



# Lezione 5: JavaScript



# Introduzione

# Applicazioni web

- ▶ Risorsa in rete identificata tramite URL è di tre tipi
  - ▶ Pagina statica (HTML) il cui contenuto è fisso, definito nel momento in cui la pagina è stata scritta
    - ▶ <http://www.example.com/biblioteca.html>
  - ▶ Pagina “dinamica” (per esempio PHP, ASP, o JSP) il cui contenuto viene generato (selezionato, composto) al momento della richiesta (o della visualizzazione)
    - ▶ <http://www.example.com/biblioteca.php>
  - ▶ Indirizzo di un programma (ad es., Java Servlet) il cui compito è quello di generare dinamicamente una pagina Web, in base alla richiesta del client
    - ▶ <http://www.di.unimi.it/exServlet>

# Applicazioni web

- ▶ Applicazione web = software sviluppato e utilizzato attraverso tecnologie web
- ▶ Linguaggi di mark-up, scripting, programmazione
- ▶ Tecnologie client-side e server-side

# Applicazioni web: Linguaggi

- ▶ Linguaggi di mark-up descrivono documenti strutturati (contenuto + struttura)
  - ▶ Abbiamo già visto HTML e CSS
- ▶ Linguaggi di programmazione descrivono programmi come sequenza di istruzioni
- ▶ Linguaggi di scripting, tipo particolare di linguaggio di programmazione (ad es., PHP, JavaScript), per scrivere script
  - ▶ Script è un piccolo programma che inseriamo nella pagina web

# Applicazioni web: linguaggi di mark-up

- ▶ Definiscono la struttura e il contenuto (HTML)
- ▶ Definiscono l'aspetto/formattazione (CSS)
- ▶ Producono pagine web statiche interpretate dal browser

# Applicazioni web: linguaggi di programmazione e scripting

- ▶ Programma = insieme di istruzioni
- ▶ Linguaggi di programmazione = linguaggi per scrivere programmi (istruzioni)
- ▶ Linguaggio
  - ▶ Alfabeto: insieme di simboli con cui si possono costruire i termini del linguaggio (lessico)
  - ▶ Sintassi: definita da una grammatica che fornisce le regole di composizione dei termini in frasi ben formate del linguaggio
  - ▶ Semantica: definisce il significato delle frasi ben formate del linguaggio
- ▶ Analizzatore sintattico (parser): analizza frasi e decide se sono frasi ben formate del linguaggio o no
- ▶ Linguaggio di programmazione:
  - ▶ Lessico = "keywords" del linguaggio
  - ▶ Frasi ben formate = istruzioni (programmi)
  - ▶ Semantica = esecuzione del programma

# Applicazioni web: linguaggi di programmazione e scripting

- ▶ Il programmatore scrive il programma sorgente utilizzando un linguaggio "ad alto livello"
- ▶ Occorre tradurre il programma sorgente in un programma composto da istruzioni in linguaggio macchina (programma eseguibile)
- ▶ Tecniche per effettuare questa traduzione
  - ▶ Compilazione: traduzione effettuata da strumenti chiamati compilatori (ad es., C)
  - ▶ Interpretazione: traduzione effettuata da strumenti chiamati interpreti (ad es., JavaScript, Ruby)
  - ▶ Approccio misto (ad es., Java, Python)



# Applicazioni web: linguaggi di programmazione e scripting

- ▶ Traduzione eseguita da compilatore
  - ▶ Specifica per una data macchina
    - ▶ Tradurre il programma sorgente nel linguaggio macchina della macchina specifica
  - ▶ Genera un file eseguibile (.exe)
  - ▶ Vantaggi: efficiente (esecuzione veloce)
  - ▶ Svantaggi: poco flessibile (per eseguire il programma su macchine diverse è necessario ri-compilare i sorgenti)

# Applicazioni web: linguaggi di programmazione e scripting

- ▶ Traduzione eseguita da interprete
  - ▶ Interpreta un programma (traduzione simultanea)
  - ▶ Ogni istruzione viene tradotta in un insieme di istruzioni nel linguaggio macchina della piattaforma specifica ed eseguita
  - ▶ Interprete specifico per la piattaforma
  - ▶ Vantaggi: flessibile (il sorgente è direttamente eseguibile su macchine diverse, dato l'interprete)
  - ▶ Svantaggi: poco efficiente (esecuzione lenta)

# Applicazioni web: linguaggi di programmazione e scripting

- ▶ Approccio misto
  - ▶ Programma sorgente tradotto in formato intermedio (bytecode) indipendente dalla macchina (compilatore)
  - ▶ Programma in formato intermedio viene interpretato (interprete)
- ▶ L'approccio misto somma i vantaggi dei due approcci
- ▶ Efficiente (il linguaggio intermedio è “molto vicino” al linguaggio macchina e la sua interpretazione è veloce)
- ▶ Flessibile (perché posso eseguire il programma compilato su macchine diverse)

# Applicazioni web

- ▶ Pagine Web "dinamiche"
  - ▶ Il contenuto viene generato al momento della richiesta/visualizzazione
- ▶ Si basano su linguaggi di programmazione e scripting
  - ▶ Tecnologie client-side (pagine "debolmente" dinamiche)
    - ▶ Elaborazione lato client (browser)
    - ▶ Scripting client-side (JavaScript), Java Applet, Adobe Flash
  - ▶ Tecnologie server-side (pagine "autenticamente" dinamiche)
    - ▶ Elaborazione lato server (web server)
    - ▶ Scripting server-side (PHP, Active Server Pages, Java Server Page), programmi (Java Servlet), Ruby, Python, Perl, (JavaScript)

# Applicazioni web

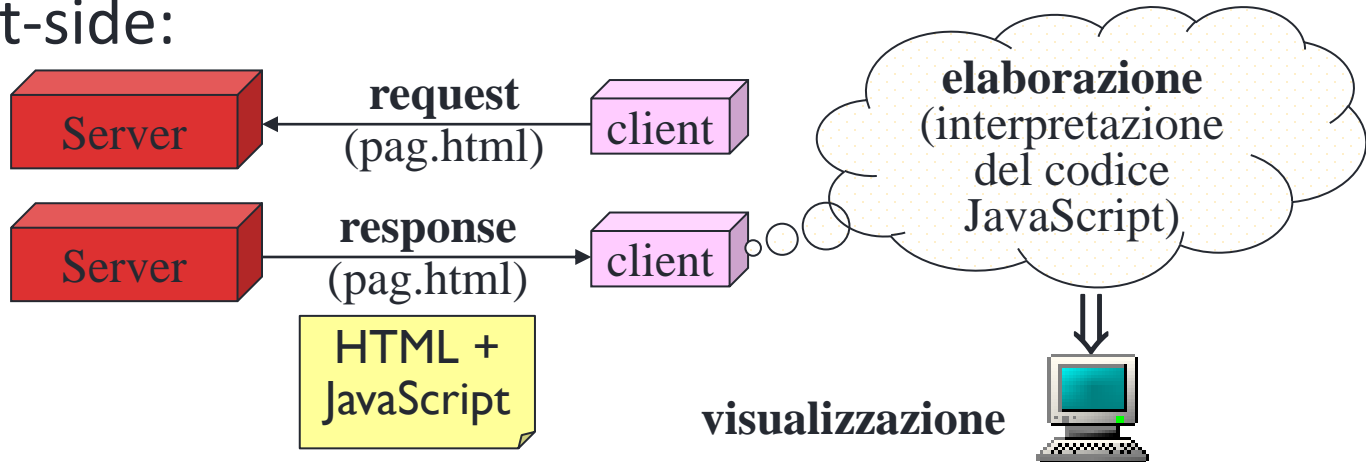
- ▶ Si sviluppano su tre livelli logici
  - ▶ Presentazione – HTML, CSS
  - ▶ Intermedio – JavaServlet, JSP, PHP, ASP, JavaScript, Flash (XSLT)
  - ▶ Dati – RDBMS, XML, JSON

# Applicazioni web

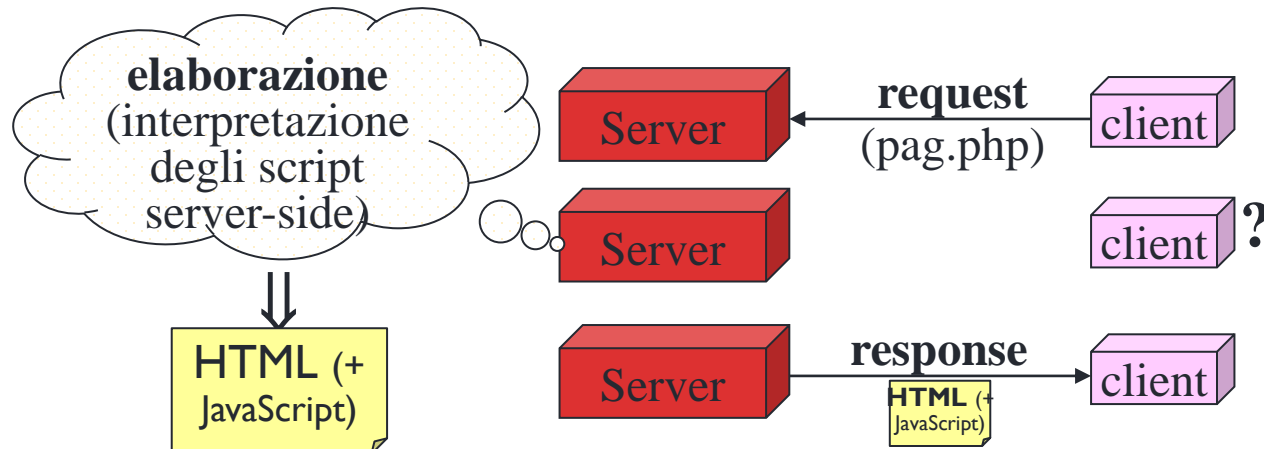
- ▶ Si sviluppano su tre livelli logici
  - ▶ Presentazione – HTML, CSS
  - ▶ Intermedio – JavaServlet, JSP, PHP, ASP, JavaScript, Flash (XSLT)
  - ▶ Dati – RDBMS, XML, JSON

# Applicazioni web

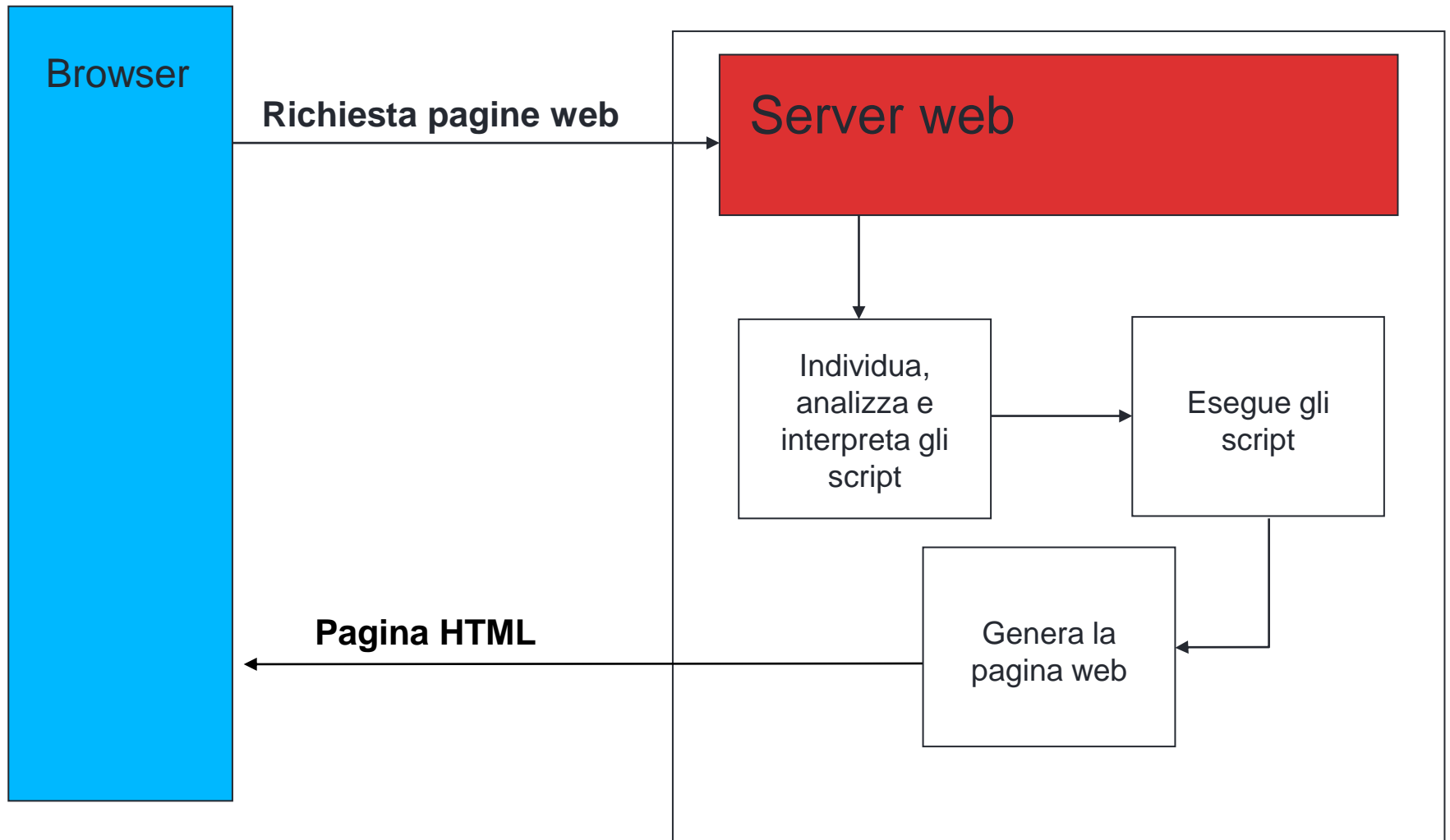
## ▶ Client-side:



