	<a> <input type = "button"> <input type = "submit"> <input type = "reset"> <input type = "radio"> <input type = "checkbox">	L'utente fa clic su un collegamento. Viene selezionato un pulsante. Viene selezionato il pulsante Submit. Viene selezionato il pulsante Reset. Viene selezionato un pulsante di opzione. Viene selezionata una casella di controllo.

JavaScript

Gli eventi

Evento	Tag	Descrizione
error	<img> < body> < frameset>	Si è verificato un errore nel caricamento di un'immagine. Si è verificato un errore nel caricamento di un documento. Si è verificato un errore nel caricamento di un set di frame.
focus	<body> <frameset> <frame> <input type= "text"> <textarea> <select>	Un documento diventa attivo con l'input. Un set di frame diventa attivo con l'input. Un frame viene diventa attivo con l'input. Un campo di testo diventa attivo con l'input.. Un'area di testo diventa attiva con l'input. Un elemento di selezione diventa attivo con l'input.
load	<img> <body> <frameset>	L'utente ha caricato e visualizzato un'immagine. Il caricamento del documento è completato. Il caricamento del documento è completato.

JavaScript

Gli eventi

Evento	Tag	Descrizione
mouseout	<a> < area>	L'utente allontana il cursore del mouse dal collegamento. Il cursore del mouse viene spostato fuori dalla mappa immagine.
mouseover	<a> < area>	Il cursore del mouse viene spostato su un collegamento. Il cursore del mouse viene spostato sull'area di una mappa immagine.
reset	<form>	Viene selezionato il pulsante Reset.
resize	<body> <frameset>	L'utente modifica le dimensioni della finestra. L'utente modifica le dimensioni del frame
select	<input type "text"> <textarea>	Il cursore del mouse viene spostato su un campo di testo. Il cursore del mouse viene spostato all'intemo di un'area di testo.
submit	<form>	Viene selezionato il pulsante Submit.
unload	<body> <frameset>	L'utente esce dal documento. L'utente esce dal set di frame.

JavaScript

L'oggetto Event

- Il DOM prevede che ad ogni gestore di eventi venga passato come parametro l'**oggetto event** contenente informazioni su di esso.

```
function gestoreEvento(e){
      //e contiene un'istanza dell'oggetto event
      e.target.id
      //id dell'oggetto che ha scatenato l'evento
}
```

- Per assegnare un'event handler attraverso l'HTML che gestisca l'oggetto *event*, occorre specificarlo tra i parametri passati

```
<p id="myP" onclick="gestoreEvento(event)">…
```

mentre non cambia nulla se l'evento è definito via Javascript

```
document.getElementById("myP").onclick =
                gestoreEvento
```

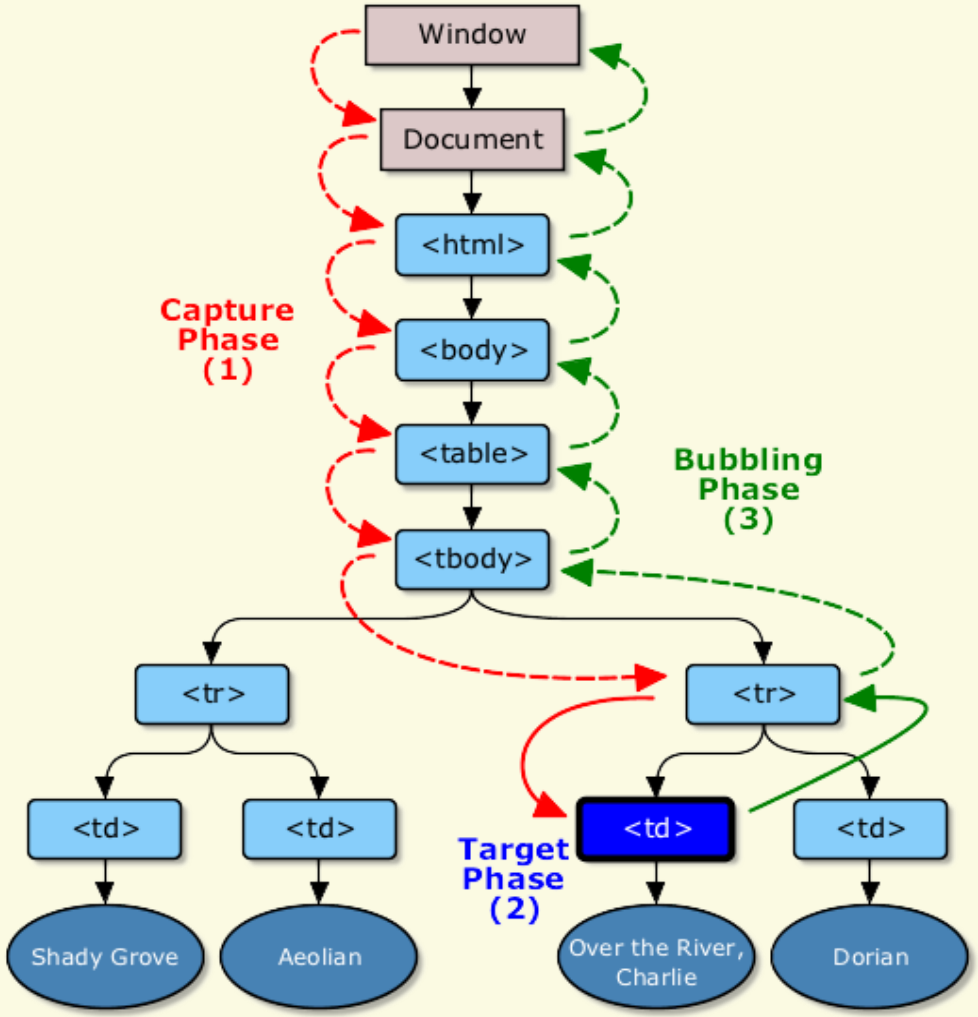
JavaScript
L'oggetto Event: propagazione

