

## Introduzione Javascript

- **Javascript** è un linguaggio di scripting per pagine web e integrabile con HTML
- Consiste in un linguaggio interpretato dove noi scriviamo il nostro codice e l'interprete lo esegue. Gli errori li vedremo non quando salviamo ma quando eseguiamo quanto scritto (in contrapposizione al C++, per esempio)
- La bellezza di Javascript è la possibilità di eseguire eventi in risposta ad azioni svolte dall'utente.
- Javascript è semplice da imparare, è adatto per task ripetitive e per realizzare piccoli programmi.

**Attenzione:** quando andiamo a validare il codice valideremo solo HTML e CSS.

**Origine:** Javascript è stato inventato da Netscape e inizialmente usato solo nel browser di Netscape. Adesso tutti i browser supportano Javascript.

**Standard:** Oggi siamo all'11esima versione di Javascript. Lo standard attuale è l'ECMA-262, approvato dall'ISO.

### Debolezze

- Poche funzioni
- Linguaggio solo client-side (non più vero, esiste anche una versione lato server).
- Il codice non può essere nascosto.

### Esempi di obiettivi

Oltre a quanto già detto possiamo

- validare dati nei form
- gestire ricerche in una tabella
- recuperare le informazioni salvate nei cookies.

### Cosa non possiamo fare?

- Accedere e modificare file su server.
- Non posso accedere a documenti aperti in altre finestre del browser o frame.
- Non posso scrivere sull'hard disk locale, avviare una stampa o modificare le preferenze del browser.

L'unica eccezione a queste regole è la possibilità di leggere e scrivere i cookies, presenti sull'hard disk.

### Inserimento del Javascript in una pagina HTML

Per inserire codice Javascript utilizzeremo l'elemento *script*. Possiamo farlo

- indicando come valore dell'attributo type *text/javascript*. Il contenuto dell'elemento consiste nel codice Javascript L'elemento può essere posto
  - o nell'head (consigliato)
  - o nel body (caldamente sconsigliato)
- indicando come valore dell'attributo src l'indirizzo della risorsa contenente il codice javascript e come valore dell'attributo type *text/javascript*. Questa soluzione è quella consigliata.

**Attenzione:** non può esserci codice HTML nella pagina che stiamo includendo (abbiamo stabilito che il controllo della pagina viene passato all'interprete javascript)

### Apici e doppi apici

- Quando scriviamo il valore degli attributi HTML usiamo i doppi apici
- Quando scriviamo valori in Javascript, invece, utilizziamo i singoli apici

### ~~Indicare con meta il linguaggio di scripting~~ (DEPRECATA)

~~Indicare il linguaggio di scripting nel codice:~~

~~<meta http-equiv="Content-Script-Type" content="text/javascript">~~

### ~~Elemento noscript~~ (DEPRECATA)

~~Un elemento *noscript* permette di indicare un contenuto alternativo al codice javascript non eseguito.~~

### Quando viene eseguito il codice?

- **Global code:** codice nell'elemento head o in un file esterno. Viene eseguito durante il rendering della pagina.
- **Code in functions:** eseguito solo se viene chiamata la funzione
- **Event onLoad:** il codice viene eseguito quando la pagina è stata caricata dal browser e dopo l'esecuzione del codice globale.

## Codice Javascript

- Un codice Javascript è una sequenza di javascript statements. Il browser esegue questi statements nell'ordine in cui sono posti.
- Alla fine di ogni riga si pone punto e virgola (la cosa non è obbligatoria ma consigliata)
- Javascript è case-sensitive (keyword, nomi di funzioni, nomi di variabili...)
- Come in C++ le istruzioni javascript possono essere raggruppamenti in blocchi (mediante parentesi graffe)
- I commenti sono come in C++

## Regole per i nomi delle variabili

- Le variabili sono case-sensitive
- Il primo carattere deve essere una lettera o un underscore
- Il resto del nome può essere formato da lettere ma anche da numeri o altri underscore.
- La dichiarazione si introduce con la keyword *var*
- Le variabili appena dichiarate sono vuote.
- Le variabili possono essere inizializzate.
- Un'assegnazione di valore a una variabile non dichiarata porterà a una dichiarazione implicita della stessa
- Ridichiarare una variabile Javascript **non comporta** perderne il valore originario.
- Attenzione all'inizializzazione delle variabili: un conto è porre come valore un numero, un altro porre due numeri come stringa (tra doppi apici). L'operatore somma non ci restituisce la somma dei due numeri: il valore numerico è convertito in stringa e avviene la concatenazione delle due variabili.

## Tipi di valori

Il linguaggio non è tipizzato: questo significa che non avremo variabili di un certo tipo, non che non avremo valori di un certo tipo. I tipi possibili sono i seguenti:

- *numeri*. Numeri rappresentabili in base decimale, ottale ed esadecimale.
- *stringhe*. Insieme di 0 o più caratteri racchiusi in doppi o singoli apici (valide entrambe le notazioni). Presenti anche caratteri di escape: questi sono riconosciuti
  - o in elementi pre (la write aggiunge questi caratteri all'interno di un elemento pre)
  - o in textarea
  - o nei comandi alert(), confirm() e prompt()
- *booleani* (true o false, 1 e 0 non sono considerati booleani in Javascript)
- *null*
- *undefined* (o NaN, *not a number*)

Relativamente allo **scope** osserviamo che

- Dichiarare nuovamente una variabile nella funzione comporta nascondere temporaneamente una funzione globale.
- Questa cosa però avviene solo se poniamo var: senza var andiamo a modificare il valore di una variabile globale.

## Operatori in Javascript

- Operatori aritmetici uguali (pagina 38)
- Operatori di assegnamento uguali (pagina 39)
- Operatore + utilizzato con le stringhe per la concatenazione (come già visto)
- Operatori di confronto presentano delle differenze. Abbiamo due operatori:
  - o Per l'uguaglianza (con due uguale abbiamo l'uguaglianza al di là del tipo del valore, con tre uguale abbiamo un'uguaglianza stretta in cui si considera anche il tipo del valore)

```
var risposta = prompt("Indicare un valore");  
alert(typeof risposta); // prompt restituisce sempre una string
```

```
if(risposta == 1) { alert("Alert con 1 qualunque"); }  
if (risposta === 1) { // non avverrà mai, dovrei avere una risposta di tipo numerica  
    alert("Alert con 1 stringa");  
}
```

- o Per la disuguaglianza (come prima, disuguaglianza e disuguaglianza stretta con due e tre uguale, rispettivamente). In questo caso la disuguaglianza stretta restituisce true se
  - non si ha uguaglianza nel valore, e/o

- non si hanno elementi dello stesso tipo

```
var testo = '1';

if(testo !== 1) {
    alert("testo diverso da 1 numerico");
}
if (risposta !== 1) { // non avverrà
    alert("testo diverso da 1");
}
```

- Due variabili con valore NaN **NON** sono uguali

```
var num1 = NaN;
var num2 = NaN;
```

```
alert(num1 == num2); // sempre false
```

- Oggetti, array e funzioni sono confrontati per riferimento: si ha uguaglianza solo se si fa riferimento allo stesso elemento. Questo significa che due array separati non potranno essere mai uguali, anche se contengono gli stessi elementi.