## ▶ Server-side:

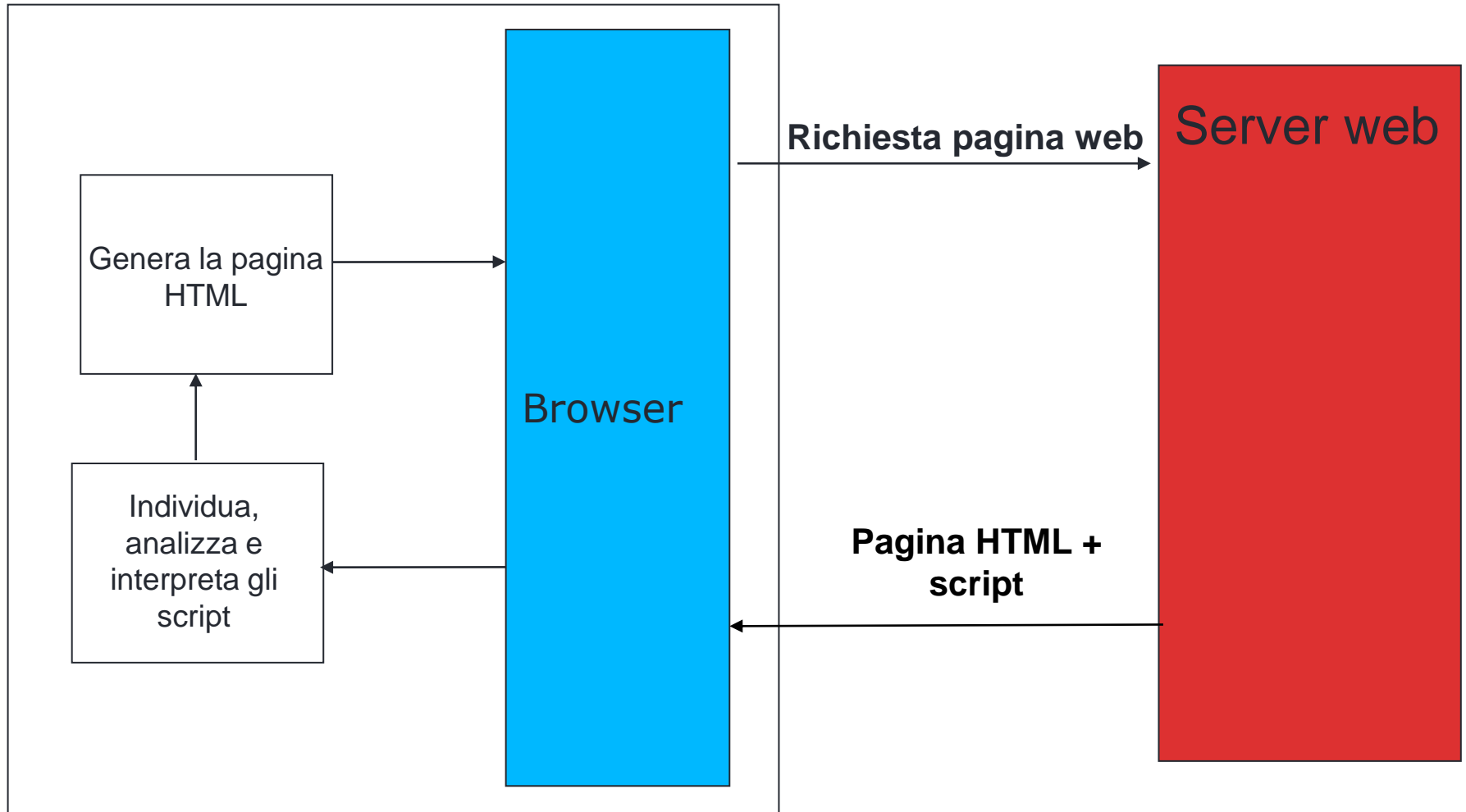


# Architettura server-side

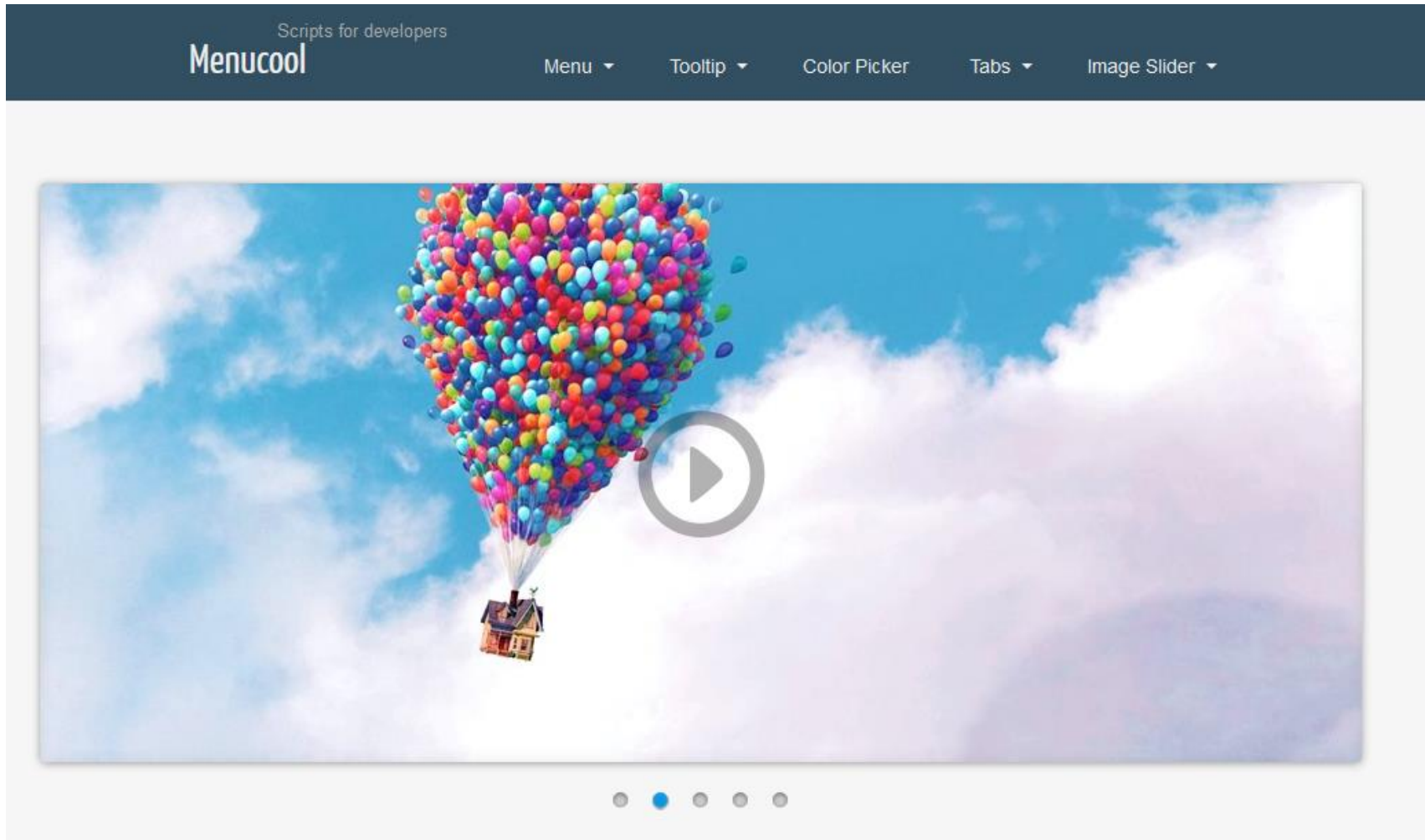




# Architettura client-side



# Pagine dinamiche client-side: esempi



# Pagine dinamiche server-side: esempi



html5 + css3



[Web](#) [Images](#) [Videos](#) [News](#) [Books](#) [More ▾](#) [Search tools](#)

About 22,000,000 results (0.67 seconds)

## HTML5 Introduction - W3Schools

[www.w3schools.com/html/html5\\_intro.asp](http://www.w3schools.com/html/html5_intro.asp) ▾

Free HTML CSS JavaScript DOM jQuery XML AJAX Angular ASP .NET PHP SQL ... The DOCTYPE declaration for HTML5 is very simple: <!DOCTYPE html>.

## Beginner's Guide To: Building HTML5/CSS3 Webpages ...

[www.hongkiat.com/blog/building-html5-css-webpages/](http://www.hongkiat.com/blog/building-html5-css-webpages/) ▾

HTML5 and CSS3 have swept the web by storm in only 2 years. Before them there have been many altered semantics in the way web designers are expected to ...

[Inspiring HTML5 Experiments](#) - [Beginner's Guide to CSS3](#) - [HTML5-Powered Web ...](#)

## Images for html5 + css3

[Report images](#)



[More images for html5 + css3](#)

## Learn HTML5 and Get CSS Training – Microsoft Virtual ...

[www.microsoftvirtualacademy.com/.../html5-css3-fundamentals-develop...](http://www.microsoftvirtualacademy.com/.../html5-css3-fundamentals-develop...) ▾

★★★★★ Rating: 5 - 502 votes

Aug 8, 2013 - Get ready to learn HTML5 and CSS3 with our targeted training for absolute beginners. This training is led by an expert and is offered free of ...

## Free Responsive Html5 and Css3 Templates

[www.html5xcss3.com/](http://www.html5xcss3.com/) ▾

Free high quality html5 templates that you can use to build a website : free html5 template, free responsive themes, css3 templates, css3 menus...

[Cyprass Bootstrap Responsive ...](#) - [Best Html5 Responsive ...](#) - [Css3 Menu Tutorials](#)

## Shop for html5 + css3 on Google

Sponsored ⓘ



HTML 5 e CSS 3

€24.65

[Amazon.it](#)



HTML5 und CSS3: Das ...

€38.78

[Amazon.it](#)



Murach's HTML5 and ...

€50.68

[Amazon.it](#)



HTML5 e CSS3. Per il ...

€38.17

[Amazon.it](#)



HTML5 e CSS3. Guida ...

€29.67

[Amazon.it](#)



HTML 5 e CSS 3

€24.65

[Mondadori St...](#)



HTML5 and CSS3 All-in- ...

€38.04

[Amazon.it](#)



Beginning HTML5 and ...

€31.15

[Amazon.it](#)

Ad

## SCADA Software

[webaccess.advantech.com/product.php](http://webaccess.advantech.com/product.php) ▾

SCADA Software, Industry 4.0  
iFactory, Monitorign System

[See your ad here »](#)

# Applicazioni web

- ▶ Per visualizzare una pagina Web "debolmente" dinamica (che utilizza una tecnologia client-side) NON HO bisogno di un server
  - ▶ Posso aprire la pagina fornendo al browser il path sul file system locale
- ▶ Per visualizzare una pagina Web autenticamente dinamica (che utilizza una tecnologia server-side) HO bisogno di un server
  - ▶ Devo connettermi al server (e richiedere la pagina) tramite un URL

