



# **Tecnologie Web T**

## **JavaScript**

Home Page del corso: <http://www-db.disi.unibo.it/courses/TW/>  
Versione elettronica: 3.04.JavaScript.pdf  
Versione elettronica: 3.04.JavaScript-2p.pdf

# Che cos'è JavaScript

---

- JavaScript è un **linguaggio di *scripting*** sviluppato per **dare interattività alle pagine HTML**
- Può essere inserito direttamente nelle pagine Web
  - in pratica è lo standard «**client-side**» per implementare **pagine «dinamiche»**
- Il suo nome ufficiale è **ECMAScript**
  - È diventato standard ECMA (European Computer Manufactures Association) (ECMA-262) nel 1997
  - È anche uno standard ISO (ISO/IEC 16262)
- Sviluppato inizialmente da Netscape (il nome originale era **LiveScript**) e introdotto in Netscape 2 nel 1995
- In seguito anche Microsoft ha lavorato sul linguaggio producendo una sua variante chiamata **JScript**

# Processo di standardizzazione di JavaScript

---

- È diventato standard ECMA nel 1997 (ECMA-262)
- Nel dicembre 1999 si è giunti alla versione ECMA-262 Edition 3, anche noto come **ECMAScript Edition 3**, corrisponde a **JavaScript 1.5**
- Nel dicembre 2009 si è definita la versione **ECMAScript Edition 5** (superset di ECMAScript Edition 3), corrispondente a **JavaScript 1.8**
- Nel giugno 2011 si è giunti **ECMAScript Edition 5.1** (superset di ECMAScript Edition 5), corrispondente a **JavaScript 1.8.5**
- 2015, **ECMAScript Edition 6**, 2017, **ECMAScript Edition 7**, 2018, **ECMAScript Edition 8**
- La **versione corrente** dello standard **ECMAScript** è l'**Edition 9**, rilasciata nel 2018

# JavaScript e Java

---

- Al di là del nome, **Java** e **JavaScript** sono due cose completamente diverse
- L'unica similitudine è legata al fatto di aver entrambi adottato la sintassi del C
- Esistono profonde differenze
  - JavaScript è **interpretato** e non compilato
  - JavaScript è **object-based** ma **non class-based**
    - Esiste il concetto di oggetto
    - Non esiste il concetto di classe
  - JavaScript è **debolmente tipizzato** (*weakly typed*):
    - Non è necessario definire il tipo di una variabile
    - Attenzione però: questo non vuol dire che i dati non abbiano un tipo (sono le variabili a non averlo in modo statico)

# Cosa si può fare con JavaScript

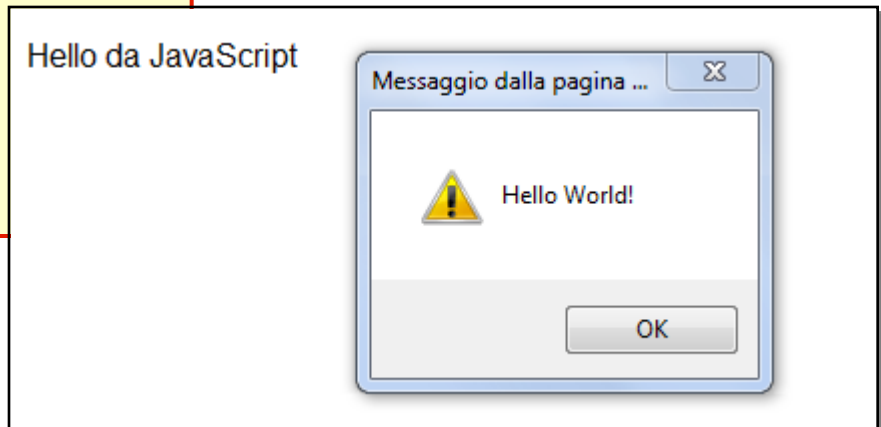
---

- Il codice JavaScript viene eseguito da un **interprete** contenuto all'interno del **browser**
- Nasce per dare **dinamicità alle pagine Web**
- Consente quindi di:
  - **Accedere e modificare elementi** della pagina HTML
  - **Reagire ad eventi generati dall'interazione con l'utente**
  - **Validare i dati inseriti** dall'utente
  - **Interagire con il browser:**
    - e.g., determinare il browser utilizzato, la dimensione della finestra in cui viene mostrata la pagina, lavorare con i browser cookie, ecc.