- Se due valori sono entrambi nulli o undefined sono uguali.

Se un valore è nullo e un altro undefined si ha uguaglianza

```
var var1 = null;
var var2 = null;
var var3 = undefined;
var var4 = undefined;
```

```
alert(var1 == var2); // sempre true
```

```
alert(var3 == var4); // sempre true
```

```
alert(var1 == var3); // sempre true
```

- Operatori logici come in C++ (diapositiva 43 del Javascript).

### **Conversione di valori in operazioni di confronto**

Se i tipi dei due valori differiscono si cerca di convertire uno dei due in modo tale che si abbia lo stesso tipo.

Precisamente:

- se un valore è un numero e un altro è una stringa si converte la stringa in numero e si tenta di nuovo il confronto
 

```
var first = 1;
var second = '2';
if(first < second)
    alert('First è più piccolo di second');
```
- se uno dei due valori è true si converte questo in 1 e si tenta di nuovo il confronto.  
se uno dei due valori è false si converte questo in 0 e si tenta di nuovo il confronto.
 

```
alert(true > 2); //false
alert(true > 0); //true
```
- se un valore è un oggetto e un altro è un numero o una stringa si converte l'oggetto in un valore primitivo.
- Qualunque altra combinazione di tipi non può essere uguale neanche passando attraverso conversioni.

### **Operatore condizionale**

- Presente operatore condizionale, uguale in tutto e per tutto a quello già visto in C++  
*(condition)? val1 : val2*

### **Operatore typeof**

- L'operatore *typeof* restituisce una stringa che indica il tipo dell'operando posto come argomento.
- Come vediamo dall'esempio è possibile chiamare questo operatore in due modi.

```
var numero = 5; // typeof: number
```

```
var stringa = 'testo'; // typeof: string
var booleano = true; // typeof: boolean
var nullo1 = null; // typeof: object
var nullo2 = undefined; // typeof: undefined
var nullo3 = NaN; // typeof: number

alert(typeof numero);
alert(typeof(stringa));
```

### **Operatore void**

- L'operatore può essere espresso in due modi:
  - o void(expression)
  - o void expression
- L'operatore valuta un'espressione senza restituire valori (dice la diapositiva, in realtà restituisce undefined)
- **Esempio:**  
<a href="javascript:void(document.body.style.backgroundColor='red');">  
Click me to change the background color of body to red  
</a>

### **Istruzioni**

- Buona parte delle istruzioni sono uguali a quelle presenti in C++.
- Abbiamo visto:
  - o **if-statements**, con then-statement ed eventualmente else-statement

```
var risposta = confirm('Vuoi uscire?');

if(risposta == 1) { // Conversione della variabile risposta
    alert('Sei uscito!');
}
else {
    alert('Non sei uscito!');
}
```
  - o **switch-statement**

```
var risposta = confirm('Vuoi uscire?');

switch(risposta) {
    case true:
        alert('Sei uscito!');
        break;
    case false:
        alert('Non sei uscito!');
        break;
    default: // (non penso succeda, messo per dare tutto il costrutto)
        alert('Qualcosa è andato storto.');
```
  - o **for-statement**

```
for(var contatore = 0; contatore < 10; contatore++) {
    alert(contatore);
}

alert(contatore); // valore mantenuto pure con var dentro il for.
```
  - o **while-statement** (presente anche il do-while)

```
var contatore = 0;

while(contatore != 10) {
    alert(contatore++);
}
```

- *break-statement* e *continue-statement*. Novità interessante è la possibilità di gestire loop annidati in modo molto simile a quello visto a basi di dati con il linguaggio SQL).

```
var primonum = prompt('Numero finale del primo output');
var secondonum = prompt('Numero finale del secondo loop');

var contatore = 0;
primoloop : while(contatore <= primonum) {
    var contatore2 = 0;
    secondoloop : while(contatore2 <= secondonum) {
        alert(contatore + ' ' + contatore2);
        contatore2++;

        var azione = prompt('Azione');
        if(azione == 'esci_1')
            break secondoloop;
        else if(azione == 'esci_2')
            break primoloop;
    }
    contatore++;
}
```

- *with-statement*, con esso stabiliamo l'oggetto di default per una serie di statements. Nel corpo di questo statement potremo richiamare proprietà e/o funzioni dell'oggetto omettendo l'oggetto stesso (come se fossero delle funzioni o variabili globali)

```
var a, x, y;
var r=10;
with (Math) {
    a = PI * r * r;
    x = r * cos(PI);
    y = r * sin(PI/2);
}
```

Nell'esempio qua sopra abbiamo posto la costante *PI* e le funzioni *cos* e *sin* di *Math*.

- *for...in statement*, permette di scorrere gli elementi di un array o le proprietà di un oggetto.
 

```
for(variable in object) {
    code to be executed
}
```

  - Il codice nel body è eseguito una volta per ogni elemento o proprietà.
  - L'argomento *variable* può essere un elemento dell'array (non l'indice) o una proprietà di un oggetto (il nome, non il valore).
- *try...catch statement*

```
try {

}
catch (err if expression) {

}
```

  - Molto simile a quanto visto in C++: permette di testare un blocco di codice e gestire eventuali errori runtime.
  - Il codice si trova nel blocco *try*, mentre nei blocchi *catch* sono presenti i codici da eseguire in caso di cattura di un certo errore.
  - *err* consiste nell'oggetto *exception* (cioè quanto captato)
  - *expression* consiste in un'espressione che indica l'errore che vogliamo gestire. Può essere omessa (se poniamo solo *err* gestiremo qualunque errore o, dopo una serie di blocchi *catch*, eccezioni non gestite prima).

- *throw-statement*, istruzione con cui possiamo creare nuove eccezioni. L'eccezione può essere una stringa, un intero, un Boolean o un oggetto.

```
try {
    if(x > 10)
        throw "Err1";
    else if(x < 0)
        throw "Err2";
    else if(isNaN(x))
        throw "Err3";
}
catch(er) {
    if(er == "Err1")
        alert("Errore! Valore troppo alto");
    if(er == "Err2")
        alert("Errore! Valore troppo basso");
    if(er == "Err3")
        alert("Errore! Valore non numerico");
}
```

## **Funzioni**

- Le funzioni possono essere chiamate da un qualunque punto della pagina e anche in documenti esterni se necessario.
- I parametri sono passati da funzioni per valore. Non è necessario indicare il tipo degli argomenti delle funzioni.
- Il cambio di valore sui parametri non influisce sulle variabili globali, mentre un cambio di valore su variabili non parametriche comporta un cambiamento visibile globalmente.
- Gli argomenti di una funzione sono mantenuti nell'array *arguments*.