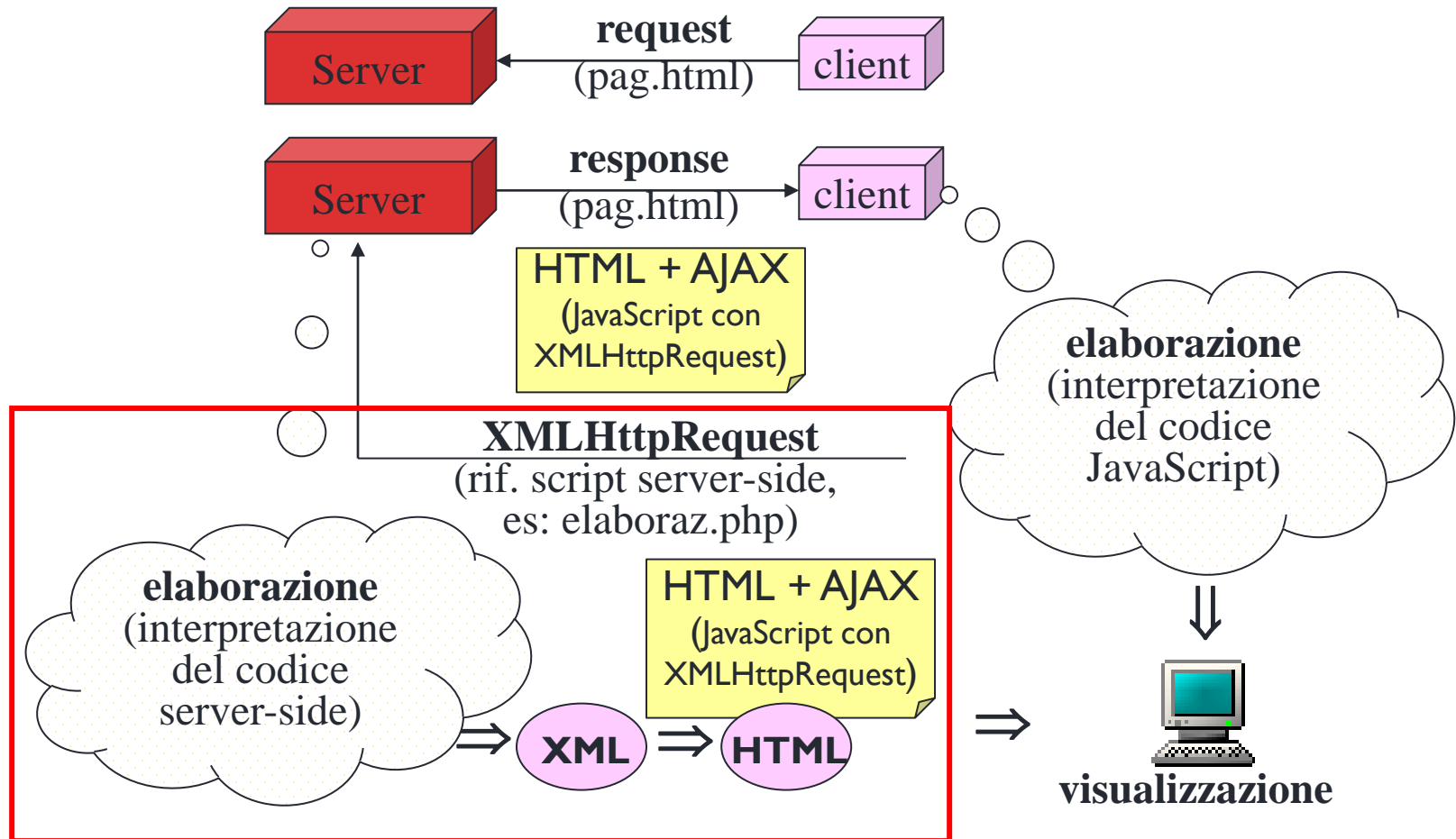
# Applicazioni web

- ▶ Se chiedo al browser di visualizzare il codice sorgente della pagina...
  - ▶ Nel caso di una pagina Web "debolmente" dinamica (che utilizza una tecnologia client-side) vedo l'HTML + il codice "dinamico" client-side (ad es., JavaScript)
  - ▶ Nel caso di una pagina Web autenticamente dinamica (che utilizza una tecnologia server-side) vedo solo l'HTML: al posto del codice "dinamico" server-side (ad es., PHP), il server ha infatti sostituito il risultato dell'elaborazione (cioè codice HTML)

# Applicazioni web

- ▶ Esistono approcci ibridi
  - ▶ Utilizzano sia tecnologie client-side che tecnologie server-side
  - ▶ Esempi: AJAX, JQuery

# Pagine Web dinamiche: AJAX



richiesta **asincrona** (non "blocca"  
l'interazione con l'utente)



# JavaScript: Fondamenti



# JavaScript

- ▶ JavaScript è un linguaggio di scripting, tipicamente utilizzato client-side
- ▶ Nonostante la somiglianza nel nome, è un linguaggio completamente distinto da Java
- ▶ Come tutti i linguaggi di scripting, è interpretato
  - ▶ Il sorgente non deve essere compilato per essere eseguito
- ▶ L'interprete di JavaScript è generalmente contenuto all'interno del browser

# JavaScript

## ▶ Storia

- ▶ Definito da Netscape (LiveScript)
- ▶ Nome modificato in JavaScript dopo accordo con Sun nel 1995
- ▶ Microsoft lo chiama JScript (differenze minime)
- ▶ Standard di riferimento: ECMAScript 262

# JavaScript

- ▶ A differenza di HTML, JavaScript è case-sensitive
  - ▶ "ciao" è diverso da "Ciao"
- ▶ Purtroppo possono esserci incompatibilità e differenze tra i diversi browser
  - ▶ A volte si comportano in maniera diversa o non funzionano
- ▶ Si basa su due concetti principali:
  - ▶ DOM (Document Object Model)
  - ▶ Eventi (script event-driven)

# Tipi

- ▶ JavaScript è un linguaggio debolmente tipato
  - ▶ Il tipo delle variabili (e dei parametri/argomenti delle funzioni) non viene dichiarato esplicitamente, ma definito implicitamente al primo assegnamento
  - ▶ `numero = 7;` `numero` è di tipo `Number`
- ▶ JavaScript converte automaticamente i tipi durante l'esecuzione (quando possibile)
- ▶ Tipi principali in JavaScript
  - ▶ `Number`: interi e decimali (virgola mobile); ad es: `7`, `7.7`
  - ▶ `Boolean`: valori booleani (vero/falso); ad es: `true`, `false`
  - ▶ `String`: sequenze di caratteri; ad es: `"ciao"`

# Variabili e istruzioni

- ▶ Dichiarazioni delle variabili
  - ▶ Non è obbligatorio l'uso della keyword `var`
  - ▶ Sono senza tipo
  - ▶ `var variabile;`
- ▶ Dichiarazioni delle variabili locali
  - ▶ È obbligatorio l'uso della keyword `var`
  - ▶ Sono senza tipo
  - ▶ `var prezzo_scontato;`
- ▶ Le inizializzazioni sono facoltative (ma è buona norma farle all'atto della definizione e prima di utilizzare le variabili)
- ▶ Tutte le istruzioni JavaScript devono terminare con punto-e-virgola

# Variabili

- ▶ Loosely typed
  - ▶ È possibile assegnare a una stessa variabile prima un valore stringa, poi un numero, poi altro ancora
- ▶ Ad esempio
  - ▶ `alfa = 10`
  - ▶ `beta = "Pippo"`
  - ▶ `alfa = "Erika" // tipo diverso!!`
- ▶ Sono consentiti incrementi, decrementi e operatori di assegnamento estesi (`++`, `--`, `+=`, ... )

# Variabili e Scope

- ▶ Due possibili scope
  - ▶ Globale, per le variabili definite fuori da funzioni
  - ▶ Locale, per le variabili definite esplicitamente dentro a funzioni (compresi i parametri ricevuti)
- ▶ **ATTENZIONE:** un blocco **NON** delimita uno scope!
  - ▶ Tutte le variabili definite fuori da funzioni, anche se dentro a blocchi innestati, sono globali
- ▶ `x = '3' + 2; // la stringa '32'`  
`{`  
`{ x = 5 } // blocco interno`  
`y = x + 3; // x denota 5, non "323"`  
`}`

# Tipo dinamico

- ▶ Operatore typeof ritorna il tipo di una espressione
  - ▶ Risolve le variabili incluse
    - ▶ `typeof(10/2) = number`
    - ▶ `typeof("stringa") = string`
    - ▶ `typeof(false) = boolean`
    - ▶ `typeof(document) = object`
    - ▶ `typeof(document.write) = function`
- ▶ Il tipo è dinamico: rappresenta il tipo in quel momento temporale e corrispondente al valore attuale della variabile (o dell'oggetto...)
  - ▶ `variabile = 10;`
    - ▶ `typeof(variabile) = number`
  - ▶ `variabile = "pippo";`
    - ▶ `typeof(variabile) = string`



# Istruzioni

- ▶ Devono essere separate da un fine riga o da un punto e virgola (simile al Pascal)
- ▶ Istruzione1 // fine riga  
istruzione2; istruzione3  
istruzione4

# Costanti e Commenti

- ▶ Le costanti numeriche sono sequenze di caratteri numerici non racchiuse da virgolette o apici
  - ▶ Tipo number
- ▶ Le costanti booleane sono true e false
  - ▶ Tipo boolean
- ▶ Altre costanti sono null, NaN e undefined
  - ▶ undefined indica un valore indefinito
- ▶ I commenti in JavaScript sono come in Java:
  - ▶ // commento su riga singola
  - ▶ /\* commento su + righe  
commento su + righe  
commento su + righe \*/

# Operatori e commenti

- ▶ Aritmetici: +, -, \*, /, ++, --
- ▶ Di confronto
  - ▶ ==, != (numeri e stringhe)
  - ▶ >, >=, <, <= (numeri)
  - ▶ === (fa anche il controllo del tipo)
- ▶ Booleani: && (AND), || (OR), ! (NOT)
- ▶ Concatenazione (di stringhe): +
- ▶ Assegnamento: =

# Espressioni

- ▶ Espressioni simili a quelle Java
  - ▶ Espressioni numeriche: somma, sottrazione, prodotto, divisione (sempre fra reali), modulo, shift, ...
  - ▶ Espressioni condizionali con `? ... :`
  - ▶ Espressioni stringa: concatenazione con `+`
  - ▶ Espressioni di assegnamento: con `=`
- ▶ Esempi:
  - ▶ `document.write(a/b)`
  - ▶ `document.write(a%b)`
  - ▶ `document.write("a" + 'b')`

# Condizioni booleane

- ▶ Una condizione booleana è un'espressione che ha valore vero (true) o falso (false)
- ▶ Le condizioni booleane sono espressioni composte da
  - ▶ Costanti
  - ▶ Variabili
  - ▶ Operatori di confronto
  - ▶ Operatori logici
- ▶ Ad esempio
  - ▶  $3 > 5$  false
  - ▶  $3 < 5$  true
  - ▶  $x == y$  [dato  $x=33.3$  e  $y=20.7$ ] false
  - ▶  $x == y$  [dato  $x="Pippo"$  e  $y="PIPPO"$ ] false
  - ▶  $z \ \&\& \ (x \leq y)$  [dato  $z=true$ ,  $x=10$ ,  $y=10$ ] true
  - ▶  $!z \ || \ (x \neq y)$  [dato  $z=true$ ,  $x=10$ ,  $y=12$ ] true

# Stringhe

- ▶ Delimitate sia da virgolette sia da apici singoli
- ▶ Per annidare virgolette e apici, occorre alternarli
  - ▶ `document.write('<IMG src="image.gif">')`
  - ▶ `document.write("<IMG src='image.gif">")`
- ▶ Concatenazione con +
  - ▶ `document.write("paolino" + 'paperino')`
  - ▶ Concatenazione fra stringhe e numeri comporta la conversione automatica del valore numerico in stringa
- ▶ Le stringhe JavaScript sono oggetti dotati di proprietà (ad es., `length`) e metodi (ad es., `substring(first,last)`)

# Specifica dello script

- ▶ Il codice di un programma JavaScript viene incluso in un file HTML per mezzo del tag `<SCRIPT>`
  - ▶ Sono possibili più programmi nella stessa pagina
  - ▶ Una pagina HTML può contenere più tag `<script>`

- ▶ `<BODY>`

```
...  
<SCRIPT language="JavaScript">  
    prima istruzione;  
    seconda istruzione;  
    terza istruzione;  
    ...  
</SCRIPT>
```

```
...  
</BODY>
```

- ▶ L'interprete HTML lo invia all'interprete JavaScript

# Definizione funzione

- ▶ Le definizioni delle funzioni vengono generalmente incluse nella sezione <HEAD>... <HEAD> (funz\_lordo.html)

- ▶ <HEAD>

```
...  
<SCRIPT language="JavaScript">  
    Definizione della funzione f  
</SCRIPT>  
</HEAD>
```

- ▶ I richiami alle funzioni (predefinite nel linguaggio o definite dal programmatore) avvengono dove occorre, nel body della pagina HTML

- ▶ <BODY>

```
...  
<SCRIPT language="JavaScript">  
     $y = f(x);$   
</SCRIPT>  
...  
</BODY>
```



# Esempio funzione

- ▶ Definizione della funzione (funz\_lordo.html):
  - ▶ <HEAD>
    - ...

```
<SCRIPT language="JavaScript">  
  function lordo(netto, tara) {  
    var risultato = 0;  
    risultato = Number(netto) + Number(tara);  
    return risultato;  
  }  
</SCRIPT>
```
    - ...

```
</HEAD>
```
  - ▶ L'interprete valuta l'espressione e restituisce il valore contenuto in risultato
    - ▶ Parametri formali: netto, tara
    - ▶ Parametro di ritorno: risultato
    - ▶ Keyword di definizione della funzione: function

# Chiamata funzione

- ▶ Invocazione della funzione:

- ▶ <BODY>

...

```
<SCRIPT language="JavaScript">
```

```
  var netto_in = prompt("Inserire il peso netto", "");
```

```
  var tara_in = prompt("Inserire la tara", "");
```

```
  var ris = lordo(netto_in,tara_in);
```

```
  document.write("<h1>Prezzo scontato:" + ris + "</h1>");
```

```
</SCRIPT>
```