## Esempio

- Vediamo la versione JavaScript dell'ormai mitico *HelloWorld!*
- Viene mostrato un popup con la scritta «HelloWorld»
- Lo script viene inserito nella pagina HTML usando il tag `<script>`:

```
<html>
  <body>
    <p>Hello da JavaScript</p>
    <script type="text/javascript">
      alert("Hello World!");
    </script>
  </body>
</html>
```



# Sintassi del linguaggio

---

- La sintassi di JavaScript è modellata su quella del C con alcune varianti significative
- In particolare:
  - È un linguaggio **case-sensitive**
  - Le istruzioni sono terminate da ';' ma il terminatore può essere omesso se si va a capo
  - Sono ammessi sia commenti multilinea (delimitati da **/\*** e **\*/**) che mono-linea (iniziano con **//**)
  - Gli identificatori possono contenere lettere, cifre e i caratteri '\_' e '\$' **ma** non possono iniziare con una cifra

# Variabili

---

- Le variabili vengono dichiarate usando la parola chiave **var**:  
*var nomevariabile;*
- **Non hanno un tipo**
  - possono contenere valori di qualunque tipo
- È prevista la possibilità di inizializzare una variabile contestualmente alla dichiarazione  
`var f = 15.8`
- Possono essere dichiarate in linea:  
`for (var i = 1, i<10, i++)`
- Esiste lo **scope globale** e quello **locale** (ovvero dentro una funzione) ma, a differenza di Java, non esiste lo scope di blocco



## Valori speciali

---

- Ad ogni variabile può essere assegnato il valore **null** che rappresenta l'assenza di un valore
- Come in SQL, **null** è un concetto diverso da zero (0) o stringa vuota ("")
- Una variabile non inizializzata ha invece un valore indefinito **undefined**
- I due concetti si assomigliano ma non sono uguali

# Tipi primitivi: numeri e booleani

---

- Javascript prevede pochi tipi primitivi: **numeri**, **booleani** e **stringhe** (forse! 😊)
- Numeri (**number**):
  - Sono rappresentati in formato *floating point* a 8 byte
  - Non c'è distinzione fra interi e reali
  - Esiste il valore speciale *NaN* (*not a number*) per le operazioni non ammesse (ad esempio, radice di un numero negativo)
  - Esiste il valore *infinite* (ad esempio, per la divisione per zero)
- Booleani (**boolean**):
  - ammettono i valori **true** e **false**

# Il concetto di tipo in JavaScript

---

- Come abbiamo detto, alle variabili non viene attribuito un tipo: lo assumono **dinamicamente** in base al dato a cui vengono agganciate
- I dati hanno un tipo; per ogni tipo esiste una sintassi per esprimere le costanti (**literal**)
  - Per i numeri, ad esempio, le costanti hanno la forma usuale: **1.0**, **3.5** o in altre basi
  - Per i booleani sono gli usuali valori **true** e **false**

```
var v; // senza tipo  
  
v = 15.7; // diventa di tipo number  
  
v = true; // diventa di tipo boolean
```

# Oggetti

---

- Gli oggetti sono tipi composti che contengono un certo numero di **proprietà** (attributi)
  - Ogni proprietà ha un **nome** e un **valore**
  - Si accede alle proprietà con l'operatore **'.'** (**punto**)
- Le proprietà non sono definite a priori
  - possono essere aggiunte dinamicamente
- Gli oggetti vengono creati usando l'operatore **new**:  
`var o = new Object()`

## **! Attenzione:**

`Object()` è un costruttore e non una classe. Le classi non esistono e quindi i due concetti non si sovrappongono come avviene in Java!

## Costruire un oggetto

---

- Un oggetto appena creato è completamente vuoto
  - non ha ne proprietà né metodi
- Possiamo costruirlo dinamicamente
  - appena assegniamo un valore ad una proprietà la proprietà comincia ad esistere
- Nell'esempio sottostante creiamo un oggetto e gli aggiungiamo 3 proprietà numeriche: x, y e tot:

```
var o = new Object();  
o.x = 7;  
o.y = 8;  
o.tot = o.x + o.y;  
alert(o.tot);
```

## Costanti oggetto

- Le costanti oggetto (**object literal**) sono racchiuse fra parentesi graffe **{ }** e contengono un elenco di attributi nella forma: **nome:valore**  
**var nomeoggetto =**  
**{prop1:val1, prop2:val2, ...}**
- Usando le costanti oggetto creiamo un oggetto e le proprietà (valorizzate) nello stesso momento
- I due esempi seguenti sono del tutto equivalenti:

```
var o = new Object();  
o.x = 7;  
o.y = 8;  
o.tot = 15;  
alert(o.tot);
```

```
var o = {x:7, y:8, tot:15};  
alert(o.tot);
```

# Array

---

- Gli array sono tipi composti i cui elementi sono accessibili mediante un indice numerico
  - l'indice parte da **zero**
  - non hanno una dimensione prefissata (simili agli *ArrayList* di Java)
  - espongono attributi e metodi
- Vengono istanziati con **new Array**(*[dimensione]*)
  - opzionale  
↙
- Si possono creare e inizializzare usando delle **costanti array** (**array literal**) delimitate da **[ ]**:  
**var varname = [val, val2, ..., valn]**
  - Es. **var a = [1, 2, 3];**
- Possono contenere elementi di tipo eterogeneo:
  - Es. **var b = [1, true, "ciao", {x:1, y:2}];**