`arguments[i]`

`functionName.arguments[i]`

- Il numero totale di argomenti si ottiene mediante `arguments.length`. Definiamo una funzione in cui, dato un numero di partenza, si aggiungono una serie di numeri:

```
function adder(base /*, n2, ... */) {
    base = Number(base);
    for (var i = 1; i < arguments.length; i++) {
        base += Number(arguments[i]);
    }
    return base;
}
```

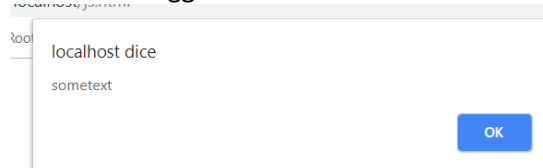
## **Funzioni predefinite**

Qua di seguito abbiamo alcune funzioni predefinite offerte da Javascript:

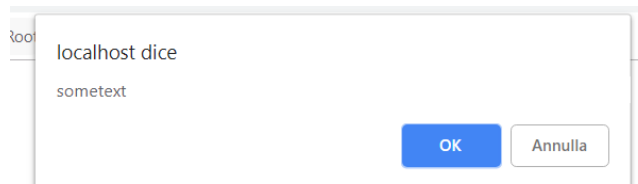
- `eval(string)`  
Valuta una stringa e la esegue come se fosse una parte di codice  
`eval('alert(\'ciao\')');` // (Esegue la funzione alert)
- `isFinite()`  
Restituisce un booleano con cui si dice se un valore è finito, cioè se è un valore legale dal punto di vista della sintassi
- `isNaN()`  
Restituisce un booleano con cui si dice se un valore è un numero non previsto nella nostra sintassi (NaN = Not A Number)
- `parseInt(string, radix)`  
Analizza una stringa e restituisce un intero della radice specificata. La radice è un numero compreso tra 32 e 36 che indica il sistema numerico adottato. Si restituisce NaN nel caso in cui non si possa convertire in un numero il primo carattere.  
`alert(parseInt('123456'));` // 123456  
`alert(parseInt('40 years'));` // 40

```
alert(typeof parseInt('40 years')); //number
alert(parseInt('1000', 2)); //Converto 1000 -> 8
```

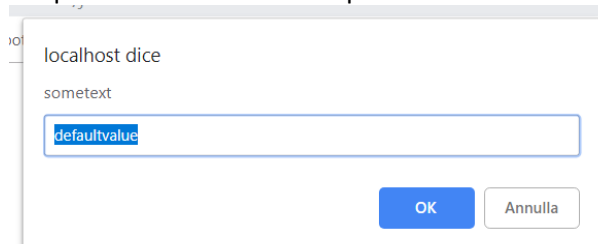
- `parseFloat(string)`  
Analizza una stringa e restituisce un numero float. Si restituisce NaN nel caso in cui non si possa convertire in un numero il primo carattere.
- `Number(object)` e `String(object)`  
convertito l'argomento oggetto in un numero o in una stringa che rappresenta il valore dell'oggetto. Se la conversione non può avvenire viene restituito NaN
- `escape(string)` e `unescape(string)`  
la prima codifica una stringa (la stringa diventa portabile e può essere trasmessa via rete a un qualunque computer che supporta la codifica ASCII). La decodifica (percorso in senso opposto) avviene con la `unescape`.
- `document.write(exp1, exp2, exp3)`  
Permette di aggiungere contenuto HTML all'interno del documento. Possibile indicare più argomenti.
- `document.writeln(exp1, exp2, exp3)`  
Stesso scopo della precedente, ma alla fine di ogni espressione si aggiunge una newline (ricordarsi che l'HTML ignora le newline a meno che non ci si trovi all'interno di un elemento `pre`). Possibile indicare più argomenti.
- `alert("sometext")`  
Ferma l'esecuzione e fa apparire un messaggio di allerta. Dobbiamo cliccare su OK per proseguire



- `confirm("sometext")`  
Comportamento simile alla precedente. La differenza è la possibilità di poter scegliere tra due bottoni: OK o Cancel. La funzione restituisce il valore indicato dall'utente.



- `prompt("sometext", "defaultvalue")`  
Usata per chiedere all'utente di inserire un input prima di entrare in una pagina. Se l'utente preme OK la funzione restituisce il valore di input indicato. Se l'utente preme Cancel la funzione restituisce null.



## Javascript Objects

- Javascript non ha ereditarietà (di questo ne riparleremo alla fine), non presenta proprietà o metodi privati (manca il concetto di classe e di *information hiding*). I nomi di oggetti e proprietà sono case sensitive.
- C++ è un linguaggio *object-oriented*, Javascript è un linguaggio *object-based*: non definiamo classi ma abbiamo oggetti come contenitori vuoti. Questi contenitori possono essere riempiti con proprietà e funzioni.

- **Come si definiscono le proprietà di un oggetto?** Assegnando alla proprietà un valore! Supponiamo esiste un oggetto chiamato myCar. Assegno le proprietà nel seguente modo:

```
myCar.make = "Ford";  
myCar.model = "Mustang";  
myCar.year = 1969;
```

- Si capisce che l'oggetto inizialmente è un contenitore vuoto.

- **Come accedo alle proprietà di un oggetto?** Abbiamo due vie

- o `objectName.propertyName` (classico)
- o `objectName["propertyName"]` (metodo ereditato dagli array associativi)

Attenzione: array associativi e oggetti in javascript sono interfacce differenti aventi la stessa struttura dati.

- I `for...in` statements possono essere usati per visitare i valori di tutte le proprietà di un oggetto

```
persona = { nome : 'Gabriele', cognome : 'Frassi', falza : false}  
for(var el in persona)  
    alert('Attributo ' + el + ' : ' + persona[el]);
```

- **Come si creano oggetti?** Abbiamo due vie

- o Usare l'inizializzatore di oggetto. Utile quando ci interessa creare un'unica istanza dell'oggetto.  
`objectName = { property1 : value1, ..., propertyN: valueN }`

dove *propertyX* è l'identificativo della proprietà ed *objectName* il nome del nuovo oggetto.

- Introdurre nuove proprietà dopo aver già inizializzato l'oggetto è possibile definendo il valore della singola nuova proprietà nel modo visto prima.
- Osservazione: una proprietà può essere a sua volta un oggetto, quindi possiamo creare oggetti annidati.

- o Usare una funzione costruttore e istanziare un oggetto con l'operatore *new*. Si pongono i valori delle proprietà dell'oggetto all'interno della funzione ricorrendo alla keyword *this*

```
function Car(make, model, year) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
}  
mycar = new Car("Eagle", "Talon TSi", 1993);  
kenscar = new Car("Nissan", "300ZX", 1992);  
vpgscar = new Car("Mazda", "Miata", 1990);
```

Quanto viste consiste in un certo senso nella simulazione di quanto avviene attraverso una classe.