Secondo le specifiche del W3C, la propagazione di un evento avviene in tre fasi:

Capture phase	In questa fase l'evento si propaga dalla radice del DOM verso l'elemento destinatario effettivo
Target phase	In questa fase l'evento raggiunge l'elemento destinatario
Bubble phase	Questa è la fase in cui l'evento risale l'albero del DOM partendo dall'elemento target fino a raggiungere la radice, passando quindi dagli stessi nodi attraversati nella fase di cattura.

Durante le 3 fasi, un oggetto *event* associato all'evento viene passato agli eventuali gestori incontrati lungo il cammino

JavaScript
L'oggetto Event: propagazione



Normalmente, nella fase di capturing i gestori ignorano l'evento e lo fanno fluire verso l'elemento target. Una volta raggiunto l'elemento target viene eseguito il codice del gestore associato all'evento. Nella successiva fase di bubbling vengono eseguiti i gestori dell'evento che si incontrano man mano che si va verso la radice del DOM

JavaScript
L'oggetto Event: proprietà

- **`target`** restituisce un riferimento al nodo obiettivo del flusso d'evento nella sua fase di capturing (punto di partenza della successiva fase di bubbling)
- **`relatedTarget`** nel caso di un evento di *onmouseover*, la proprietà contiene un riferimento all'elemento dal quale il mouse proviene, cioè l'elemento appena lasciato. Nel caso dell'*onmouseout*, contiene un riferimento all'elemento verso il quale il mouse è diretto, ovvero l'elemento nel quale ci si sposta.
- **`currentTarget`** restituisce un riferimento al nodo per il quale il flusso d'evento sta passando

JavaScript
L'oggetto Event: proprietà

- **`screenX, screenY`** restituisce la coordinata X/Y del puntatore del mouse relativa allo schermo
- **`x, y`** restituisce la coordinata X/Y rispetto alla finestra del browser
- **`button`** restituisce quale pulsante del mouse ha cambiato il proprio stato, ovvero sia stato premuto o rilasciato. I valori restituiti possono essere:
  - **0** : pulsante sinistro [1 per Netscape]
  - **1** : pulsante centrale se possiede 3 pulsanti [2 per Netscape]
  - **2** : pulsante destro [3 per Netscape]
- **`charCode`** codice del carattere Unicode premuto da tastiera

JavaScript
L'oggetto Event: proprietà

- **`type`** restituisce una stringa che descrive il tipo di evento, come ad esempio: "click","mouseover", ecc
- **`cancelable`** indica se l'azione di default di un evento possa essere cancellata (true|false)
- **`preventDefault()`** consente di cancellare l'azione di default per quegli eventi per cui la proprietà **`cancelable`** restituisce il valore true
- **`stopPropagation`** questo metodo permette di interrompere la propagazione dell'evento, arrestando l'*Event Flow* indipendentemente che sia nella sua fase di capturing o in quella di bubbling

JavaScript
L'oggetto Event: esempio

```
<div id="mainDiv">
    <p id="p1">Clicca su questo paragrafo</p>
    <p id="p2">Altro paragrafo</p>
</div>
…
var myDiv = document.getElementById("mainDiv");
var myP = document.getElementById("p1");
var handler = function() { alert(this.id)};
myDiv.addEventListener("click", handler);
myP.addEventListener("click", handler);
```

Otteniamo prima l'id del primo paragrafo e poi quello del .