# Oggetti e array

- Gli oggetti in realtà sono **array associativi**
  - strutture composite i cui elementi sono accessibili mediante un **indice di tipo stringa** (nome) anziché attraverso un **indice numerico**
- Si può quindi utilizzare anche una sintassi analoga a quella degli array
- Le due sintassi sono del tutto equivalenti e si possono mescolare

```
var o = new Object();  
o.x = 7;  
o.y = 8;  
o.tot = o.x + o.y;  
alert(o.tot);
```

```
var o = new Object();  
o["x"] = 7;  
o.y = 8;  
o["tot"] = o.x + o["y"];  
alert(o.tot);
```



# Stringhe

---

- Non è facile capire esattamente cosa sono le stringhe in JavaScript
- Potremmo dire che mentre in Java sono oggetti che sembrano dati di tipo primitivo in JavaScript sono **dati di tipo primitivo che sembrano oggetti**
- Sono sequenze arbitrarie di caratteri in formato UNICODE a 16 bit e sono immutabili come in Java
- Esiste la possibilità di definire costanti stringa (**string literal**) delimitate da apici singoli ( `'ciao'` ) o doppi ( `"ciao"` )
- È possibile la concatenazione con l'operatore **+**
- È possibile la comparazione con gli operatori **<, >, >=, <= e !=**

## Stringhe come oggetti?

---

- Possiamo però invocare metodi su una stringa o accedere ai suoi attributi
- Possiamo infatti scrivere

```
var s = "ciao";  
var n = s.length;  
var t = s.charAt(1);
```
- Non sono però oggetti e la possibilità di trattarli come tali nasce da due caratteristiche:
  - Esiste un tipo **wrapper String** che è un oggetto
  - JavaScript fa il boxing in automatico come C#
    - quando una variabile di tipo valore necessita essere convertita in tipo riferimento, un oggetto box è allocato per mantenere tale valore

# Espressioni regolari

---

- JavaScript ha un supporto per le espressioni regolari (**regular expressions**) che sono un tipo di dato nativo del linguaggio
- Come per gli altri tipi di dato, esistono le costanti di tipo espressione regolare (**regexp literal**) con la sintassi

*/ expression /*

- Una espressione regolare può essere creata anche mediante il costruttore **RegExp( )**:

```
var r = /[abc]/;
```

```
var r = new RegExp( "[abc]" );
```

# Tipi valore e tipi riferimento

---

- Si può tentare di interpretare il sistema dei tipi di JavaScript usando una logica simile a quella di C#
- Si può quindi distinguere fra **tipi valore** e **tipi riferimento**
  - Numeri e booleani sono tipi valore
  - Array e Oggetti sono tipi riferimento
- Per le stringhe abbiamo ancora una situazione incerta
  - Pur essendo un tipo primitivo si comportano come un tipo riferimento
- **Le stringhe Javascript sono l'equivalente informatico dell'ornitorinco!**



# Funzioni

- Una funzione è un frammento di codice JavaScript che viene definito una volta e usato in più punti
  - Ammette parametri che sono privi di tipo
  - Restituisce un valore il cui tipo non viene definito
- La mancanza di tipo è coerente con la scelta fatta per le variabili
- Le funzioni possono essere definite utilizzando la parola chiave **function**
- Una funzione può essere assegnata ad una variabile

```
function sum(x,y)
{
    return x+y;
}
```

```
var s = sum(2,4);
```

# Costanti funzione e costruttore Function

- Esistono **costanti funzione** (**function literal**) che permettono di definire una funzione e poi di assegnarla ad una variabile con una sintassi decisamente inusuale:

```
var sum =  
    function(x,y) { return x+y; }
```

- Una funzione può essere anche creata usando un costruttore denominato **Function** (le funzioni sono quindi equivalenti in qualche modo agli oggetti)

```
var sum =  
    new Function("x","y","return x+y;");
```

# Metodi

---

- Quando una funzione viene assegnata ad una proprietà di un oggetto viene chiamata metodo dell'oggetto
- La cosa è possibile perché, come abbiamo visto, una funzione può essere assegnata ad una variabile
- In questo caso all'interno della funzione si può utilizzare la parola chiave **this** per **accedere all'oggetto di cui la funzione è una proprietà**
- Costruiamo un oggetto con 2 attributi e un metodo

```
var o = new Object();  
o.x = 7;  
o.y = 8;  
o.tot = function() { return this.x + this.y; }  
alert(o.tot());
```

# Costruttori

- Un costruttore è una funzione che ha come scopo quello di costruire un oggetto
- Se viene invocato con **new** riceve l'oggetto appena creato e può aggiungere proprietà e metodi
- L'oggetto da costruire è accessibile con la parola chiave **this**
- In qualche modo definisce il tipo di un oggetto

```
function Rectangle(w, h)
{
  this.w = w;
  this.h = h;
  this.area = function()
    { return this.w * this.h; }
  this.perimeter = function()
    { return 2*(this.w + this.h); }
}
```

```
var r = new Rectangle(5,4);
alert(r.area());
```



## Proprietà e metodi statici

---

- JavaScript ammette l'esistenza di proprietà e metodi statici con lo stesso significato di Java
- Non esistendo le classi sono associati al costruttore
- Per esempio, se abbiamo definito il costruttore `Circle()` che serve per creare oggetti di tipo cerchio, possiamo aggiungere l'attributo `PI` in questo modo:

```
function Circle(r)
{
    this.r = r;
}
Circle.PI = 3.14159;
```