...

```
</BODY>
```

# Costrutto If-Else

- ▶ Costrutto if-else per esprimere un'azione condizionale

```
▶ if (condizione1) {  
    sequenza_di_azioni_1  
}  
else if (condizione2) {  
    sequenza_di_azioni_2  
}  
...  
else {  
    sequenza_di_azioni_n  
}
```

# Esempio

- ▶ Riprendiamo l'esempio del calcolo del peso lordo (form\_lordo2.html)
- ▶ 

```
<SCRIPT language="JavaScript">  
    function lordo(netto, tara) {  
        var risultato = 0;  
        if (tara!=0)  
            {risultato = Number(netto) + Number(tara);  
            return risultato;}  
        else {return "lordo=netto"}  
    }  
</SCRIPT>
```

# Liste

- ▶ Sequenza ordinata di elementi
  - ▶ Elenco di link in una pagina Web
  - ▶ Elenco degli iscritti alle liste elettorali
  
- ▶ Operazioni
  - ▶ Calcolo lunghezza
  - ▶ Stampa di tutti gli elementi
  - ▶ Aggiunta di un elemento
  - ▶ Cancellazione di un elemento

# Array

- ▶ Lista rappresentata come array con indice a partire da 0
  - ▶ Le celle di un array JavaScript non hanno il vincolo di omogeneità in tipo: ogni cella può contenere indistintamente numeri, stringhe, oggetti, altri array, ...
- ▶ Creazione e inizializzazione di un array
  - ▶ Array vuoto
    - ▶ `var lista = new Array();`
    - ▶ `lista[0] = "Pippo";`
    - ▶ ...
  - ▶ Array inizializzato in fase di definizione
    - ▶ `lista = new Array("Pippo", "Erika", "Denise");`

# Array

- ▶ Inserimento valori
  - ▶ I singoli elementi sono referenziati con l'usuale notazione a parentesi quadre: ad esempio, lista[x]
  - ▶ lista[i] = "stringa";
  
- ▶ Lettura contenuto in posizione i
  - ▶ var elem = lista[i];

# Array

- ▶ Lunghezza di in un array (attributo length)
  - ▶ `var lunghezza = lista.length;`
- ▶ Per (sovra)scrivere l'ultimo elemento dell'array:
  - ▶ `lista[lunghezza-1]="stringa";`
  - ▶ Ogni scrittura sovrascrive l'elemento che era memorizzato in precedenza
- ▶ Per leggere l'ultimo elemento dell'array
  - ▶ `var elem = lista[lunghezza-1];`



# Array - Costruzione Alternativa

- ▶ A partire da JavaScript 1.2, anche per gli array esiste un modo alternativo di costruzione: basta elencare la sequenza, racchiusa fra parentesi quadre, di valori iniziali separati da virgole
  - ▶ `vett = [ 1, -2, "tre" ]`

# Cicli

- ▶ Strutture di controllo per l'accesso sequenziale a un set di elementi (ad es., lista)
- ▶ I cicli sono generalmente basati sul concetto di indice (i): l'indice scorre lungo la lista indicando, via via, posizioni successive
- ▶ JavaScript supporta for, while, do/while, for... in..., with

# Ciclo for

- ▶ `for (inizio; test; incremento) {  
    istruzioni  
}`
- ▶ Istruzioni eseguite a partire da inizio, finché test è vero, avanzando ad ogni passo di quanto è indicato da incremento
  - ▶ Inizio = la posizione iniziale dell'indice
  - ▶ Test = condizione che vera (true) fa sì che il ciclo prosegua; falsa (false) provoca l'uscita dal ciclo
  - ▶ Incremento = incremento dell'indice ad ogni ciclo

# Ciclo for: Esempio

- ▶ For\_img.html

```
<SCRIPT language=JavaScript>
```

```
    var images = new Array();
```

```
    images[0] = "figure1.jpg";
```

```
    images[1] = "figure2.jpg";
```

```
    images[2] = "figure3.jpg";
```

```
    for (var i=0; i<images.length; i++) {
```

```
        document.write("<p><img src='"+images[i]+'></p>");
```

```
    }
```

```
</SCRIPT>
```

# Ciclo for: Esempio

- ▶ Quando l'interprete incontra il ciclo (l'istruzione for) per la prima volta
  - ▶ Inizializza l'indice (`var i=0;`)
  - ▶ Valuta il test (`i<images.length`) `0<3 true`
  - ▶ Esegue le istruzioni
  - ▶ Incrementa l'indice (`i++`)
  - ▶ Ripete il ciclo
- ▶ Quando l'interprete ripete il ciclo (incontra l'istruzione for per la seconda, terza, ... volta):
  - ▶ Valuta il test (`i<images.length`) `1<3 true`
  - ▶ Esegue le istruzioni
  - ▶ Incrementa l'indice (`i++`)
  - ▶ Ripete il ciclo
- ▶ Finché... (uscita dal ciclo)
  - ▶ Valuta il test (`i<images.length`) `3<3 false`
  - ▶ Si ferma (prosegue con l'istruzione successiva al for)

# Ciclo for: Esempio

- ▶ `document.write("<p><img src='images[i]'/>"</p>");`
  - ▶ Passo `i=0`: `document.write("<p><img src='images[0]'/>"</p>");`
    - ▶ Scrive `<p><img src='figure1.jpg'/></p>`
  - ▶ Passo `i=1`: `document.write("<p><img src='images[1]'/>"</p>");`
    - ▶ Scrive `<p><img src='figure2.jpg'/></p>`
  - ▶ Passo `i=2`: `document.write("<p><img src='images[2]'/>"</p>");`
    - ▶ Scrive `<p><img src='figure3.jpg'/></p>`

# Ciclo for: Esempio array

- ▶ È possibile aggiungere elementi dinamicamente a un array e stamparli usando un ciclo for

- ▶ `lista = new Array("Pippo", "Erika", "Denise")`

...

```
lista[3] = "Nino";
```

```
for (i=0; i<lista.length; i++)
```

```
    document.write(lista[i] + " ")
```

# Ciclo while

- ▶ while (condizione) {  
    istruzioni  
}
- ▶ Finché la condizione è vera esegui le istruzioni



# Ciclo while

- ▶ Riscriviamo il ciclo for usando il while
  - ▶ while1\_images.html

```
<SCRIPT language="JavaScript">
  var images = new Array();
  images[0] = "figure1.jpg";
  images[1] = "figure2.jpg";
  images[2] = "figure2.jpg";
  var i=0;
  while (i<images.length) {
    document.write("<p><img src='"+images[i]+'></p>");
    i++;
  }
</SCRIPT>
```

# Ciclo while

- ▶ While richiede di inizializzare l'indice prima del ciclo:
  - ▶ `var i=0;`
- ▶ Quando l'interprete incontra il ciclo
  - ▶ Valuta il test (`i<images.length`) `0<3 true`
  - ▶ Esegue le istruzioni
    - ▶ L'incremento dell'indice (`i++`), nel caso del `while`, deve essere l'ultima istruzione del ciclo
  - ▶ Ripete il ciclo
- ▶ Finché... (uscita dal ciclo)
  - ▶ Valuta il test (`i<images.length`) `3<3 false`
  - ▶ Si ferma (prosegue con l'istruzione successiva al `while`)
- ▶ Se la condizione di entrata nel ciclo è sempre vera, il ciclo non termina (loop)
- ▶ Altro esempio `while_pwd.html`

# Cicli for e while

- ▶ For e while si equivalgono
- ▶ Per scorrere una lista si può usare l'uno o l'altro
  - ▶ Generalmente più semplice il for
- ▶ Il while è più versatile e può essere usato per scopi diversi dalla gestione delle liste (ad esempio quando non si conosce il numero di cicli)



# Oggetti DOM, eventi, finestre, nodi

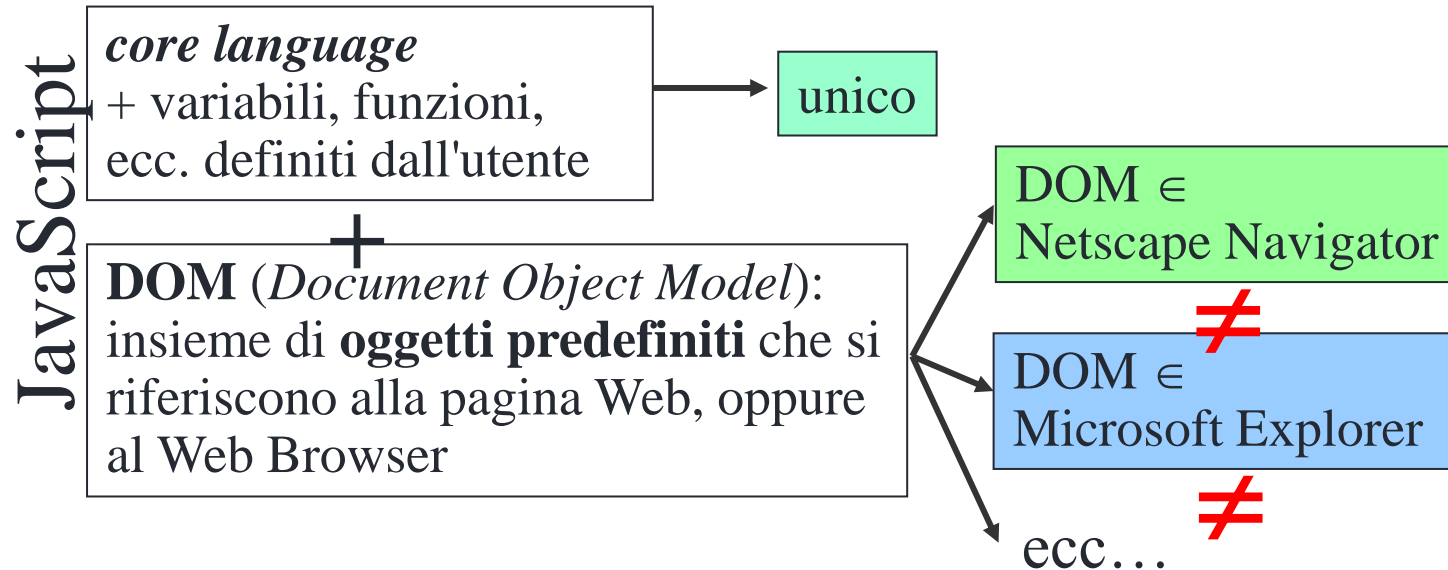
# Document Object Model (DOM)

- ▶ È uno standard W3C (World Wide Web Consortium)
- ▶ Definisce uno standard per accedere documenti
  - ▶ *"The W3C Document Object Model (DOM) is a platform and language-neutral interface that allows programs and scripts to dynamically access and update the content, structure, and style of a document."*
- ▶ Separato in tre parti
  - ▶ Core DOM – modello standard per tutti i tipi di documenti
  - ▶ XML DOM – modello standard per documenti XML
  - ▶ HTML DOM – modello standard per documenti HTML

# HTML DOM

- ▶ È uno standard object model e programming interface per HTML
- ▶ Definisce
  - ▶ Elementi HTML come oggetti
  - ▶ Proprietà degli elementi HTML
  - ▶ I metodi per accedere agli elementi HTML
  - ▶ Gli eventi per tutti gli elementi HTML
- ▶ In altre parole, è uno standard che definisce come ottenere, cambiare, aggiungere o modificare elementi HTML

# HTML DOM



- ▶ JavaScript usa HTML DOM per modificare tutti gli elementi di una pagina web
  - ▶ Quando una pagina è caricata il browser crea il DOM della pagina
  - ▶ Pagina rappresentata come un albero
  - ▶ DOM è definito dal browser
    - ▶ Definizione è fatta separatamente per Explorer, Mozilla, ...
    - ▶ Possono nascere incompatibilità