Vale anche in questo caso l'annidamento di oggetti:

```
function Person(name, age, sex) {  
    this.name = name;  
    this.age = age;  
    this.sex = sex;  
}  
function Car(make, model, year, owner) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.owner = owner;  
}  
owner = new Person("Frankie Black", 45, "M");  
mycar = new Car("Eagle", "Talon TSi", 1993, owner);
```



### Operatore new

- L'operatore new permette di creare istanze di un oggetto vuoto definito da un utente.
- Può essere usato anche per creare oggetti di tipo predefinito (*array, boolean, date, function, image, number, object, regexp, string*).

```
objectName = new objectType (param1 [,param2]...[,paramN] );
```

### Proprietà prototype

- Possiamo creare nuove proprietà per ogni istanza di un object-type attraverso questo *prototype*.
- Si definiscono proprietà che sono condivise da tutti gli oggetti di un tipo specificato, piuttosto che da uno solo come abbiamo visto prima.

```
car.prototype.color = null;  
car1.color = "black";
```

Con la prima istruzione aggiungo una proprietà color a tutti gli oggetti di tipo car, mentre con la seconda definisco il valore della proprietà color di una particolare

### Funzioni associate ad oggetti

- Quando parlo di un oggetto non parlo solo di proprietà ma anche di funzioni.
- Possiamo associare a un oggetto una funzione nel seguente modo  
`object.methodName = function_name;`
- A questo punto potremo chiamare la funzione nel seguente modo  
`object.methodName(params);`

```
function displayCar(){  
    for (var a in this)  
        if (typeof(this[a])=="object")  
            for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);  
        else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);  
    document.writeln();  
}  
function Car(make, model, year, owner) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.owner = owner;  
    this.displayCar = displayCar;  
}
```

- Possiamo inizializzare una funzione anche nel seguente modo (metodo consigliato)

```
function Car(make, model, year, owner) {  
    this.make = make;  
    this.model = model;  
    this.year = year;  
    this.owner = owner;  
}  
Car.prototype.displayCar = function () {  
    for (var a in this)  
        if (typeof(this[a])=="object")  
            for (var i in this[a]) document.writeln(a + '.' + i + ':' + this[a][i]);  
        else if (typeof(this[a])!="function") document.writeln(a+':'+this[a]);  
    document.writeln();  
}
```

### Operatore delete

- Operatore che permette di cancellare proprietà/oggetti.
- Se l'operatore delete ha successo la proprietà o l'elemento assumono il valore undefined
- L'operatore restituisce true se l'operazione è possibile, false se non lo è.
- Questo operatore può essere usato solo per cancellare variabili definite implicitamente; **NON** può essere usato per cancellare variabili dichiarate esplicitamente con lo statement *var*

```
delete objectName; delete objectName.propertyName; delete variableName;
```

## Oggetti predefiniti

- Gli oggetti predefiniti *build-in* in Javascript sono:
  - o Array
  - o Boolean
  - o Date
  - o Function
  - o Math
  - o Number
  - o RegExp
  - o String
- Abbiamo anche oggetti *lato-client* generati direttamente dal browser quando viene fatto il parsing del documento HTML. Questi documenti sono gestiti direttamente dal browser. Essi sono:
  - o Navigator
  - o Window
  - o Document
  - o Location
  - o History
- **Osservazione:** tutti i nomi iniziano per maiuscola (stessa filosofia del nome delle classi in C++)

### Array

- Insieme ordinato di valori che si riferiscono attraverso un nome e un indice
- Gli elementi di un array possono essere di tipi diversi (contrariamente al C++, no tipizzazione forte)
- Gli array presentano funzioni che permettono la manipolazione: concatenare array, ordinare elementi...
- **Creazione dell'array:** possibile in tre modi diversi

```
var myCars=new Array(); // possibile intero per controllare la dimensione dell'array
myCars[0]="Saab";
myCars[1]="Volvo";
myCars[2]="BMW";
```

```
var myCars=new Array("Saab","Volvo","BMW");
```

```
var myCars=["Saab","Volvo","BMW"];
```

- **Accesso all'array:**  
`myCars[indice_elemento]`
- **Modifica del valore in un array:**  
`myCars[0] = "Opel";`
- **Estensione dinamica dell'array possibile:**  
`myCars[6] = "Fiat";` (abbiamo aggiunto il settimo elemento a un array che aveva sei elementi)
- **Scorrere tutto l'array:**  

```
oggetto = new Array(1,2,3,4,5,6,7,8,9);
for(x in oggetto)
    alert(x + ': ' + oggetto[x]);
```
- **Proprietà dell'array:**
  - o constructor (restituisce la funzione che ha creato il prototipo dell'oggetto)
  - o length (restituisce il numero di elementi nell'array)
  - o prototype (permette di aggiungere proprietà e metodi)
- **Funzioni membro:**
  - o `concat(array1, ..., arrayN)`  
concatenazione di due o più array  

```
array1 = new Array(1,2,3,4,5);
array2 = new Array(6,7,8,9,10);
```

```
array1 = array1.concat(array2); //(1,2,3,4,5,6,7,8,9,10)
```

- `join(separator)`  
equivalente della `implode` in PHP, si uniscono gli elementi dell'array in una stringa stabilendo un separatore. Se si omette il separatore gli elementi saranno divisi da virgole.  

```
array1 = new Array(1,2,3,4,5);  
var stringa = array1.join("|");  
alert(stringa); //"1|2|3|4|5"
```
- `pop()`  
si rimuove l'ultimo elemento dell'array, restituendolo.  

```
array1 = new Array(1,2,3,4,5);  
array1.pop(); //(1,2,3,4)
```
- `push(element1,..., elementN)`  
aggiunge nuovi elementi alla fine dell'array e restituisce la nuova length dell'array  

```
array1 = new Array(1,2,3,4,5);  
array1.push(6); //(1,2,3,4,5,6)
```
- `reverse()`  
inverte l'ordine degli elementi nell'array  

```
array1 = new Array(1,2,3,4,5);  
array1.reverse(); //(5,4,3,2,1)
```
- `shift()`  
rimuove il primo elemento dell'array e lo restituisce  

```
array1 = new Array(1,2,3,4,5);  
alert(array1.shift()); //1 -----> Array ottenuto: (2,3,4,5)
```
- `slice(start, end)`  
seleziona una parte dell'array e la restituisce come nuovo array.
  - La *start* è obbligatoria, e consiste nell'indice del primo elemento da considerare. La *start* può essere negativa (effetto Pacman)
  - La *end* è opzionale (se non si pone gli elementi considerati vanno da *start* fino alla fine dell'array) e consiste nell'indice del primo elemento da non considerare.

```
array1 = new Array("a", "b", "c", "d", "e");  
alert(array1.length); //5  
array2 = array1.slice(1, array1.length-1); //"b", "c", "d". ("e" è il primo el. da non considerare)
```
- `sort(nomeFunzione)`  
funzione che permette di ordinare elementi secondo un criterio stabilito dalla funzione *nomeFunzione*. Il nome della funzione può essere omissso se ci basta ordinare stringhe (NON NUMERI) in ordine alfabetico ascendente. Molto simile alla *sort* vista ad Algoritmi e strutture dati.
  - Criterio di ordinamento nella funzione:
    - se la funzione *nomeFunzione(a,b)* restituisce un valore negativo a viene posta prima di b;
    - se la funzione *nomeFunzione(a,b)* restituisce zero non si cambia niente;
    - se la funzione *nomeFunzione(a,b)* restituisce un valore positivo b viene posto prima di a.
- `splice(index, howmany, ..., elementX)`  
permette di aggiungere/rimuovere elementi da un array. Con *index* si indica l'indice del primo elemento considerato. Con *howmany* si indica il numero di elementi considerato (se si pone 0 non succederà niente). Successivamente si pongono i valori degli elementi che vogliamo aggiungere.  

```
array1 = new Array("a", "b", "c", "d", "e");  
array1.splice(2, 1, "4", "5", "6"); //(“a”, “b”, “4”, “5”, “6”, “d”, “e”)  
abbiamo indicato come primo elemento “c” (indice 2) dicendo di rimuovere soltanto un elemento (appunto “c”).
```