- Anche in Javascript esiste l'oggetto **Math** che definisce solo metodi statici corrispondenti alle varie funzioni matematiche

# Ricapitolando

---

- In Javascript abbiamo solo tipi primitivi e oggetti
- I tipi primitivi sono numeri, booleani e stringhe (**forse!**)
- Tutte le altre cose sono oggetti:
  - **Oggetti generici**
    - quelli vuoti creati con `new Object()`
  - **Funzioni**
  - **Array**
  - **Espressioni regolari**
  - **Oggetti predefiniti**: Date, Math, Document, ecc.
  - **Oggetti wrapper**: String, Number, Boolean
  - **Oggetti definiti dall'utente** mediante definizione di un costruttore

# Operatori

---

- JavaScript ammette tutti gli operatori presenti in C e in Java
- Valgono le stesse regole di priorità e associatività
- Esistono alcuni operatori tipici
  - **delete**: elimina una proprietà di un oggetto
  - **void**: valuta un'espressione senza restituire alcun valore
  - **typeof**: restituisce il valore di un operando
  - **===**: identità o uguaglianza stretta (diverso da **==** che verifica l'uguaglianza)
  - **!==**: non identità (diverso da **!=**)

# Istruzioni

---

- Un **programma JavaScript** è una **sequenza di istruzioni**
- Buona parte delle istruzioni JavaScript hanno la stessa sintassi di C e Java
- Si dividono in:
  - **Espressioni** (uguali a Java): assegnamenti, invocazioni di funzioni e metodi, ecc.
  - **Istruzioni composte**: blocchi di istruzioni delimitate da parentesi graffe (uguali a Java)
  - **Istruzione vuota**: punto e virgola senza niente prima
  - **Istruzioni etichettate**: normali istruzioni con un etichetta davanti (sintassi: *label: statement*)
  - **Strutture di controllo**: **if**, **for**, **while**, ecc.
  - **Definizioni e dichiarazioni**: **var**, **function**
  - **Istruzioni speciali**: **break**, **continue**, **return**

## Strutture di controllo

---

- `if/else`, `switch`, `while`, `do/while` e `for` funzionano come in C e Java
- La struttura `for/in` permette di **scorrere le proprietà di un oggetto** (e quindi anche un array) con la sintassi: *`for (variable in object) statement`*

```
var x;  
var mycars = new Array();  
mycars[0] = "Panda";  
mycars[1] = "Uno";  
mycars[2] = "Punto";  
mycars[3] = "Clio";  
for (x in mycars)  
{  
    document.write(mycars[x]+"<br />");  
}
```

## L'oggetto globale e funzioni predefinite

---

- In JavaScript esiste un **oggetto globale** implicito
- Tutte le variabili e le funzioni definite in una pagina appartengono all'oggetto globale
- Possono essere utilizzate senza indicare questo oggetto
- Questo oggetto espone anche alcune funzioni predefinite:
  - **eval(expr)** valuta la stringa expr (che contiene un'espressione Javascript)
  - **isFinite(number)** dice se il numero è finito
  - **isNaN(testValue)** dice se il valore è NaN
  - **parseInt(str [,radix])** converte la stringa str in un intero (in base radix - opzionale)
  - **parseFloat(str)**: converte la stringa str in un numero

# Inserimento di JavaScript in una pagina HTML

---

- HTML prevede un apposito tag per inserire script; la sua sintassi è `<script> <!-- script-text  
//--> </script>`
- Il commento HTML (`<!-- //-->`) che racchiude il testo dello script serve per gestire la compatibilità con i browser che non gestiscono JavaScript
  - In questi casi il contenuto del tag viene ignorato
- La sintassi completa prevede anche la definizione del **tipo di script** definito (Javascript è il default per gran parte dei browser); si può fare in due modi:

`<script language="Javascript">` (deprecato) oppure

`<script type="text/javascript">` (rif. HTML 4) (deprecato)

`<script type="application/javascript">` (rif. HTML 5)

! `text/javascript` è stato deprecato e sostituito da `application/javascript`; a solo ciò “leggibile” dall’uomo fu riservato il MIME type `text/*`.