JavaScript
L'oggetto Event: esempio

- Per invertire l'ordine di gestione si usa un terzo parametro opzionale del metodo `addEventListener()` che abilita la gestione dell'evento nella fase di capturing

```
myDiv.addEventListener("click", handler, true);
```

- Per fare in modo che venga eseguito solo un gestore dell'evento si blocca la propagazione

```
var handler = function(e) {
    console.log(this.id);
       //this elemento su cui si è verificato l'evento
    e.stopPropagation();
};
```

JavaScript

Drag & drop: eventi

Evento	Tag	Descrizione
ondragstart	tutti gli elementi con attributo draggable="true	si sta trascinando l'elemento
ondragover	tutti gli elementi con attributo draggable="true	quando un elemento viene trascinato su una destinazione di rilascio valida
ondrop	tutti gli elementi con attributo draggable="true"	si è rilasciato un elemento sopra questo

- `.dataTransfer` Restituisce un oggetto contenente i dati trascinati/rilasciati o inseriti/eliminati. Il metodo `dataTransfer.setData("key",value)` imposta una coppia chiave/valore e `getData("key")` restituisce il valore.
- Per impostazione predefinita, gli elementi trascinati non possono essere rilasciati in altri elementi. Per consentire un inserimento, dobbiamo impedire la gestione predefinita dell'elemento trascinato e dell'elemento "ricevente". Si deve chiamare per entrambi il metodo `e.preventDefault()`

JavaScript

Drag & drop: eventi

```
Trascina la scritta nell'altro paragrafo
<script>
function allowDrop(e) {e.preventDefault();}
function drag(e) {
      e.dataTransfer.setData("text",   e.target.id);}
function drop(e) {
    e.preventDefault();
    var data = e.dataTransfer.getData("text");
  e.target.appendChild(document.getElementById(data));
}
</script>

<div id="div1" ondrop="drop(event)"
ondragover="allowDrop(event)"></div>
<div id="drag1" draggable="true"
ondragstart="drag(event)">ciao</div>
```

Approfondimento

Attributi di script

async defer

- Sono utili per ottimizzare il caricamento della pagina evitando stop nel rendering
- Il file HTML viene analizzato fino a quando non viene richiamata una dipendenza (uno <script>), a quel punto l'analisi si ferma ed il browser inizia a scaricare il file (se è esterno). Lo <script> verrà eseguito subito dopo il download, il rendering dell'HTML continuerà soltanto dopo che lo <script> viene eseguito. Per i server lenti e pesanti richiamare <script> in questo modo significa che la visualizzazione della pagina web sarà ritardata.