- o `toString()`  
si converte l'array in una stringa restituendo il risultato dell'operazione  
`array1 = new Array(1,2,"3",4,5);`  
`alert(array1.toString()); // "1,2,3,4,5"`
- o `unshift(element1, ..., elementN)`  
opposto della `shift`. Si aggiungono elementi all'inizio dell'array, restituendo la nuova *length* dell'array.  
`array1 = new Array(1,2,3,4,5);`  
`alert(array1.unshift(6,7)); // 7 -----> Array ottenuto: (6,7,1,2,3,4,5)`

## **Boolean**

- Oggetto il cui valore sarà `true` o `false`.  
`booleanObjectName = new Boolean(value);`
- Se l'oggetto booleano non ha nessun valore iniziale o se è uguale a 0, -0, null, "", false, undefined o NaN l'oggetto è uguale a `false`. In tutti gli altri casi il booleano è uguale a `true`!

```
booleano = new Boolean(5);
alert(booleano); // true
```

```
booleano = new Boolean(-1);
alert(booleano); // true
```

```
booleano = new Boolean(0);
alert(booleano); // false
```

```
booleano = new Boolean(null);
alert(booleano); // false
```

## **Date**

- Uno degli oggetti built-in più usati.
- Javascript memorizza le date come numero espresso in millisecondi dalla Epoch (January 1, 1970, 00.00.00)
- Possiamo creare l'oggetto in quattro modi:
  - o `new Date()` (Si ottiene la CURRENT DATE)
  - o `new Date(milliseconds)`
  - o `new Date(dateString)` (Data espressa nel formato "*Month day year hours:minutes:seconds*")
  - o `new Date(year, month, day, hours, minutes, seconds, milliseconds)`
- Per la seguente carrellata utilizzeremo  
`data_attuale = new Date();`  
`document.write(data_attuale); // Fri Oct 30 2020 09:45:46 GMT+0100 (Ora standard dell'Europa centrale)`
- **Funzioni membro:**
  - o `getDate()`  
giorno del mese  
`alert(data_attuale.getDate()); // 30`
  - o `getDay()`  
giorno della settimana  
`alert(data_attuale.getDay()); // 5 (non a caso è venerdì)`
  - o `getFullYear()`  
anno a quattro cifre  
`alert(data_attuale.getFullYear()); // 2020 (anno di merda)`

- `getHours()`  
**ora (tra 0 e 23)**  
`alert(data_attuale.getHours()); //9`
- `getMilliseconds()`  
**millisecondi (tra 0 e 999)**  
`alert(data_attuale.getMilliseconds()); //364`
- `getMinutes()`  
**minuti (tra 0 e 59)**  
`alert(data_attuale.getMinutes()); //45`
- `getMonth()`  
**mese (tra 0 e 11)**  
`alert(data_attuale.getMonth()); // Attenzione: mese attuale -1`
- `getSeconds()`  
**secondi (tra 0 e 59)**  
`alert(data_attuale.getSeconds()); //46`
- `getTime()`  
**millisecondi passati dalla Epoch.**  
`alert(data_attuale.getTime()); //1604047546000`

- Presenti una marea di altre funzioni dalla diapositiva 139. Si dia un'occhiata, in particolare, alle seguenti funzioni:
  - `parse()`  
equivalente dell'inizializzazione mediante argomento (quando si crea l'oggetto). Richiede obbligatoriamente una stringa rappresentante la data. La converte nel numero di millisecondi passati dalla Epoch.
  - `setDate()`  
imposta il giorno del mese (valore da 1 a 31)
  - `setFullYear()`  
imposta l'anno (anno in formato a quattro cifre)
  - `setHours()`  
imposta l'ora (valore da 0 a 23)
  - `setMilliseconds()`  
imposta i millisecondi (valore da 0 a 999)
  - `setMinutes()`  
imposta i minuti (valore da 0 a 59)
  - `setMonth()`  
imposta il mese (valore da 0 a 11)
  - `setSeconds()`  
imposta i secondi (valore da 0 a 59)
  - `setTime()`  
imposta una data aggiungendo o sottraendo una certa cifra (millisecondi)

- `toDatestring()`  
conversione della porzione relativa alla data in una stringa leggibile
- `toTimeString()`  
conversione della porzione relativa all'ora in una stringa leggibile
- `toLocaleDateString()`  
conversione della porzione relativa alla data in una stringa leggibile secondo le convenzioni locali
- `toLocaleTimeString()`  
conversione della porzione relativa all'ora in una stringa leggibile secondo le convenzioni locali
- `toLocaleString()`  
conversione dell'oggetto in una stringa leggibile secondo le convenzioni locali
- `toString()`  
conversione dell'oggetto in una stringa leggibile. Si osservi che  
`alert(typeof data_attuale); // object (l'alert funziona anche senza conversione)`  
`alert(typeof data_attuale.toString()); // string`
- `valueOf()`  
conversione dell'oggetto nel suo valore primitivo.