In HTML 5 il parametro `type` è opzionale (Javascript è il default)

# Script interni ed esterni

---

- Nell'uso del tag `<script>` abbiamo due possibilità:
  - **Script esterno:** il tag contiene il riferimento ad un file con estensione `.js` che contiene lo script:  

```
<SCRIPT language="Javascript" src="nomefile.js">
</SCRIPT>
```
  - **Script interno:** lo script è contenuto direttamente nel tag:  

```
<script type="text/javascript">
    alert("Hello World!");
</script>
```
- Un'altra forma di script interno, ancora più integrata con HTML, è il codice di risposta agli eventi che vedremo nel seguito



## Considerazione sugli script interni

---

- Se lo script è interno può essere inserito sia nell'intestazione che nel body
- Una pagina HTML viene eseguita in ordine sequenziale, dall'alto verso il basso, per cui:
  - gli script di intestazione vengono caricati prima di tutti gli altri
  - quelli nel body vengono eseguiti secondo l'ordine di caricamento
- Una variabile o qualsiasi altro elemento Javascript può essere richiamato solo se caricato in memoria:
  - ciò che si trova nell'header è visibile a tutti gli script del body
  - quello che si trova nel body è visibile solo agli script che lo seguono

# Gestire l'assenza di Javascript

---

- Ci sono browser che non gestiscono JavaScript
  - es. browser dei cellulari
    - fanno eccezione, ad esempio, Opera Mobile, Bolt, NetFront Web Browser, ... 😊
- Un utente può disabilitare Javascript (per esempio per motivi di sicurezza)
- HTML prevede un tag (**<noscript>**) da inserire in testata per gestire contenuti alternativi in caso di non disponibilità di Javascript
- Ad esempio:

```
<noscript>  
<meta http-equiv="refresh" content=  
                                "0;url=altrapagina.htm">  
</noscript>
```

# Cosa si può fare con JavaScript

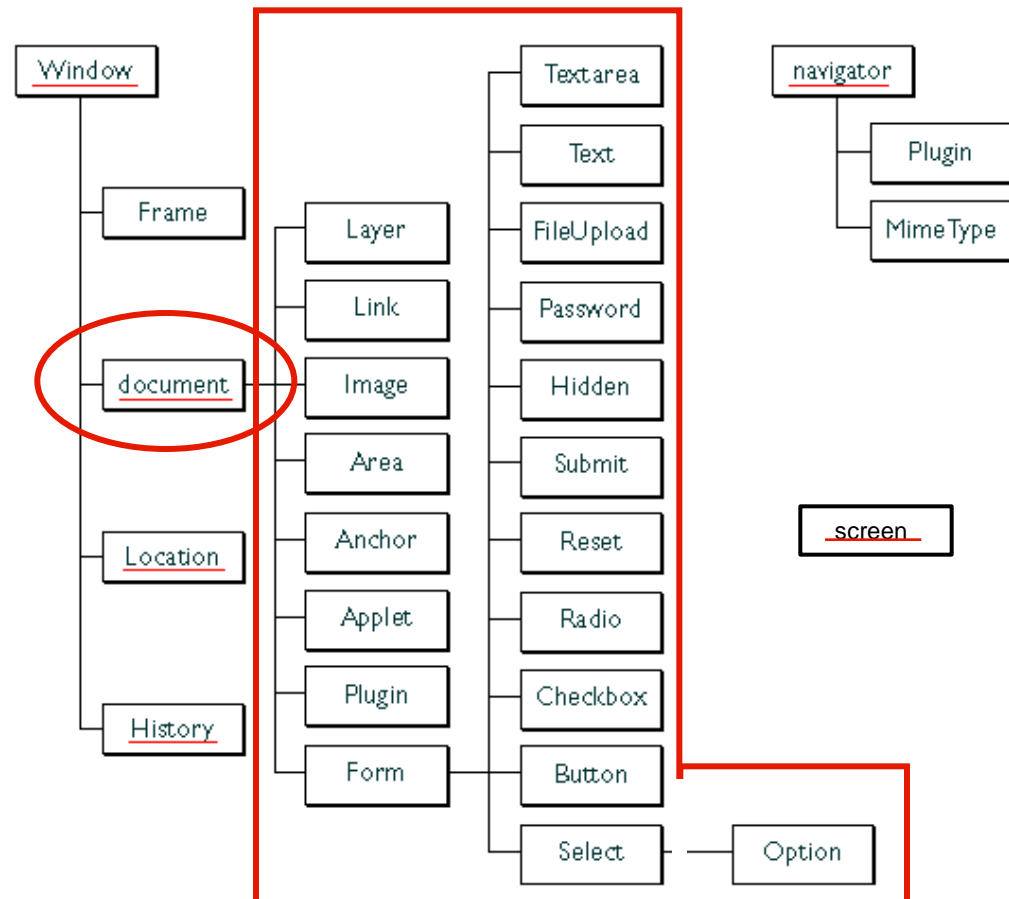
---

- Con JavaScript si possono fare essenzialmente quattro cose
  - **Costruire dinamicamente parti della pagina** in fase di caricamento
  - **Rilevare informazioni sull'ambiente** (tipo di browser, dimensione dello schermo, ecc.)
  - **Rispondere ad eventi** generati dall'interazione con l'utente
  - **Modificare dinamicamente il DOM** (si parla in questo caso di **Dynamic HTML** o **DHTML**)
- Tipicamente gli script agiscono su più aspetti in modo coordinato: ad esempio, modificando il DOM in risposta ad un evento

# Browser Objects

- Per interagire con la pagina HTML , Javascript utilizza una gerarchia di oggetti predefiniti denominati **Browser Objects** e **DOM Objects**