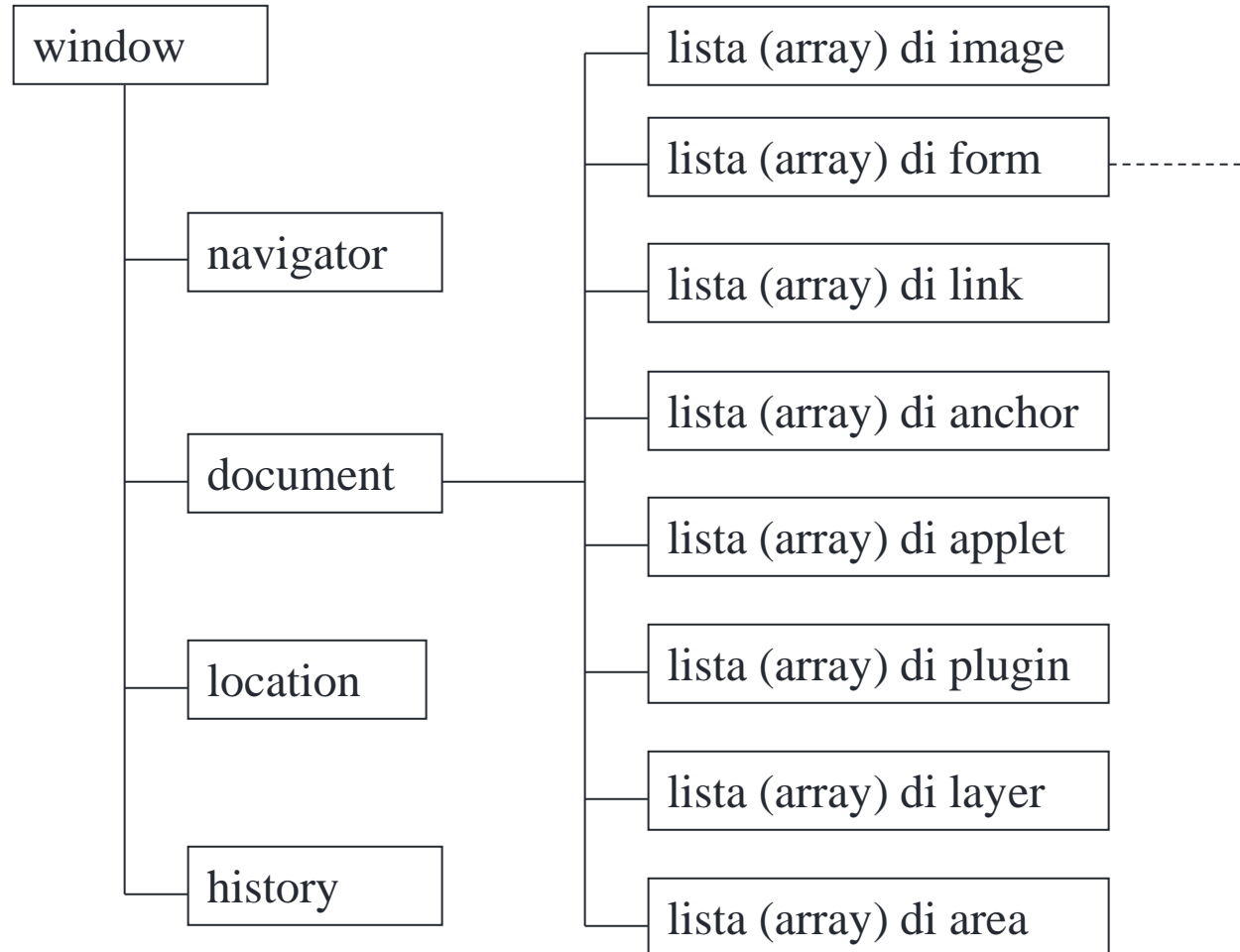
# Oggetti (DOM)

- ▶ Tramite il modello DOM a oggetti, JavaScript può
  - ▶ Cambiare tutti gli elementi e attributi HTML di una pagina
  - ▶ Cambiare tutti gli stili CSS
  - ▶ Rimuovere elementi e attributi HTML
  - ▶ Aggiungere elementi e attributi HTML
  - ▶ Reagire a eventi nella pagina
  - ▶ Creare nuovi eventi



# Oggetti (DOM): Liste di oggetti

- ▶ Il DOM è organizzato secondo una gerarchia:



# Oggetti (DOM)

window

document

Sito di Scienze della Comunicazione - Microsoft Internet Explorer

Indirizzo <http://www.scidecom.unito.it/gestioneAppelli/appelliStudenti.asp>

Scienze della Comunicazione  
Università di Torino

Home Corsi di laurea Didattica Per le aziende

**Iscrizione agli appelli**

Per accedere al servizio bisogna essere utenti registrati.  
Se sei già registrato, digita il tuo numero di matricola e la password.  
Altrimenti puoi [registrarti ora](#)

**Matricola**

**Password**

entra

[Per ulteriori informazioni](#)

Torino Segreteria Scidecom  
Via Verdi 25  
Tel 011 883040  
Fax: 011 8122173

Ivrea (TO)  
Segreteria Scidecom Ivrea  
Ex Officine H--Via Monte navale 11  
Tel 0125 627665  
Fax: 0125 421728

prima immagine: images[0]

prima form: forms[0]

seconda immagine: images[1]

Operazione completata

Internet

# Oggetti (DOM): Window

- ▶ Window (this) = la finestra corrente del browser
- ▶ Figli dell'oggetto window
  - ▶ navigator = il browser (in quanto applicazione)
    - ▶ Ad esempio, per sapere quale browser si sta utilizzando (Firefox/Chrome)
  - ▶ document = il contenuto della finestra
  - ▶ location = informazioni sull'URL corrente
    - ▶ Ad esempio, per caricare nella finestra un URL differente
  - ▶ history = elenco degli URL visitati di recente
    - ▶ Ad esempio, per tornare alla pagina Web precedente

# Window: componenti principali (estesi)

- ▶ self
- ▶ window
- ▶ parent
- ▶ top
- ▶ navigator
  - ▶ plugins (array), navigator, mimeTypees (array)
- ▶ frames (array)
- ▶ location
- ▶ history
- ▶ document
  - ▶ ...segue intera gerarchia di sotto-oggetti...

# Window

- ▶ Radice della gerarchia DOM
  - ▶ Rappresenta la finestra del browser
  - ▶ Fornisce diversi metodi
- ▶ Metodo Alert: `window.alert(messaggio)` propone una finestra di avviso con messaggio
  - ▶ `x = 2; y = 3;`  
`window.alert("moltiplicazione di " + x + " e " + y + ": " + (x*y));`
  - ▶ `window.alert=this.alert=alert`
  - ▶ `alert.html` e `alert_tre_formati.html`
- ▶ La funzione `alert` è usabile anche in un link e non restituisce nessun valore

# Window

- ▶ Altri metodi
  - ▶ `confirm`, che fa apparire una finestra di conferma con il messaggio dato
    - ▶ Restituisce `true` o `false`
    - ▶ `confirm.html`
  - ▶ `prompt`, che fa apparire una finestra di dialogo per immettere un valore
    - ▶ Restituisce il valore string
    - ▶ `prompt.html`

# Document

- ▶ Rappresenta il documento corrente: il contenuto della pagina web attuale
  - ▶ Da non confondere con la finestra corrente!
- ▶ Componenti principali di document
  - ▶ forms (array) = lista dei moduli (form) contenute nella pagina
  - ▶ elements (array di Buttons, Checkbox, etc etc...)
  - ▶ anchors (array)
  - ▶ links (array)
  - ▶ images (array) = lista delle immagini contenute nella pagina
  - ▶ applets (array)
  - ▶ embeds (array)

# Document

- ▶ Diversi metodi disponibili
  - ▶ Metodo write stampa un valore a video: stringhe, numeri, immagini, ...
- ▶ Esempi (document\_write.html)
  - ▶ `document.write("paperone")`
  - ▶ `document.write(18.45 - 34.44)`
  - ▶ `document.write('paperino')`
  - ▶ `document.write('<IMG src="image.gif">')`
  - ▶ `document.write = this.document.write`
- ▶ Altro esempio: file HTML con immagine con nome "image\_1":
  - ▶ `document.image_1.width` si riferisce alla larghezza dell'immagine
  - ▶ `document.image_1.width = 40`



# Oggetti: proprietà e funzioni

- ▶ Oggetto è una collezione di
  - ▶ proprietà (variabili): sono a loro volta oggetti
  - ▶ funzioni (metodi, operazioni)

- ▶ Per accedere alle proprietà di un oggetto

```
window.status = 'hello!';
```

↓                      ↓  
nome oggetto   nome proprietà

- ▶ Per invocare le funzioni di un oggetto

```
window.print(); → funzione
```

```
window.document.write("<p>Ciao!</p>");
```

↓                      ↓                      ↓  
nome oggetto                      funzione

- ▶ Per accedere agli oggetti contenuti in una lista (array)

```
window.document.images[0].src = 'sole.gif';
```

↓  
la proprietà *src* della prima immagine della pagina

# Oggetti: proprietà e funzioni

- ▶ L'invocazione di una funzione apparentemente senza alcun oggetto si riferisce all'oggetto window
  - ▶ `prompt("Come ti chiami?", "boh");`
  - ▶ `window.prompt("Come ti chiami?", "boh");`
- ▶ Un riferimento all'oggetto document non preceduto da un riferimento all'oggetto window, è equivalente al caso in cui document è preceduto da window (implicito)
  - ▶ `document.write("<p>Ciao!</p>");`
  - ▶ `window.document.write("<p>Ciao!</p>");`

# Accesso agli oggetti nella pagina

- ▶ Lista dei moduli (<FORM...) contenuti in una pagina
  - ▶ `window.document.forms[i]`
- ▶ Lista delle immagini (<IMG...) contenute in una pagina
  - ▶ `window.document.images[i]`

# Accesso agli oggetti nella pagina

- ▶ I campi di testo sono oggetti dotati di nome posti all'interno di un oggetto form pure esso dotato di nome
  - ▶ Come tali sono referenziabili con la "dot notation":  
`document.nomeform.nomeTextField`
- ▶ Il campo di testo è caratterizzato dalla proprietà `value`
- ▶ Esempio:
  - ▶ 

```
<FORM name="myform">  
  <INPUT type="text" name="cognome" size=20>  
  <INPUT type="button" name="bottone" value="Show"  
    onClick="alert(document.myform.cognome.value)">  
</FORM>
```

# Accesso agli oggetti nella pagina

- ▶ **Attributo NAME:**
  - ▶ `<FORM NAME="modulo">`
    - ▶ `<INPUT TYPE="text" NAME="codice_fiscale">`
      - ▶ `window.document.modulo.codice_fiscale.value = ...`
  - ▶ `<IMG SRC="pippo.gif" NAME="pippo">`
    - ▶ `window.document.pippo.src = ...`

# Accesso agli oggetti nella pagina

- ▶ Metodo `getElementById(idname)`
  - ▶ Ritorna l'elemento con uno specifico id
  - ▶ Ritorna null se non esiste l'elemento
- ▶ Esempio
  - ▶ `<INPUT TYPE="text" ID="codice_fiscale">`
    - ▶ `window.document.getElementById("codice_fiscale").value=...`
  - ▶ `<IMG SRC="pippo.gif" ID="pippo">`
    - ▶ `img=window.document.getElementById("pippo");`  
`img.src=...`

# Eventi

- ▶ I programmi JavaScript sono tipicamente "guidati dagli eventi" (event-driven)
  - ▶ Eventi sono scatenati da azioni dell'utente sulla pagina Web
    - ▶ Ogni volta che l'utente scrive qualcosa in una casella, preme un pulsante, ridimensiona una finestra ecc... genera un "evento"
  - ▶ Un programma JavaScript deve contenere un gestore di eventi (event handler), che sia in grado di ricevere e interpretare le azioni dell'utente (eventi)
  - ▶ Il DOM fornisce una serie di gestori di eventi predefiniti
    - ▶ L'accadere di un evento nella pagina Web mette automaticamente in azione il corrispondente gestore di eventi

# Eventi

- ▶ `<a href="#" onClick = "window.print()" >`
  - ▶ Attributo `onClick`: evento click innesca il gestore
  - ▶ `window.print()` è un codice JavaScript che viene eseguito dal gestore
  - ▶ `href`
    - ▶ `"#"` resta nella pagina in cui si trova (salta al Top)
    - ▶ URL va alla pagina indicata (carica la nuova pagina)
    - ▶ `<A HREF="#" onClick="JavaScript:window.print(); return false;">` il browser resta nella pagina corrente, senza saltare al Top
  
- ▶ Esempio `alert.html`, `print.html`



# Eventi

- ▶ Un'istruzione JavaScript può essere inserita all'interno di un tag HTML, (anziché essere racchiusa nei tag `<SCRIPT>...</SCRIPT>`)
- ▶ In questi casi, il gestore di evento invocato farà riferimento al tag in cui si trova l'istruzione
  - ▶ `<A HREF="#" onClick="window.print()" >`
    - ▶ Quando l'utente fa click (onClick) sul link (`<A...>`)
  - ▶ `<FORM NAME="modulo" onSubmit="alert('Ciao!');" >`
    - ▶ Quando l'utente invia (onSubmit) il modulo (`<FORM...>`)
  - ▶ `<INPUT TYPE="text" NAME="login" onFocus="...;">`
    - ▶ Quando l'utente porta il cursore (onFocus) nel campo di testo (`<INPUT TYPE="text"...>`)
  - ▶ `<BODY onLoad="alert('caricato');" >`
    - ▶ Quando l'utente carica (onLoad) la pagina (`<BODY...>`)

# Eventi

- ▶ Gli eventi intercettabili su un link: onClick, onMouseOver, onMouseOut
- ▶ Gli eventi intercettabili su una finestra: onLoad, onUnload, onBlur
- ▶ Esempio:
  - ▶ `<BODY onLoad = "alert('caricato')" >`  
    `<FORM name="myform">`  
        `<INPUT type="button" name="bottone"`  
            `value="Premi qui"`  
            `onClick = "document.write(sum(1,13))" >`  
    `</FORM>`  
    `</BODY>`

# Gestione eventi

- ▶ Per sfruttare il valore restituito da `confirm`, `prompt`, o qualsiasi altra funzione JavaScript occorre inserire come valore dell'attributo `onClick` un programma JavaScript (una sequenza o una chiamata di funzione)
- ▶ Esempi:
  - ▶ `onClick = "x = prompt('Cognome e Nome:'); document.write(x)"`
  - ▶ `onClick = "ok = confirm('Va bene così?'); if(!ok) alert('ATTENTO...')"`