#### **Esempi relativi alle ultime funzioni di conversione in stringa**

```
data_attuale = new Date("Oct 30 2020 09:45:46");
alert(data_attuale.valueOf()); // Valore primitivo: 1604047546000
alert(data_attuale.toDateString()); // Fri Oct 30 2020
alert(data_attuale.toTimeString()); // 09:45:46 GMT+0100 (Ora standard dell'Europa centrale)
alert(data_attuale.toLocaleDateString()); // 30/10/2020
alert(data_attuale.toLocaleTimeString()); // 09:45:46
alert(data_attuale.toLocaleString()); // 30/10/2020, 09:45:46
alert(data_attuale.toString()); // Fri Oct 30 2020 09:45:46 GMT+0100 (Ora standard dell'Europa centrale)
```

### **Math**

- L'oggetto Math permette di svolgere operazioni matematiche.
- Math non è un costruttore ma è a tutti gli effetti un oggetto: noi utilizzeremo esclusivamente una cosa chiamata Math, non creeremo istanze come abbiamo fatto con gli oggetti precedenti.
- In questo oggetto contenitore possiamo trovare (lista completa dalla diapositiva 147):
  - Costanti matematiche
    - Nepero: E
    - PI: pigreco
  - Funzioni matematiche (abs, sin, cos, tan, acos, asin, atan, ceil, floor, max, min, random, round...)
- Riflessioni relativamente alla random():
  - La random restituisce un valore reale compreso tra 0 ed 1 (1 escluso)
  - Se vogliamo estrarre un valore tra 0 e 10 cosa possiamo fare?
    - Moltiplichiamo quanto restituito dalla random per 11 (estremo destro + 1)
    - Appliciamo la funzione floor.
    - Con questo metodo i numeri sono equiprobabili!

```
var numero_random = Math.floor(Math.random()*11);
```

### **Number**

- Costruttore che permette di racchiudere valori numerici primitivi.
- Le proprietà consistono in costanti numeriche (massimo valore e minimo valore rappresentabili, per esempio).
  - `MAX_VALUE`, il numero più grande possibile in Javascript

- *MIN\_VALUE*, il numero più piccolo possibile in Javascript
- ... lista completa alla diapositiva 151

- Si utilizza questo costruttore quando abbiamo bisogno di aggiungere proprietà aggiuntive ai valori numerici mediante **prototype**.

```
var num = new Number(value);
Number.prototype.newproperty = value;
```

- **Funzioni membro:**

- `toExponential(x)`  
si converte il numero in notazione esponenziale
- `toFixed(x)`  
si stampa il numero con x cifre dopo il punto decimale. L'argomento ha come valore default 0.
- `toPrecision(x)`  
si formatta il numero in modo tale che abbia lunghezza di x cifre. L'argomento, se omissso, restituisce il numero nella sua interezza.
- `toString()`  
si restituisce il numero sottoforma di stringa.
- `valueOf()`  
si restituisce il valore primitivo dell'oggetto.

#### **Esempi di applicazioni:**

```
var num = new Number(13.3714);
document.write(Number.MAX_VALUE+"<br>"); // 1.7976931348623157e+308
document.write(Number.MIN_VALUE+"<br>"); // 5e-324
document.write(num.toExponential(4)+"<br>"); // 1.3371e+1
document.write(num.toFixed(2)+"<br>"); // 13.37
document.write(num.toPrecision(3)+"<br>"); // 13.4
document.write(num.toString()+"<br>"); // "13.3714"
document.write(typeof num.valueOf()); // 13.3714
```

### **RegExp**

- *Pongo prima della string considerando che alcune funzioni della string richiedono le regular expressions*
- Le espressioni regolari sono oggetti che descrivono un pattern di caratteri.

- **Utilità:** controlli di sintassi lato client in Javascript (form)
- Le espressioni regolari possono essere create in due modi

```
var txt=new RegExp(pattern, modifiers);
var txt=/pattern/modifiers;
```

- **Gli argomenti sono:**

- *pattern*, ciò che vogliamo trovare
- *modifiers*, si specifica se la ricerca dovrà essere globale, case-sensitive, ... I valori possibili sono:
  - *i*, matching case-insensitive
  - *g*, match globale (trovo tutti i possibili match, non mi fermo al primo)
  - *m*, (match multilinea, si analizzano testi su più linee)

- **Funzioni membro:**

- `exec()`, si testa se è presente un match in una stringa. Si restituisce il primo match.
- `test()`, stessa cosa di prima. Si restituisce true o false.

- **Come si scrivono le espressioni regolari?**

- [abc], set di caratteri che ci interessa trovare (in questo caso trovo a, b, oppure c)

```
/[abc]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com, and is proudly hosted by Media Temple.

- [^abc], ci interessano tutti i caratteri tranne quelli posti tra parentesi quadrate

```
/[^abc]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com, and is proudly hosted by Media Temple.

- [0-9], cifre tra 0 e 9

```
/[0-9]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- [a-z], caratteri lowercase da a a z

```
/[a-z]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- [A-Z], caratteri uppercase da A a Z

```
/[A-Z]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- [A-z], caratteri dalla A uppercase alla z lowercase (Marcelloni ha messo prima la lowercase, sembra sbagliato...)

```
/[A-a-z]/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- adgk, trovare sequenza di caratteri (tutti insieme, non singoli caratteri come abbiamo visto con le parentesi quadre)

```
/Reg/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.



- (contenuto), le parentesi tonde permettono di selezionare una sequenza di caratteri consecutivi.

```
/(RegExr)/g
```

Text Tests **NEW**

RegExr was created by gskinner.com, and is proudly hosted by Media Temple.

- (red | blue | green), trovare una delle alternative specificate. La barra verticale è sinonimo di OR

```
/(Reg|gskinner|[1-8])/g
```

Text Tests **NEW**

RegExr was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- **All'interno delle espressioni possiamo utilizzare i cosiddetti *metacaratteri*:**

- ., per trovare un singolo carattere (tranne newline o line terminator)
- \w, per trovare un carattere stringa

```
/\w/g
```

Text Tests **NEW**

RegExr was created by gskinner.com, and is proudly hosted by Media Temple.  
Explore results with the Tools below. Replace & List output custom results. Details lists English.

- \W, per trovare un carattere non stringa

```
/\W/g
```

Text Tests **NEW**

RegExr was created by gskinner.com, and is proudly hosted by Media Temple.  
Explore results with the Tools below. Replace & List output custom results. Details lists English.

- \d, per trovare un carattere cifra

```
/\d/g
```

Text Tests **NEW**

RegExr was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \D, per trovare un carattere non cifra

```
/\D/g
```

Text Tests **NEW**

RegExr was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \s, per trovare un carattere spazio

```
/\s/g
```

Text Tests **NEW**

RegExr was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \S, per trovare un carattere non spazio

```
/\S/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \b, per trovare corrispondenze all'inizio o alla fine di una parola

```
/\b(Reg)/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \B, per trovare corrispondenze non all'inizio e non alla fine di una parola