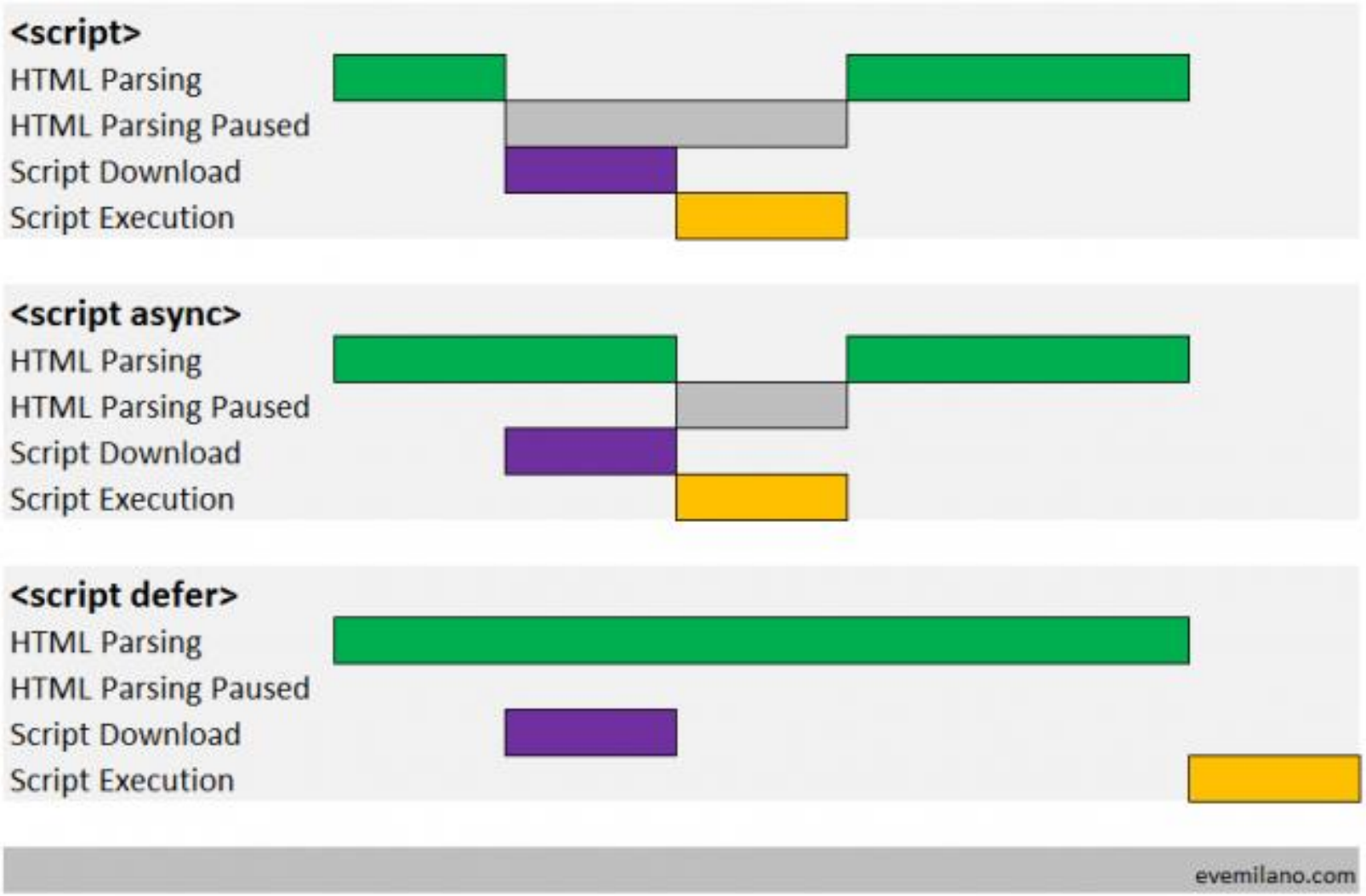
<script async>

- L'attributo async evita di interrompere il rendering dell'HTML durante il download dello <script>, che di fatto avviene in parallelo.
- Lo <script> viene eseguito subito dopo il download.
- Il parsing della pagina viene messo in pausa soltanto mentre lo <script> viene eseguito
- Se non importa quando lo <script> sarà disponibile, usare async, per esempio per gli <script> come Google Analytics. È quello da preferire

<script defer>

- L'attributo defer evita di interrompere il rendering dell'HTML durante il download dello <script>, che di fatto avviene in parallelo
- Lo <script> viene eseguito subito dopo il parsing della pagina HTML
- Il parsing della pagina non viene mai messo in pausa
- Il DOM sarà già pronto per lo <script>.

Confronto


<script>
HTML Parsing
HTML Parsing Paused
Script Download
Script Execution
<script async>
HTML Parsing
HTML Parsing Paused
Script Download
Script Execution
<script defer>
HTML Parsing
HTML Parsing Paused
Script Download
Script Execution
evemilano.com

Quando usarli?

Ecco alcune regole generali da seguire:

- Se lo <script> è modulare e non si basa su altri <script>, sarebbe meglio utilizzare async
- Se lo <script> si basa su altri <script> o viene invocato da un altro <script>, sarebbe meglio utilizzare defer
- Se lo <script> è di piccole dimensioni ed è invocato da uno <script> asincrono, sarebbe meglio utilizzare uno <script> in linea senza attributi posto sopra gli <script> asincroni
- Di solito la libreria jQuery non è un buon candidato per l'attributo async perché altri <script> potrebbero dipendere da quella libreria (molto utilizzata). Puoi usare async ma devi assicurarti che gli altri <script> non vengano eseguiti fino al caricamento di jQuery.