# Eventi (form)

- ▶ JavaScript permette di
  - ▶ Intercettare eventi che "accadono" nei campi di un modulo
  - ▶ Modificare i campi di un modulo

# Eventi (form)

- ▶ Un form contiene solitamente campi di testo e bottoni

```
<FORM name="myform">
```

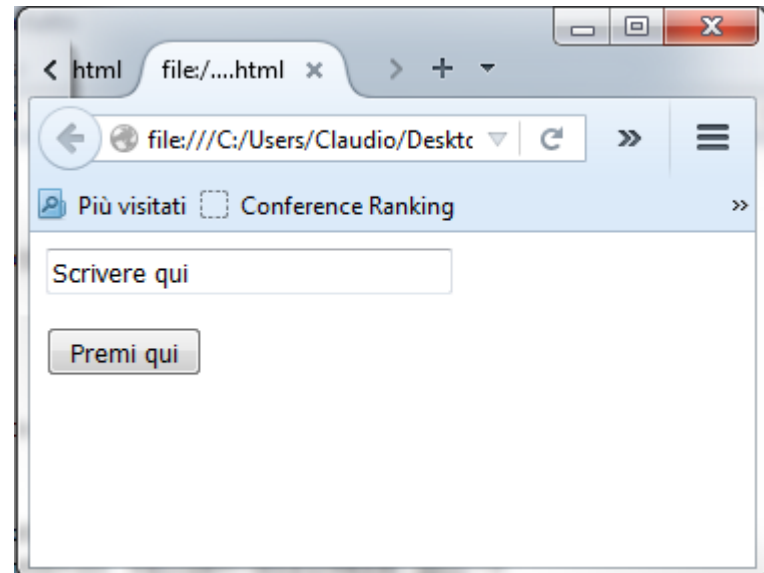
```
  <INPUT type="text" name="campoDiTesto"  
    size=30 maxlength=30 value="Scrivere qui">
```

```
<P>
```

```
  <INPUT type="button" name="bottone"  
    value="Premi qui">
```

```
</FORM>
```

- ▶ Quando il bottone viene premuto è possibile invocare una funzione JavaScript



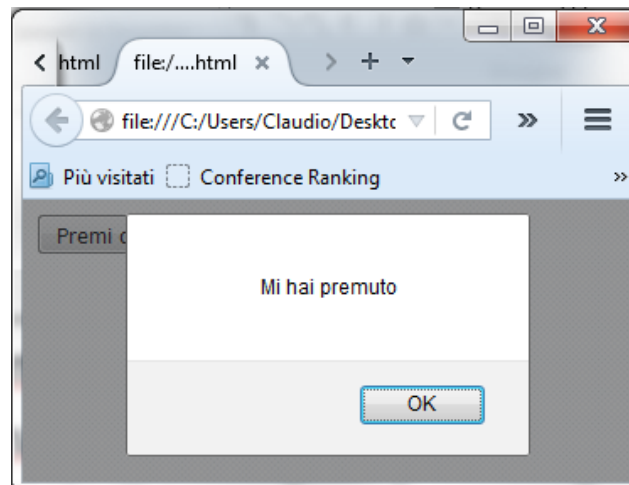
# Eventi (form)

- ▶ Quando si preme il bottone, l'evento bottone premuto può essere intercettato mediante l'attributo onClick

```
<FORM name="myform">
```

```
  <INPUT type="button" name="bottone"  
    value="Premi qui"  
    onClick = "alert('Mi hai premuto')" >
```

```
</FORM>
```



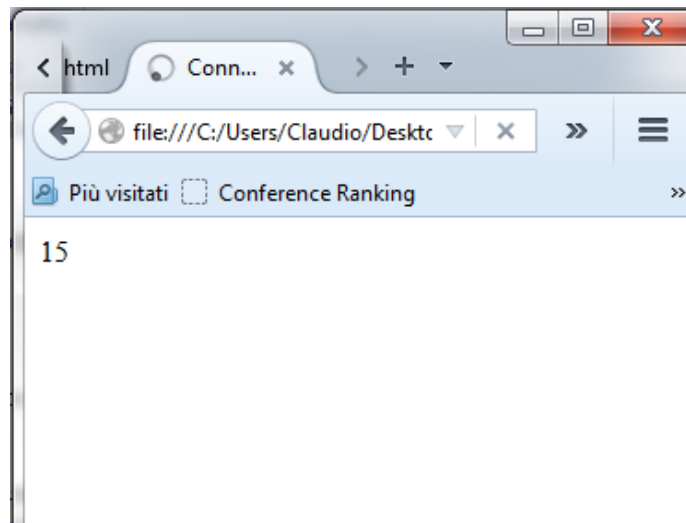
# Eventi (form)

- ▶ ALTERNATIVA: quando si preme il bottone, far scrivere il risultato di una funzione (form\_print\_temporeale.html)

```
<FORM name="myform">
```

```
  <INPUT type="button" name="bottone"  
    value="Premi qui"  
    onClick = "document.write(15)" >
```

```
</FORM>
```



# Eventi (form)

- ▶ Modifica di un campo del messaggio tramite proprietà `onMouseOver` e `onMouseOut` (form\_onmouseover.html)

```
<DIV ALIGN="CENTER">
```

```
<FORM NAME="modulo">
```

```
<INPUT TYPE="text" NAME="over" VALUE="Il mouse è sopra  
la scritta1 o la scritta2?" size="50px">
```

```
</FORM>
```

```
<A HREF="" onMouseOver= "window.document.modulo.over.value='scritta1';"  
onmouseout="window.document.modulo.over.value='Il mouse è sopra la  
scritta1 o la scritta2?'">scritta1</A>
```

...



# Eventi (form)

- ▶ Selezionare/deselezionare le checkbox al click su un link/bottone (form\_onclick\_button.html)

```
<FORM NAME="modulo">  
  Opzione 1: <INPUT TYPE="checkbox" NAME="box[]" VALUE="v1">  
  ...  
  <input type=button onclick="checkAll(3)"  
        value="Seleziona/deseleziona tutte le opzioni">
```

HTML

```
function checkAll(lungh) {  
  for (var i=0; i<lungh; i++) {  
    if (document.modulo.elements[i].checked==true)  
      {document.modulo.elements[i].checked=false;}  
    else if (document.modulo.elements[i].checked==false)  
      {document.modulo.elements[i].checked=true;}  
  }  
}
```

JavaScript

# Eventi (form)

- ▶ Selezionare/deselezionare le checkbox al click su un link/bottone (form\_onclick\_link1.html e form\_onclick\_link2.html)

```
<FORM NAME="modulo">  
  Opzione 1: <INPUT TYPE="checkbox" ID="box1" VALUE="v1">  
  ...  
</FORM>  
<A HREF="#" onClick="checkAll(3);">Seleziona/deseleziona tutte le opzioni</A>
```

HTML

```
function checkAll(lungh) {  
  for (var i=0; i<lungh; i++) {  
    var n = i+1;  
    if (document.getElementById("box"+n).checked==true)  
      {document.getElementById("box"+n).checked=false;}  
    else if (document.getElementById("box"+n).checked==false)  
      {document.getElementById("box"+n).checked=true;}  
  }  
}
```

JavaScript

# Eventi (form)

- ▶ Impostare dinamicamente le voci di un menu (a), intercettare il click del mouse sul pulsante di invio di un form e di conseguenza mostrare un alert (b)
- ▶ Menù (form\_change\_menu.html)

```
<FORM NAME="modulo" onSubmit=" warn(); return false;">  
  <SELECT NAME="menu" id="menuid">  
    <OPTION ID="fr" VALUE="inter">Inter</OPTION>  
    <OPTION ID="pt" VALUE="milan">Milan</OPTION>  
    <OPTION ID="st" VALUE="juve">Juve</OPTION>  
  </SELECT>  
  <INPUT TYPE="Submit" VALUE="OK">  
</FORM>
```

**return false** evita il refresh della pagina:  
Che avviene di default con il pulsante submit

# Eventi (form)

- ▶ Impostare dinamicamente le voci di un menu (a)

```
<SCRIPT language="JavaScript">
var nazione = prompt("Sei Italiano o Inglese?", "Italiano");
if (nazione == "inglese") {
    window.document.modulo.menu.options[0].text="Chelsea";
    window.document.modulo.menu.options[1].text="Manchester United";
    window.document.modulo.menu.options[2].text="Manchester City";
    var select = document.getElementById('menuid');
    var opt = document.createElement('option');
    opt.id = "tt";
    opt.value = "liverpool";
    opt.innerHTML = "Liverpool";
    select.appendChild(opt);
}
</SCRIPT>
```

# Eventi (form)

- ▶ Intercettare il click del mouse sul pulsante di invio di un form e di conseguenza mostrare un alert (b)

```
<SCRIPT language="JavaScript">
function warn() {
    if (window.document.modulo.menu.selectedIndex == 0) {
        alert("Very good!");
    }
}
</SCRIPT>
```

# Eventi (onLoad)

- ▶ Evento scatenato quando un oggetto viene caricato
  - ▶ Ad esempio permette di ridirezionare l'utente ad un altro URL (j1.html):

```
<SCRIPT language="JavaScript">  
  function jump(){  
    window.location.href="http://www.unimi.it";  
  }  
</SCRIPT>  
</HEAD>  
<BODY onLoad = "jump()" >
```

- ▶ onLoad carica il gestore che viene innescato tramite la funzione jump()
  - ▶ Ridireziona all'URL puntata dalla proprietà href dell'oggetto location

# Eventi (onLoad)

- ▶ Ridirezione dopo timeout (ms) – j2.html

```
<HTML>
```

```
<HEAD>
```

```
<SCRIPT language="JavaScript">
```

```
function jump(){
```

```
    window.setTimeout("window.location.href=  
                        'http://www.di.unimi.it/';", 5*1000);} 
```

```
</SCRIPT>
```

```
</HEAD>
```

```
<BODY onLoad = "jump()">
```

```
</BODY>
```

```
</HTML>
```

# Eventi (onMouseOver)

- ▶ Evento scatenato al passaggio del mouse su un oggetto
  - ▶ Ad esempio, utilizzato per cambiare stile di un oggetto
  - ▶ Altro utilizzo molto comune è l' image swap (o roll-over), cambiamento dell'aspetto di un'immagine
  - ▶ 

```
<p ALIGN="CENTER" style="background-color:green" onMouseOver="style='background-color:yellow';« onMouseOut="style='background-color:green';">  
    Cambia il colore dello sfondo.  
</p>
```
- ▶ onMouseOver/Out cambia il valore src dell'immagine
  - ▶ rollover\_style.html, rollover\_image.html



# Gestione finestre (open)

- ▶ Oggetto window permette di agire e gestire le finestre del browser
  - ▶ `window.open(URL, nome, [proprietà]);` apre una nuova finestra (o una nuova scheda)
    - ▶ URL = indirizzo della pagina da caricare
    - ▶ Nome = identificatore della finestra
    - ▶ Proprietà (opzionale): lista delle proprietà della nuova finestra (se omesso, la nuova finestra mantiene le proprietà della corrente)
    - ▶ Ritorna un riferimento alla finestra aperta (null se c'è errore)
- ▶ Esempio: apertura finestra nella stessa finestra/scheda
  - ▶ `<a href="#" onClick = "window.open('http://www.di.unimi.it', 'pippo' ); return false;">nuova finestra!</a>`

# Gestione finestra (open)

- ▶ Un altro esempio
  - ▶ `<a href="#" onClick = "window.open('http://www.di.unimi.it', 'pippo', 'scrollbars=yes,resizable=yes,width=730,height=600, status=no,location=no,toolbar=no,menubar=no'); return false;">finestra con proprietà</a>`
    - ▶ Presenza barre di scorrimento
    - ▶ Possibilità di ridimensionare la finestra
    - ▶ Larghezza e altezza (in pixel)
    - ▶ Presenza della barra di stato (status)
    - ▶ Presenza della barra degli indirizzi (location)
    - ▶ Presenza della barra dei pulsanti (toolbar)
    - ▶ Presenza della barra del menù
  - ▶ Se stabilisco delle proprietà, verrà sicuramente aperta una nuova finestra (con quelle proprietà) e NON una scheda

# Gestione finestre (open)

## ▶ Un esempio completo

### ▶ Finestra home (index.html)

- ▶ `<a href="#" onClick="window.open('controllo.html','pannello', 'width=200,height=120, status=no,location=no,toolbar=no,menubar=no'); window.focus();">START</a>`

- ▶ Finestra controllo.html messa in primo piano

### ▶ Finestra visualizza (visualizza.html)

- ▶ ``

# Gestione finestra (open)

- ▶ Finestra controllo.html messa in primo piano

- ▶ `var diapositive = window.open("slideShow.html","foto", "width=500, height=500"); window.focus();`

...

```
<a href="#" onClick="diapositive.document.diapo.src  
='paesaggio.jpg'; return false;">Paesaggio</a>
```

```
<a href="#" onClick="diapositive.document.diapo.src='calaChia.jpg';  
return false;">Chia: spiaggia 1</a>
```

```
<a href="#" onClick="diapositive.document.diapo.src='colonna.jpg';  
return false;">Nora: rovine romane</a>
```

```
<a href="#" onClick="diapositive.document.diapo.src='acquaChia.jpg';  
return false;">Chia: spiaggia 2</a>
```