La gerarchia che ha come radice *document* corrisponde al DOM



## Costruzione dinamica della pagina

---

- La più semplice modalità di utilizzo di JavaScript consiste nell'inserire nel corpo della pagina **script che generano dinamicamente parti della pagina HTML**
- Bisogna tener presente che questi script vengono eseguiti solo una volta durante il caricamento della pagina e quindi **non si ha interattività con l'utente**
- L'uso più comune è quello di generare **pagine diverse in base al tipo di browser o alla risoluzione dello schermo**
- La **pagina corrente** è rappresentata dall'oggetto **document**
- Per scrivere nella pagina si utilizzano i metodi **document.write()** e **document.writeln()**

# Rilevazione del browser

- Per accedere ad informazioni sul browser si utilizza l'oggetto **navigator** che espone una serie di proprietà:

Proprietà	Descrizione
<b>appName</b>	Nome in codice del browser (poco utile)
<b>appName</b>	Nome del browser (es. Microsoft Internet Explorer)
<b>appVersion</b>	Versione del Browser (es. 5.0 (Windows))
<b>cookieEnabled</b>	Dice se i cookies sono abilitati
<b>platform</b>	Piattaforma per cui il browser è stato compilato (es. Win32)
<b>userAgent</b>	Stringa passata dal browser come header user-agent (es. "Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;)") È possibile esplorare la proprietà userAgent per mobile browser quali iPhone, iPad, o Android.

```
<html>
  <body>
    <script>
      document.write('Hello '+navigator.appName+'!<br>');
      document.write('Versione: '+navigator.appVersion+'<br>');
      document.write('Piattaforma: '+navigator.platform);
    </script>
  </body>
</html>
```

## Rilevazione delle proprietà dello schermo

- L'oggetto **screen** permette di ricavare informazioni sullo schermo
- **screen** espone alcune utili proprietà tra cui segnaliamo **width** e **height** che permettono di ricavarne le dimensioni

```
<html>
  <body>
    <script>
      document.write( 'Schermo:
        '+screen.width+'x'+screen.height+' pixel<br>' );
    </script>
  </body>
</html>
```

Schermo: 1360x768 pixel

# Modello ad eventi e interattività

---

- Per avere una **reale interattività** bisogna utilizzare il **meccanismo degli eventi**
- JavaScript consente di **associare script agli eventi causati dall'interazione dell'utente** con la pagina HTML
- L'associazione avviene mediante attributi collegati agli elementi della pagina HTML
- Gli script prendono il nome di **gestori di eventi (event handlers)**
- Nelle risposte agli eventi si può intervenire sul DOM modificando dinamicamente la struttura della pagina (**DHTML**) **DHTML = JavaScript + DOM + CSS**
- È un **modello di tipo reattivo** simile a quello di Swing o delle applicazioni Windows sviluppate con .NET



# Eventi - 1

Evento	Applicabilità	Occorrenza	Event handler
Abort	Immagini	L'utente blocca il caricamento di un'immagine	onAbort
Blur	Finestre e tutti gli elementi dei form	L'utente toglie il focus a un elemento di un form o a una finestra	onBlur
Change	Campi di immissione di testo o liste di selezione	L'utente cambia il contenuto di un elemento	onChange
Click	Tutti i tipi di bottoni e i link	L'utente 'clicca' su un bottone o un link	onClick
DragDrop	Finestre	L'utente fa il drop di un oggetto in una finestra	onDragDrop
Error	Immagini, finestre	Errore durante il caricamento	onError
Focus	Finestre e tutti gli elementi dei form	L'utente dà il focus a un elemento di un form o a una finestra	onFocus
KeyDown	Documenti, immagini, link, campi di immissione di testo	L'utente preme un tasto	onKeyDown
KeyPress	Documenti, immagini, link, campi di immissione di testo	L'utente digita un tasto (pressione + rilascio)	onKeyPress
KeyUp	Documenti, immagini, link, campi di immissione di testo	L'utente rilascia un tasto	onKeyUp

## Eventi - 2

Evento	Applicabilità	Occorrenza	Event handler
Load	Corpo del documento	L'utente carica una pagina nel browser	onLoad
MouseDown	Documenti, bottoni, link	L'utente preme il bottone del mouse	onMouseDown
MouseMove	Di default nessun elemento	L'utente muove il cursore del mouse	onMouseMove
MouseOut	Mappe, link	Il cursore del mouse esce fuori da un link o da una mappa	onMouseOut
MouseOver	Link	Il cursore passa su un link	onMouseOver
MouseUp	Documenti, bottoni, link	L'utente rilascia il bottone del mouse	onMouseUp
Move	Windows	La finestra viene spostata	onMove
Reset	Form	L'utente resetta un form	onReset
Resize	Finestre	La finestra viene ridimensionata	onResize
Select	Campi di immissione di testo (input e textarea)	L'utente seleziona il campo	onSelect
Submit	Form	L'utente sottomette il form	onSubmit
Unload	Corpo del documento	L'utente esce dalla pagina	onUnload

## Gestori di evento

---

- Come si è detto, per «agganciare» un gestore di evento ad un evento si utilizzano gli attributi degli elementi HTML