```
/\B(i)/g
```

Text Tests **NEW**

RegExp was created by gskinner.com in 2010, and is proudly hosted by Media Temple.

- \0, trova un carattere nullo
- \n, trova un carattere newline
- \f, trova un carattere form feed
- \r, trova un carattere carriage return
- \xxx, trova il carattere specificato attraverso il numero ottale xxx
- \xdd, trova il carattere specificato attraverso l'esadecimale xdd

- **Altro elemento sono i quantificatori.** I quantificatori sono dei simboli speciali usati nelle espressioni regolari per selezionare una quantità variabile di caratteri nel testo che rispettano la stessa condizione.

**Come usare i quantificatori nella RegEx?** Nelle espressioni regolari i quantificatori devono essere messi subito dopo la condizione da quantificare.

- N+, si verifica che all'interno della stringa esista ALMENO un carattere N
- N\*, si verifica se abbiamo un pattern CON ZERO O PIÙ occorrenze di N
- N?, si verifica se abbiamo un pattern CON ZERO O UNA occorrenza di N
- N{X}, si verifica se è presente un pattern contenente una sequenza formata da X (numero) N.
- N{X, Y}, si verifica se è presente nella stringa una sequenza formata da X (numero) o Y (numero) N
- N{X, }, (virgola presente) si verifica se è presente nella stringa una sequenza di almeno X (numero) N
- N\$, si verifica la presenza di un pattern con N alla fine
- ^n, si verifica la presenza di un pattern con n all'inizio
- ?=n, si verifica la presenza di un pattern seguito da una specifica stringa n
- ?!n, si verifica la presenza di un pattern non seguito da una specifica stringa n.

- **Esempio:** attraverso il seguente esempio ricerchiamo un pattern che contiene sequenze di 3 o 4 cifre. La ricerca viene fatta a livello globale: non ci fermiamo alla prima stringa ma la scorriamo tutta.

```
<script type="text/javascript">
var str="100, 1000 or 10000?";
var patt1=/\d{3,4}/g;
document.write(str.match(patt1));
</script>
```

**Output:** 100,1000,1000

- **Sito per allenarsi:** <https://regexr.com/>

## String

- Costruttore. Non confondiamo il letterale stringa con l'oggetto stringa.  
`s1 = "Hi"; // string literal value`  
`s2 = new String("Hi"); // string object`
- Possibile chiamare le funzioni dell'oggetto stringa sul literal stringa: javascript converte automaticamente il letterale in un oggetto temporaneo. Dopo aver svolto quanto necessario converte l'oggetto stringa in letterale.
- Si hanno due tipi di metodi:
  - o Quelli che restituiscono la stringa modificata
  - o Quelli che restituiscono una versione formattata in HTML della stringa
- **Funzioni membro** presenti dalla diapositiva alla 156. Per gli esempi successivi useremo  
`oggetto_stringa = new String("Ciao Marco, come stai?");`
  - o `charAt(n)`  
restituisce il carattere presente nella posizione con indice n  
`alert(oggetto_stringa.charAt(5)); // M`
  - o `concat(string1, string2, ..., stringN)`  
permette di concatenare due o più stringhe. Restituisce la stringa concatenata  
`oggetto_stringa = oggetto_stringa.concat(" (cit.pcineverdies)");`  
`alert(oggetto_stringa); // Ciao Marco, come stai? (cit.pcineverdies)`
  - o `toLowerCase()` e `toUpperCase()`  
permette di porre un'intera stringa in minuscolo o maiuscolo. Restituiscono la stringa modificata.  
`oggetto_stringa = oggetto_stringa.toLowerCase();`  
`alert(oggetto_stringa); // "ciao marco, come stai?"`  
`oggetto_stringa = oggetto_stringa.toUpperCase();`  
`alert(oggetto_stringa); // "CIAO MARCO, COME STAI?"`
  - o `substr(n1, n2)`  
estrae i caratteri di una stringa, precisamente i caratteri dalla stringa in posizione con indice n1 e gli n2 caratteri successivi (incluso il primo in n1). Restituisce la sottostringa estratta.  
`alert(oggetto_stringa.substr(5,5)); // "Marco"`
  - o `substring(n1, n2)` e `slice(begin, end)`  
estraggono i caratteri di una stringa, precisamente quelli che vanno dalla posizione con indice n1 alla posizione con indice n2-1 (n2 è il primo carattere non considerato). Il secondo argomento è facoltativo. Restituisce la sottostringa estratta.  
`alert(oggetto_stringa.substring(5,10)); // "Marco"`  
`alert(oggetto_stringa.substring(5)); // "Marco, come stai?"`

### **Differenze tra le due funzioni: (da *geeksforgeeks*)**

#### **Common Result**

Both give same results in the given cases.

1. If `start == stop`, both returns an empty string
2. If `stop` is omitted, both extracts characters till the end of the string
3. If any argument is greater than the string's length, the string's length will be used in that case.

#### **substring()**

Separate results of `substring()`

1. If `start > stop`, then function swaps both arguments.
2. If any argument is negative or is NaN, it is treated as 0.

## slice()

Separate results of slice()

1. If `start > stop`, This function will return an empty string. ("")
2. If `start` is negative, It sets char from the end of string, like `substr()`.
3. If `stop` is negative, It sets `stop = string.length - Math.abs(stop)` (original value)

- o `indexOf(string, start)`  
restituisce l'indice associato alla posizione dove si è trovata la prima occorrenza della stringa posta come argomento. Posso porre più di un carattere (si restituirà l'indice della posizione della prima lettera). L'argomento `start` è opzionale: se omissso equivale a 0. Restituisce -1 in assenza di risultati.  

```
alert(oggetto_stringa.indexOf("M")); // 5  
alert(oggetto_stringa.indexOf("M", 6)); //-1
```
- o `lastIndexOf(string, start)`  
stesse funzioni della precedente, ma si restituisce l'ultima occorrenza!  

```
oggetto_stringa = new String("Ciao Marco, come stai???");  
alert(oggetto_stringa.indexOf("?")); // 21  
alert(oggetto_stringa.lastIndexOf("?")); // 23
```
- o `match(regex)`  
tramite regular expression si ricerca del testo all'interno della stringa. Si restituiscono le corrispondenze trovate
- o `replace(regex/substr, newstring)`  
col primo argomento si indica l'area che ci interessa sostituire, con *newstring* poniamo il valore sostitutivo (se presente il valore da sostituire)
- o `search(regex)`  
si ricerca mediante regular expression testo all'interno di una stringa, in caso di corrispondenza si restituisce la posizione della corrispondenza.
- o `split(separator, limit)`  
equivalente della `explode` in PHP. Si divide la stringa sfruttando il separatore indicato. Se il separatore viene omissso sarà restituita tutta la riga. Il secondo argomento, anch'esso opzionale, permette di indicare un numero massimo di divisioni mediante separatore.  

```
array = oggetto_stringa.split(",");  
alert(array[0]); // "Ciao Marco"  
alert(array[0]); // "come stai?"
```
- o `valueOf()`  
restituisce il valore primitivo dell'oggetto stringa.  

```
alert(typeof oggetto_stringa.valueOf()); // "string"
```

## Proprietà globali e funzioni globali negli oggetti built-in di Javascript

- Si individuano funzioni e proprietà valide per tutti gli oggetti Javascript
- **Proprietà globali:**
  - o *Infinity*, valore numerico che rappresenta l'infinito positivo o l'infinito negativo
  - o *NaN*, valore "Not-A-Number"
  - o *Undefined*, non è stato assegnato un valore alla variabile.