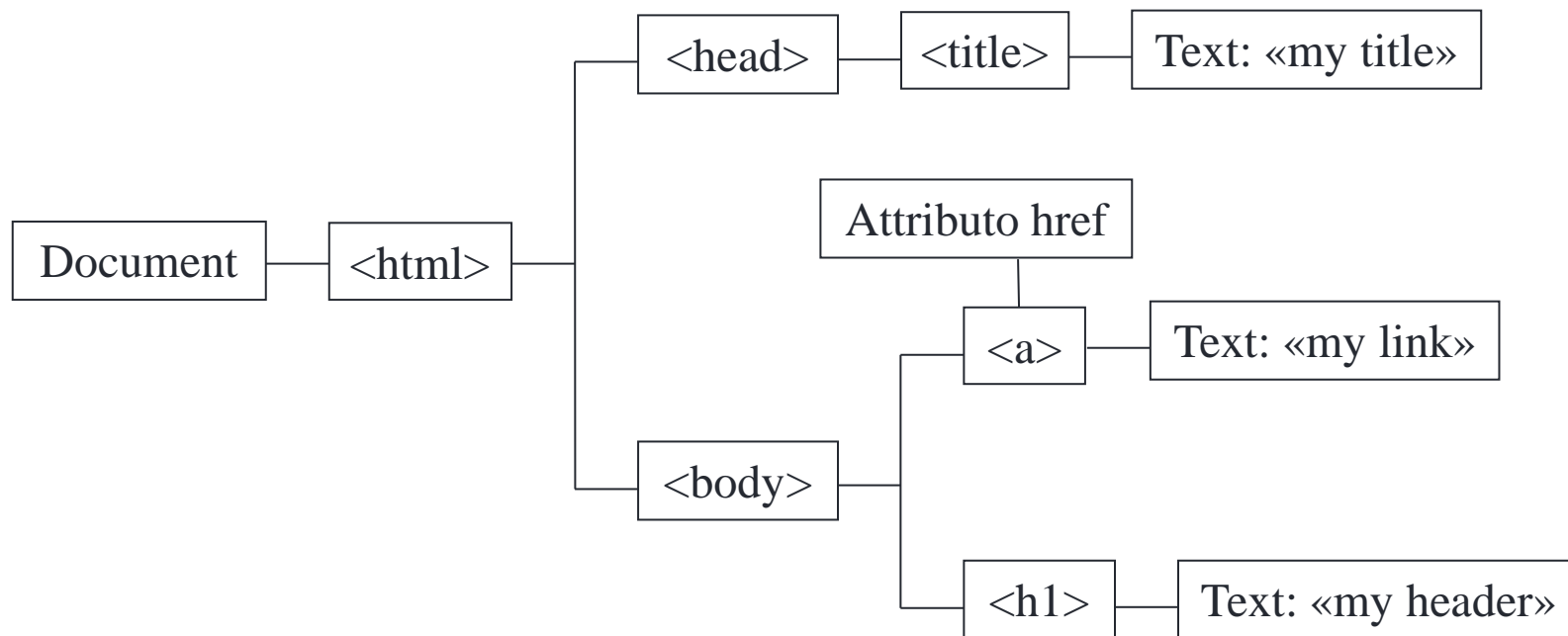
- ▶ `diapositive.document.diapo.src`: attributo `src` dell'immagine `diapo` del contenuto (`document`) della finestra `diapositive`

# Funzioni come link

- ▶ Una funzione JavaScript costituisce un valido link utilizzabile nel tag HTML `<a href= ...> </a>`
- ▶ L'effetto del click su tale link è l'esecuzione delle funzione e l'apparizione del risultato in una nuova pagina HTML all'interno però della stessa finestra
- ▶ Esempio:
  - ▶ `<a href="JavaScript:Math.sum(43,58)" >` Questo dovrebbe essere 101 `</a>`

# Accesso agli oggetti di una pagina tramite nodi

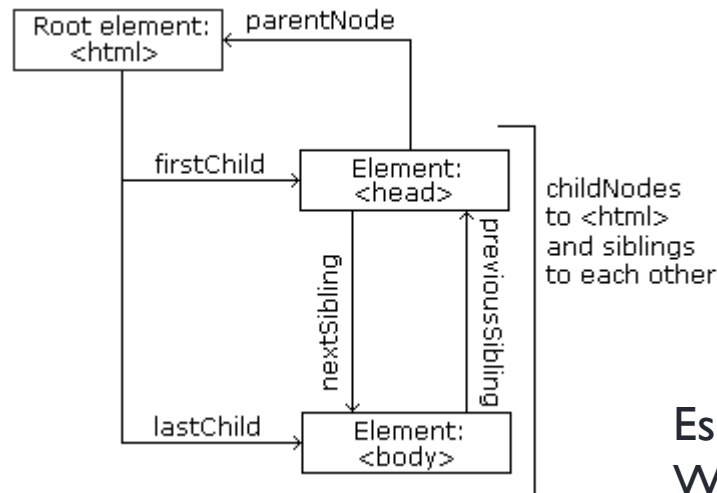
- ▶ Ogni oggetto è un nodo (HTML DOM)
  - ▶ Ogni elemento è un nodo
  - ▶ Ogni testo è un nodo testo
  - ▶ Ogni attributo un nodo attributo
  - ▶ I commenti sono dei nodi commenti



# Relazione tra nodi

- ▶ Definita attraverso diverse funzioni
  - ▶ Parent, child, sibling
  - ▶ Root è la radice del documento (<html>)
  - ▶ Ogni nodo ha un solo nodo padre a parte la root
  - ▶ Ogni nodo può avere 0 o più figli
  - ▶ Sibling sono nodi con lo stesso padre

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```



Esempio tratto da  
W3CSchool

# Relazione tra nodi

- ▶ Proprietà per la navigazione tra nodi
  - ▶ parentNode
  - ▶ childNodes[*nodenumber*]
  - ▶ firstChild
  - ▶ lastChild
  - ▶ nextSibling
  - ▶ previousSibling



# Relazione tra nodi

- ▶ Accesso al primo nodo figlio
  - ▶ `document.getElementById("intro").childNodes[0]`
  - ▶ `document.getElementById("intro").firstChild`
- ▶ Il valore testuale di un elemento è a sua volta un nodo
  - ▶ `document.getElementById("intro").childNodes[0].nodeValue`
  - ▶ `document.getElementById("intro").firstChild.nodeValue`
- ▶ Altre proprietà oltre a `nodeValue`
  - ▶ `nodeName`, `nodeType` entrambi in sola lettura

# Relazione tra nodi

```
<h1 id="intro">Testo</h1>
```

```
<p id="firstchild"></p>
```

```
<p id="childnodes"></p>
```

```
<script>
```

```
myText = document.getElementById("intro").firstChild.nodeValue;  
document.getElementById("firstchild").innerHTML = "First Child " + myText;
```

```
myText = document.getElementById("intro").childNodes[0].nodeValue;  
document.getElementById("childnodes").innerHTML = "ChildNodes[0] " +  
myText;
```

```
</script>
```

▶ first-child.html

# Aggiunta elementi

- ▶ Aggiunta di elementi all'albero DOM

```
<div id="div1">  
  <p id="p1">This is a paragraph.</p>  
  <p id="p2">This is another paragraph.</p>  
</div>
```

```
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);
```

```
var element = document.getElementById("div1");  
element.appendChild(para);  
</script>
```

- ▶ Metodo `appendChild()` può essere sostituito da `insertBefore()`

# Rimozione elementi

- ▶ Rimozione di elementi dall'albero DOM

```
<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>
```

```
<script>  
var parent = document.getElementById("div1");  
var child = document.getElementById("p1");  
parent.removeChild(child);  
</script>
```

# Sostituzione elementi

- ▶ Sostituzione di elementi nell'albero DOM

```
<div id="div1">  
<p id="p1">This is a paragraph.</p>  
<p id="p2">This is another paragraph.</p>  
</div>
```

```
<script>  
var para = document.createElement("p");  
var node = document.createTextNode("This is new.");  
para.appendChild(node);
```

```
var parent = document.getElementById("div1");  
var child = document.getElementById("p1");  
parent.replaceChild(para,child);  
</script>
```

# Collezione di nodi

- ▶ Node List è una collezione di nodi
  - ▶ Simile agli array di oggetti `images[]` e `forms[]` visti prima
  - ▶ Ad esempio, il metodo `getElementsByTagName()` ritorna una lista di nodi
    - ▶ La lista di nodi è una collection di nodi simile ad array
  - ▶ La proprietà `length` ritorna la lunghezza della lista di nodi
    - ▶ `myNodelistLength = document.getElementsByTagName("p").length`
- ▶ Esempio (node-list.html)
  - ▶ `var x = document.getElementsByTagName("p");`
    - ▶ Ritorna la lista di tutti gli elementi p
  - ▶ `y = x[1];`
    - ▶ Permette di accedere al secondo elemento p

# Metodo `getElementsByClassName`

- ▶ Metodo `getElementsByClassName(classname)`
  - ▶ Ritorna una lista `NodeList` di tutti gli elementi con una specifica classe
  - ▶ Elementi in `NodeList` acceduti tramite indici
- ▶ Esempio
  - ▶ `<div class="example">First div element with class="example".</div>`  
`<div class="example">Second div element with class="example".</div>`  
`var x=window.document.getElementsByClassName("example")`  
`document.write(x[0].innerHTML)`

# Funzioni, modello a oggetti



# Introduzione

- ▶ Object-based language (ma non object-oriented)
  - ▶ Usa l'idea di incapsulare stato e operazioni all'interno di oggetti
  - ▶ Non applica i concetti di ereditarietà e sottotipi
- ▶ JavaScript a volte riferito come prototype-based language
  - ▶ Non esistono classi, ma gli oggetti ereditano codice e dati da altri oggetti template
  - ▶ Vengono clonati oggetti che servono come «prototipi»

# Definizione di funzioni

- ▶ Definite tramite keyword *function* e sempre racchiuso in un blocco
- ▶ Possono essere considerate sia procedure...
  - ▶ Non ha istruzione return
  - ▶ 

```
function printSum(a,b) {  
    document.write(a+b)  
}
```
- ▶ ... sia funzioni in senso proprio (non esiste la keyword `void`)
  - ▶ 

```
function sum(a,b) { return a+b }
```
- ▶ I parametri formali sono senza dichiarazione di tipo

# Chiamata a funzione

- ▶ Chiamate come in un linguaggio di programmazione tradizionale, fornendo la lista dei parametri attuali
  - ▶ `document.write(sum(10,5) + "<br/>")`
  - ▶ `printSum(19, 34.33) - printsum.html`
- ▶ Se i tipi non hanno senso per le operazioni fornite, l'interprete JavaScript ritorna un errore a runtime
  - ▶ Non viene mostrato il risultato

# Parametri di funzione

- ▶ Passaggio di parametro per valore
  - ▶ Nel caso di oggetti, si copiano riferimenti
- ▶ A differenza di C e Java, è lecito definire una funzione dentro un'altra funzione (simile al Pascal)
- ▶ Se i parametri attuali sono più di quelli necessari
  - ▶ Nessun errore
  - ▶ Quelli extra vengono ignorati
- ▶ Se i parametri attuali sono meno di quelli necessari
  - ▶ Quelli mancanti sono inizializzati a undefined (una costante di sistema)

# Variabili: tipi di dichiarazione

- ▶ Dichiarazione delle variabili è
  - ▶ Implicita o esplicita per variabili globali
  - ▶ Necessariamente esplicita, per variabili locali
- ▶ Dichiarazione esplicita (keyword var)
  - ▶ `var pippo = 10` // dichiarazione esplicita
  - ▶ `pippo = 10` // dichiarazione implicita
- ▶ La dichiarazione implicita è sempre e solo per variabili globali
- ▶ La dichiarazione esplicita ha un effetto che dipende da dove si trova la dichiarazione

# Variabili: dichiarazione esplicita

- ▶ Fuori da funzioni, la parola chiave `var` è influente
  - ▶ La variabile definita è globale
- ▶ All'interno di funzioni, la parola chiave `var` ha un significato preciso
  - ▶ Indica che la nuova variabile è locale, ossia ha come scope la funzione
- ▶ All'interno di funzioni, una dichiarazione senza la parola chiave `var` introduce una variabile globale

```
x=6 // globale
function test(){
  x = 18 // globale
}
test()
// qui x vale 18
```

```
var x=6 // globale
function test(){
  var x = 18 // locale
}
test()
// qui x vale 6
```

# Variabili: environment di riferimento

- ▶ Quando ci si riferisce a una variabile
  - ▶ Prima si cerca localmente
  - ▶ Se non è definita si accede a quella globale
- ▶ Esempio in ambiente globale
  - ▶ `var f = 4` // f è comunque globale
  - ▶ `g = f * 3` // g è comunque globale, e vale 12
- ▶ Esempio in ambiente locale (dentro a funzioni)
  - ▶ `var f = 5` // f è locale
  - ▶ `g = f * 3` // g è globale, e vale 15

# Variabili: environment di riferimento

- ▶ `pippo=10;`  
`var fz=function()`  
`{`  
`alert(pippo);`  
`var pippo=4;`  
`}`  
`fz();`
- ▶ Cosa viene stampato?
  - ▶ `Scope.html`



# Variabili: environment di riferimento

- ▶ `pippo=10;`  
`var fz=function()`  
`{`  
`alert(pippo);`  
`var pippo=4;`  
`}`  
`fz();`
- ▶ Cosa viene stampato? `pippo=undefined`
  - ▶ [Scope.html](#)

# Variabili: environment di riferimento

- ▶ Il parser prima analizza tutto il codice e crea tutte le variabili e strutture lasciandole undefined
- ▶ Poi esegue una riga alla volta
  - ▶ Quando raggiungo la chiamata alert(pippo) cerca una variabile locale e la trova undefined
  - ▶ Stampa quindi undefined

# Variabili: environment di riferimento

- ▶ `pippo=10;`  
`var fz=function()`  
`{`  
`alert(pippo);`  
`}`  
`fz();`
- ▶ Cosa viene stampato?