- La sintassi è:

*<tag eventHandler="JavaScript Code">*

- Esempio:

```
<input type="button" value="Calculate"
  onClick='alert("Calcolo")' />
```

- È possibile inserire più istruzioni in sequenza, ma è **meglio definire delle funzioni** (in testata)

**!** È sempre necessario **alternare** doppi apici e apice singolo

```
<input type="button" value="Apriti sesamo!"
  onClick="window.open( 'myDoc.html' , 'newWin' )" >
```

# Esempio: calcolatrice

```
<head>
  <script type="text/javascript">
    function compute(f)
    {
      if (confirm("Sei sicuro?"))
        f.result.value = eval(f.expr.value);
      else alert("Ok come non detto");
    }
  </script>
</head>
<body>
  <form>
    Inserisci un'espressione:
    <input type="text" name="expr" size=15 >
    <input type="button" value="Calcola"
      onClick="compute(this.form)"><br/>
    Risultato:
    <input type="text" name="result" size="15" >
  </form>
</body>
```

Inserisci un'espressione: 3\*2

Calcola

Risultato: 6

# Esplorare il DOM: Document

---

- Il punto di partenza per accedere al Documento Object Model (DOM) della pagina è l'oggetto **document**
- **Document** espone 4 collezioni di oggetti che rappresentano gli elementi di primo livello:
  - **anchors[ ]**
  - **forms[ ]**
  - **images[ ]**
  - **links[ ]**
- L'accesso agli elementi delle collezioni può avvenire per indice (ordine di definizione nella pagina) o per nome (attributo name dell'elemento):  
**document.links[ 0 ]**  
**document.links[ "nomelink" ]**
- In base all'**equivalenza tra array associativi e oggetti** la seconda forma può essere scritta anche come **document.nomelink**

## Document - 2

---

- Metodi:
  - **getElementById( )**: restituisce un riferimento al primo oggetto della pagina avente l'id specificato come argomento
  - **write( )**: scrive un pezzo di testo nel documento
  - **writeln( )**: come write() ma aggiunge un a capo
- Proprietà:
  - **bgcolor**: colore di sfondo
  - **fgcolor**: colore di primo piano
  - **lastModified**: data e ora di ultima modifica
  - **cookie**: tutti i cookies associati al document
    - rappresentati da una stringa di coppie: nome-valore
  - **title**: titolo del documento
  - **URL**: url del documento

## Form - 1

---

- Un documento può contenere più oggetti form
- Un **oggetto form** può essere referenziato con il suo nome o mediante il vettore **forms[ ]** esposto da **document**:

```
document.nomeForm
```

```
document.forms[n]
```

```
document.forms["nomeForm"]
```

- Gli **elementi** del form possono essere referenziati con il loro nome o mediante il vettore **elements[ ]**

```
document.nomeForm.nomeElemento
```

```
document.forms[n].elements[m]
```

```
document.forms["nomeForm"].elements["nomeElem"]
```

- Ogni elemento ha una **proprietà form** che permette di accedere al form che lo contiene (vedi esempio “calcolatrice” precedente **this.form**)

## Form - 2

- Per ogni elemento del form esistono proprietà corrispondenti ai vari attributi:  
**id, name, value, type, className...**

```
<form name="myForm">  
  Form name:  
  <input type="text" name="text1" value="test">  
  <br/>  
  <input name="button1" type="button"  
    value="Mostra il nome del form"  
    onclick="document.myForm.text1.value=  
              document.myForm.name">  
</form>
```

Form name:

In alternativa potevamo scrivere:

```
onclick="this.form.text1.value=  
          this.form.name">
```



## Form - 3

---

- Proprietà:
  - **action**: riflette l'attributo action
  - **elements**: vettore contenente gli elementi della form
  - **length**: numero di elementi nella form
  - **method**: riflette l'attributo method
  - **name**: nome del form
  - **target**: riflette l'attributo target
- Metodi:
  - **reset()**: resetta il form
  - **submit()**: esegue il submit
- Eventi:
  - **onreset**: quando il form viene resettato
  - **onsubmit**: quando viene eseguito il submit del form

# I controlli di un form

---

- Ogni tipo di **controllo** (**widget**) che può entrare a far parte di un form è rappresentato da un oggetto JavaScript:
  - **Text:** `<input type = "text">`
  - **Checkbox:** `<input type="checkbox">`
  - **Radio:** `<input type="radio">`
  - **Button:** `<input type="button">` o `<button>`
  - **Hidden:** `<input type="hidden">`
  - **File:** `<input type="file">`
  - **Password:** `<input type="password">`
  - **Textarea:** `<textarea>`
  - **Submit:** `<input type="submit">`
  - **Reset:** `<input type="reset">`

# Elementi comuni ai vari controlli

---

- **Proprietà:**
  - **form**: riferimento al form che contiene il controllo
  - **name**: nome del controllo
  - **type**: tipo del controllo
  - **value**: valore dell'attributo value
  - **disabled**: disabilitazione/abilitazione del controllo
- **Metodi:**
  - **blur()** toglie il focus al controllo
  - **focus()** dà il focus al controllo
  - **click()** simula il click del mouse sul controllo
- **Eventi:**
  - **onblur** quando il controllo perde il focus
  - **onfocus** quando il controllo prende il focus
  - **onclick** quando l'utente clicca sul controllo