Memorizzazione dei dati

Cookie

- Un **cookie** è un insieme di informazioni salvate sul computer dell'utente, che possono essere riutilizzati in successivi accessi allo stesso sito. L'utente li deve autorizzare.
- La loro dimensione massima è di 4KB e uno stesso dominio non può utilizzarne più di 20
- Non si possono impostare cookie diversi per sessioni contemporanee sullo stesso browser dello stesso sito (due finestre aperte nello stesso dominio)

WebStorage

- Il **webstorage** è disponibile dall'HTML 5, per superare i limiti dei cookie. Possono anch'essi archiviare i dati localmente all'interno del browser dell'utente.
- È più sicuro (le informazioni non vengono mai trasferite al server) e il limite di archiviazione è molto più grande (almeno 5 MB).
- Tutte le pagine della stessa origine (dominio+protocollo), possono memorizzare e accedere agli stessi dati.

Webstorage

- Permette di memorizzare i dati sotto forma di chiavi-valore (valore è una stringa), in due oggetti:
  - **window.localStorage**: dati comuni a tutto il dominio (come i cookie) e non hanno "data di scadenza"
  - **window.sessionStorage:** memorizza i dati per una sessione (i dati vengono persi quando la scheda del browser viene chiusa)
- Prima di utilizzarlo, controllare se è supportato dal browser

```
if (typeof(Storage) !== "undefined") {
    // Code for localStorage/sessionStorage.
} else {   // Sorry! No Web Storage support..}
```

Webstorage localStorage/sessionStorage

- I metodi per gestire le coppie chiave-valore:
  - **setItem("chiave", "valore")** solo stringhe
  - **getItem("chiave")**
  - **removeItem("chiave")** elimina la chiave
  - **clear()** elimina tutte le chiavi
- Al posto del set/getItem si può usare la dot notation

```
if (localStorage.getItem("chiave")) //se !undefined
    localStorage.setItem("chiave",
   Number(localStorage.getItem("chiave"))+1);

if (localStorage.chiave) //se !undefined
   localStorage.chiave= Number(localStorage.chiave)+1;
```

Debugger

Come avviare il debbuger in IE

- **Per abilitare il debug di script in Internet Explorer**
  - Scegliere Opzioni Internet dal menu Strumenti.
  - Nella finestra di dialogo Opzioni Internet scegliere la scheda Avanzate.
  - Nella categoria Esplorazione deselezionare la casella di controllo Disabilita debug degli script.
  - Fare clic su OK.
  - Chiudere e riavviare Internet Explorer.

Come avviare il debbuger in Chrome

- **Per abilitare il debug di script in Chrome**
  - Premere la combinazione di tasti Ctrl+Shift+i

Come avviare il debbuger in Firefox

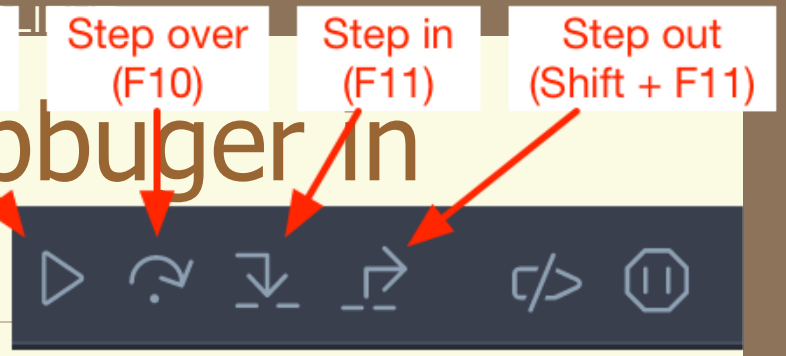
There are three ways to open the debugger:

- select "Debugger" from the Web Developer submenu in the Firefox Menu (or Tools menu if you display the menu bar or are on Mac OS X)
- press the Ctrl+Shift+S (Command Option S on OSX) keyboard shortcut
- press the menu button ( ), press "Developer", then select "Debugger".

Selezionare quindi nella scheda SORGENTI il file da debbuggare

https://developer.mozilla.org/it/docs/Tools/Debugger

Come avviare il debbuger in Firefox



When the debugger is stopped at a breakpoint, you can step through it using four buttons in the toolbar:

- Play: run to the next breakpoint
- Step over: advance to the next line in the same function.
- Step in: advance to the next line in the function, unless on a function call, in which case enter the function being called
- Step out: run to the end of the current function

Si può aggiungere il nome della variabile da controllare nella scheda ESPRESSIONI DI CONTROLLO