# Variabili: environment di riferimento

▶ pippo=10;

```
var fz=function()
```

```
{
```

```
    alert(pippo);
```

```
}
```

```
fz();
```

▶ Cosa viene stampato? pippo=10

▶ Non trovando la variabile locale, stampa la globale

# Funzioni e chiusure

- ▶ La natura interpretata di JavaScript e l'esistenza di un ambiente globale pongono una domanda
  - ▶ Quando una funzione usa un simbolo non definito al suo interno, quale definizione vale per esso?
    - ▶ La definizione che esso ha nell'ambiente in cui la funzione è definita, oppure
    - ▶ La definizione che esso ha nell'ambiente in cui la funzione è chiamata?

# Funzioni e chiusure

- ▶ Si consideri il seguente programma JavaScript
  - ▶ 

```
var x = 20;
function provaEnv(z) { return z + x; }
alert(provaEnv(18)) // visualizza certamente 38
function testEnv() {
    var x = -1;
    alert(provaEnv(18)); // COSA visualizza ???
}
```
- ▶ Nella funzione testEnv si ridefinisce il simbolo x, poi si invoca la funzione provaEnv, che usa il simbolo x ... ma QUALE x?
- ▶ Nell'ambiente in cui provaEnv è definita, il simbolo x aveva un altro significato rispetto a quello che ha ora!

# Funzioni e chiusure

- ▶ `var x = 20;`  
`function provaEnv(z) { return z + x; }`  
`function testEnv() {`  
    `var x = -1;`  
    `return provaEnv(18); // COSA visualizza ???`  
`}`
- ▶ Se vale l'ambiente esistente all'atto dell'uso di `provaEnv`, si parla di chiusura dinamica; se prevale l'ambiente di definizione di `provaEnv`, si parla di chiusura lessicale
  - ▶ JavaScript adotta la chiusura lessicale → `testEnv` visualizza ancora 38 (non 17)

# Funzioni come dati

- ▶ Variabili possono riferirsi a funzioni
  - ▶ La funzione non ha nome (anche se potrebbe)
    - ▶ `var f = function (z) { return z*z; }`
  - ▶ La funzione viene invocata tramite il nome della variabile
    - ▶ `var result = f(4);`
    - ▶ `g = f` produce aliasing
- ▶ Possibile passare funzioni come parametro ad altre funzioni
  - ▶ `function calc(f, x) {return f(x); }`
  - ▶ Se `f` cambia, `calc` calcola una funzione diversa



# Funzioni come dati - Esempi

- ▶ `function calc(f, x) { return f(x) }`
  - ▶ `calc(Math.sin, .8)` ritorna `0.7173560908995228`
  - ▶ `calc(Math.log, .8)` ritorna `-0.2231435513142097`
- ▶ Altri esempi
  - ▶ `calc(x*x, .8)` ritorna un errore
    - ▶ `x*x` non è un oggetto funzione del programma
  - ▶ `calc(funz, .8)` va bene solo se la variabile `funz` fa riferimento a un costrutto `function`
  - ▶ `calc("Math.sin", .8)` ritorna errore
    - ▶ `"Math.sin"` è una stringa non una funzione
    - ▶ Il nome di una funzione non è la funzione

# Funzioni come dati - Conseguenze

- ▶ Per utilizzare una funzione come dato occorre avere effettivamente un oggetto funzione
- ▶ Non si può sfruttare questa caratteristica per far eseguire una funzione di cui sia noto solo il nome (letto da tastiera)
  - ▶ `calc("Math.sin", .8)` ritorna errore
- ▶ o di cui sia noto solo il codice
  - ▶ `calc(x*x, .8)` ritorna errore
- ▶ Il problema è risolvibile
  - ▶ Si costruisce esplicitamente un «oggetto funzione»
  - ▶ Oppure si accede alla funzione tramite le proprietà dell'oggetto globale

# Funzioni come dati - Conseguenze

- ▶ Per utilizzare una funzione come dato occorre avere effettivamente un oggetto funzione
- ▶ Non si può sfruttare questa caratteristica per far eseguire una funzione di cui sia noto solo il nome (letto da tastiera)
  - ▶ `calc("Math.sin", .8)` ritorna errore
- ▶ o di cui sia noto solo il codice
  - ▶ `calc(x*x, .8)` ritorna errore
- ▶ Il problema è risolvibile
  - ▶ **Si costruisce esplicitamente un «oggetto funzione»**
  - ▶ Oppure si accede alla funzione tramite le proprietà dell'oggetto globale

# Oggetti

- ▶ Un oggetto JavaScript è una collezione di dati dotata di nome
  - ▶ Ogni dato interpretabile come una proprietà
  - ▶ Per accedere alle proprietà si usa la "dot notation"
    - ▶ nomeOggetto.nomeProprietà
  - ▶ Tutte le proprietà sono accessibili
- ▶ Un oggetto JavaScript è costruito tramite costruttore
  - ▶ Stabilisce la struttura dell'oggetto e quindi le sue proprietà
  - ▶ I costruttori sono invocati mediante l'operatore new
  - ▶ In JavaScript non esistono classi, sottoclassi
    - ▶ Il nome del costruttore è a scelta dell'utente
    - ▶ La struttura di un oggetto non è stabilita dalla classe

# Oggetti: Definizione

- ▶ La struttura di oggetto JavaScript viene definita dal costruttore usato per crearlo
- ▶ È all'interno del costruttore che si specificano le proprietà (iniziali) dell'oggetto, elencandole con la dot notation e la keyword this
- ▶ Identificatore globale (function expression)
  - ▶ 

```
Point = function(i,j){  
    this.x = i;  
    this.y = j;  
}
```
- ▶ Identificatore locale (function declaration)
  - ▶ 

```
function Point(i,j){  
    this.x = i;  
    this.y = j;  
}
```
- ▶ La keyword this è necessaria, altrimenti ci si riferirebbe all'environment locale della funzione costruttore

# Oggetti: Costruzione

- ▶ Per costruire oggetti si applica l'operatore `new` a una funzione costruttore
  - ▶ `p1 = new Point(3,4);`
  - ▶ `p2 = new Point(0,1);`
  - ▶ L'argomento di `new` non è il nome di una classe, è solo il nome di una funzione costruttore.
- ▶ A partire da JavaScript 1.2, si possono creare oggetti anche elencando direttamente le proprietà con i rispettivi valori
  - ▶ Sequenza di coppie `nome:valore` separate da virgole e racchiusa fra parentesi graffe.
  - ▶ `p3 = { x:10, y:7 }`

# Oggetti: Accesso alle proprietà

- ▶ Proprietà di un oggetto sono pubbliche e accessibili
  - ▶ Esistono anche proprietà "di sistema" e come tali non visibili, né enumerabili con gli appositi costrutti
  
- ▶ Accesso attraverso dot notation
  - ▶ `p1.x = 10; // da (3,4) diventa (10,4)`

# Aggiunta e rimozione di proprietà

- ▶ Le proprietà specificate nel costruttore sono le proprietà iniziali
- ▶ È possibile aggiungere dinamicamente nuove proprietà semplicemente nominandole e usandole
  - ▶ `p1.z = -3; //` da `{x:10, y:4}` diventa `{x:10, y:4, z: -3}`
  - ▶ NB: non esiste il concetto di classe come "specifica della struttura (fissa) di una collezione di oggetti", come in Java o C++
- ▶ È possibile rimuovere dinamicamente proprietà, mediante l'operatore `delete`
  - ▶ `delete p1.x; //` da `{x:10, y:4, z: -3}` diventa `{y:4, z: -3}`



# Metodi per (singoli) oggetti

- ▶ Definire metodi è semplicemente un caso particolare dell'aggiunta di proprietà
- ▶ Non esistendo il concetto di classe, un metodo viene definito per uno specifico oggetto (ad esempio, p1) non per tutti gli oggetti della stessa "classe"
- ▶ Metodo getX per p1:
  - ▶ `p1.getX = function() { return this.x; }`
  - ▶ Al solito, `this` è necessario per evitare di riferirsi all'environment locale della funzione costruttore

# Metodi per una "classe" di oggetti

- ▶ In assenza del concetto di classe, assicurare che oggetti "dello stesso tipo" abbiano lo stesso funzionamento richiede un'opportuna metodologia
- ▶ Un possibile approccio consiste nel definire tali metodi dentro al costruttore
  - ▶ 

```
Point = function(i,j) {  
  this.x = i; this.y = j;  
  this.getX = function(){ return this.x }  
  this.getY = function(){ return this.y }  
}
```

# Metodi: Invocazione

- ▶ L'operatore di chiamata () è quello che effettivamente invoca il metodo
  - ▶ `document.write( p1.getX() + "<br/>" )` permette di invocare il metodo `p1.getX = function() { return this.x; }`
- ▶ **ATTENZIONE:** se si invoca un metodo inesistente si ha errore a run-time (metodo non supportato)
  - ▶ **NB:** se l'interprete JavaScript incontra un errore a run-time, non esegue le istruzioni successive e spesso non visualizza alcun messaggio d'errore!

# Oggetti Function

- ▶ Permettono di definire funzioni
  - ▶ Ogni funzione JavaScript è un oggetto
  - ▶ Definizione implicita tramite il costrutto function
  - ▶ Definizione esplicita tramite il costruttore Function

# Oggetti Function: definizione implicita

- ▶ Definizione implicita tramite il costrutto function
  - ▶ Argomenti sono i parametri formali della funzione
  - ▶ Il corpo della funzione è racchiuso tra parentesi graffe
    - ▶ `funzione = function(x) { return f(x) }`
  - ▶ Costruito dentro il programma JavaScript
    - ▶ Valutato una sola volta
    - ▶ Efficiente ma non flessibile

# Oggetti Function: definizione esplicita

- ▶ Definizione esplicita tramite il costruttore Function
  - ▶ Argomenti sono tutte stringhe e rappresentano i parametri della funzione definita
  - ▶ Solo l'ultimo argomento ha un comportamento diverso e rappresenta il corpo della funzione
    - ▶ `funzione = new Function("x", "return f(x)")`
  - ▶ Costruito a partire da stringhe
    - ▶ Valutato ogni volta
    - ▶ Poco efficiente, ma molto flessibile

# Funzioni come dati: Oggetto function

- ▶ Riprendiamo la funzione `function calc(f, x) { return f(x) }`
  - ▶ `f` deve essere un oggetto funzione
  - ▶ `calc(Math.sin, .8)` OK
  - ▶ `calc(x*x, .8)` NO
  
- ▶ Costruttore `Function` ci viene in aiuto
  - ▶ Data il codice definiamo un oggetto funzione che passiamo come parametro alla funzione `calc`
  - ▶ `calc(new Function("x", "return x*x"), .8)` OK

# Funzioni come dati: Esempio

- ▶ Funzione on demand
  - ▶ Inserire la funzione da calcolare
    - ▶ `var funzione = prompt("Scrivere f(x): ")`
  - ▶ Inserire il/i parametri da usare
    - ▶ `var x = prompt("Calcolare per x = ? ")`
  - ▶ Calcolare la funzione (invocazione riflessiva)
    - ▶ `var f = new Function("x", "return " + funzione)`
  - ▶ Mostrare il risultato
    - ▶ `confirm("Risultato: " + f(x))`



# Funzioni come dati: Problema

- ▶ Valori immessi da linea di comando (prompt) sono stringhe
  - ▶ Ad esempio, una funzione che incrementa un numero ( $x+1$ ) viene considerata come un'operazione di concatenazione
  - ▶  $x+1$  con  $x=10$  ritorna 101
- ▶ Contromisure
  - ▶ Utente specifica il tipo del dato attraverso una conversione esplicita, ad esempio `parseInt(x)`
  - ▶ Programma implementa la conversione esplicita dopo il prompt:
    - ▶ `var x = parseInt(prompt("Calcolare per x = ? "))`
    - ▶ `typeof(x) = number`

# Conclusioni

- ▶ Linguaggio di scripting JavaScript
  - ▶ Tipi, variabili, costrutti...
  - ▶ Oggetti DOM, eventi, finestre, nodi
  - ▶ Funzioni, modello a oggetti