## L'oggetto Text (e Password)

---

- Proprietà (get/set):
  - `defaultValue` valore di default
  - `Disabled` disabilitazione / abilitazione del campo
  - `maxLength` numero massimo di caratteri
  - `readOnly` sola lettura / lettura e scrittura
  - `size` dimensione del controllo
- Metodi:
  - `select()` seleziona una parte di testo

# Oggetti Checkbox e Radio

---

- Proprietà (get/set):
  - **checked**: dice se il box è spuntato
  - **defaultChecked**: impostazione di default

# Validazione di un form

---

- Uno degli utilizzi più frequenti di JavaScript è nell'ambito della **validazione dei campi di un form**
  - Riduce il carico delle applicazioni server side filtrando l'input
  - Riduce il ritardo in caso di errori di inserimento dell'utente
  - Semplifica le applicazioni server side
  - Consente di introdurre dinamicità all'interfaccia Web
- Generalmente si valida un form in due momenti:
  - Durante l'**inserimento** utilizzando l'evento **onChange( )** sui vari controlli
  - Al momento del **submit** utilizzando l'evento **onClick( )** del bottone di submit o l'evento **onSubmit( )** del form

## Esempio di validazione - 1

```
<head>
  <script type="text/javascript">
    function qty_check(item, min, max)
    {
      returnVal = false;
      if (parseInt(item.value) < min) or
        (parseInt(item.value) > max)
        alert(item.name+"deve essere fra "+min+" e "+max);
      else returnVal = true;
      return returnVal;
    }
    function validateAndSubmit(theForm)
    {
      if (qty_check(theform.quantità,0,999))
      { alert("Ordine accettato"); return true; }
      else
      { alert("Ordine rifiutato"); return false; }
    }
  </script>
</head>
```

## Esempio di validazione - 2

```
<body>
  <form name="widget_order"
    action="lwapp.html" method="post">
    Quantità da ordinare
    <input type="text" name="quantità"
      onchange="return qty_check(this,0,999)">
    <br/>
    <input type="submit" value="Trasmetti l'ordine"
      onclick="return validateAndSubmit(this.form)">
  </form>
</body>
```

```
<form name="widget_order"
  action="lwapp.html" method="post"
  onsubmit="return qty_check(this.quantita,0,999)">
  ...
  <input type="submit" />
  ...
</form>
```



## Esempio 2

```
<head>
  <script>
    function upperCase()
    {
      var val = document.myForm.firstName.value;
      document.myForm.firstName.value = val.toUpperCase();
      val = document.myForm.lastName.value;
      document.myForm.lastName.value = val.toUpperCase();
    }
  </script>
</head>
<body>
  <form name="myForm">
    <b>Nome: </b>
    <input type="text" name="firstName" size="20" /><br/>
    <b>Cognome: </b>
    <input type="text" name="lastName" size="20"/>
    <p><input type="button" value="Maiuscolo"
      onClick="upperCase()" /></p>
  </form>
</body>
```

# JavaScript e JQuery

---

- Sebbene molti aspiranti sviluppatori Web si chiedano quale sia la differenza tra **JavaScript** e **JQuery**, in realtà è bene sapere che, in pratica, sono la stessa cosa!! 😊
- Sinteticamente, **JQuery** è una libreria JavaScript (*sviluppata da terzi*) pensata appositamente per semplificare la vita del programmatore Web
- Nel dettaglio, semplifica e velocizza
  - l'attraversamento del DOM di una pagine HTML,
  - la sua animazione,
  - la gestione di eventi, e
  - le interazioni Ajax

mediante una “*easy-to-use*” API che funziona per una moltitudine di Web browser

## JavaScript e JQuery: quali differenze?

---

- Mediante una giusta combinazione di versatilità e estensibilità, JQuery ha cambiato il modo di scrivere codice JavaScript
- Prima di JQuery, gli sviluppatori provvedevano a creare il proprio “framework JavaScript”; ciò permetteva loro di lavorare su specifici bug senza perdere tempo nel debugging di features comuni
- Ciò ha portato alla realizzazione da parte di gruppi di sviluppatori di librerie open source e gratuite
- Con JQuery, lo sviluppatore usa API JavaScript preconfezionate «pronte all’uso»

! ...Durante l’esercitazione guidata in laboratorio, faremo un’**esperienza diretta sull’uso delle API JQuery**, come alternativa al linguaggio nativo JavaScript 😊

# Riferimenti

---

- **JavaScript:**
  - <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference>
  - <http://html.it/guide/guida-javascript-per-esempi/>
- **jQuery:** <http://api.jquery.com/>
- **Tutorial (JavaScript e HTML DOM):**
  - <http://www.w3schools.com/js/default.asp>
  - <http://www.w3schools.com/jsref/default.asp>