```
// si restituisce un valore, se non fossero valide avrei un errore del tipo "Variable undefined"  
alert(Infinity); // Infinity  
alert(NaN); // NaN  
alert(undefined); // undefined
```

- **Funzioni globali:** le seguenti funzioni permettono di codificare o decodificare un URI. Nel caso nostro intendiamo gli URL

- o `encodeURIComponent()`, codifica di un URI
- o `decodeURIComponent()`, decodifica di un URI

<p>Click the button to decode a URI after encoding it.</p>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

```
<script>
function myFunction() {
  var uri = "my test.asp?name=ståle&car=saab";
  var enc = encodeURIComponent(uri);
  var dec = decodeURIComponent(enc);
  var res = "Encoded URI: " + enc + "<br>" + "Decoded URI: " + dec;
  document.getElementById("demo").innerHTML = res;
}
</script>
```

Click the button to decode a URI after encoding it.

Try it

Encoded URI: my%20test.asp?name=st%C3%A5le&car=saab

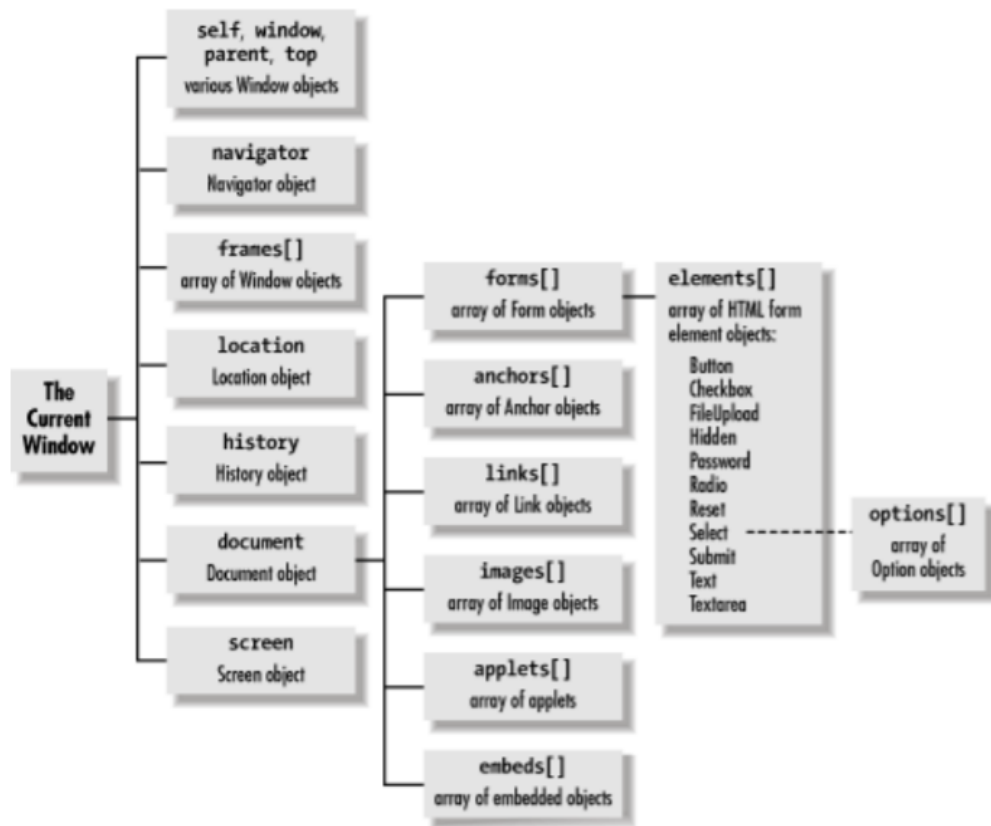
Decoded URI: my test.asp?name=ståle&car=saab

- **Tra le funzioni globali abbiamo anche molte cose già viste:**

- o ~~`Escape()`, codifica di una stringa (DEPRECATA)~~
- o ~~`Unescape()`, decodifica di una stringa codificata (DEPRECATA)~~
- o `eval()`, esecuzione di una stringa come codice javascript
- o `isFinite()` e `isNaN()`, verifica della consistenza di un numero
- o `Number()`, conversione del valore dell'oggetto in un numero
- o `parseFloat()` e `parseInt()`, conversione di una stringa in un intero o un float
- o `String()`, conversione del valore di un oggetto in una stringa

## Oggetti client-side

- Abbiamo già parlato del DOM introducendo l'HTML: il dettaglio interessante è che grazie al DOM potremo gestire, con Javascript, la struttura del documento (modificandola se necessario)
- Cosa succede quando un documento viene caricato in un browser? Quando si carica il documento si crea un certo numero di oggetti, utili per il programmatore.



### Oggetti generati in ogni pagina:

- **window**: rappresenta la viewport, la finestra che stiamo usando per visualizzare il documento.
  - o **history**: rappresenta lo storico degli URL (permette di tornare indietro o avanti nella navigazione)
  - o **location**: rappresenta l'URL attuale (permette di manipolarlo totalmente o parzialmente).
  - o **document**: rappresenta il documento e contiene proprietà legate al suo contenuto (permette di manipolare l'HTML del documento).
- **navigator**: rappresenta il browser, lo strumento utilizzato per navigare su internet.
- **screen**: informazioni relative allo schermo del computer.

### Oggetto window

- Rappresenta la finestra che contiene il nostro documento
- Se il documento contiene frames il browser crea ulteriori oggetti per ognuno di essi.
- **Tutti gli altri oggetti sono figli di uno dei oggetti window, tranne il navigator e screen.**
- Ogni window presenta lo storico delle pagine visitate in quella finestra (mediante l'oggetto history).
- **Javascript si ricorda di quale sia la finestra corrente**: questo permette di risparmiarsi un riferimento esplicito a questa finestra quando parliamo di sotto-oggetti. Noi scriveremo `document.write()` invece di `window.document.write()` e `innerWidth` invece di `window.innerWidth`
- **Proprietà object**:
  - o **history**: oggetto che contiene le pagine visitate nella finestra considerata.

- `frames`  
array di tutti i frames presenti nell'attuale finestra.
- **`document`**  
oggetto relativo al documento aperto nella finestra.
- **`location`**  
restituisce l'oggetto location relativo alla finestra
- **`screen`**  
restituisce l'oggetto screen relativo all'attuale finestra
- **`navigator`**  
restituisce l'oggetto navigator relativo alla finestra
- `parent`  
restituisce la finestra parent dell'attuale finestra
- `opener`  
restituisce un riferimento alla finestra che ha creato l'attuale finestra (se possibile)
- `self`  
restituisce l'oggetto dell'attuale finestra

- **Proprietà scalari:**

- `innerHeight` ed `innerWidth`  
restituiscono, rispettivamente, altezza e larghezza della finestra
- `outerHeight` ed `outerWidth`  
restituiscono, rispettivamente, altezza e larghezza della finestra considerando anche scrollbar e toolbar.
- `length`  
restituisce il numero di frame in una finestra.
- `closed`  
valore booleano che ci indica se la finestra è stata chiusa o no.
- `name`  
restituisce il nome della finestra
- `pageXOffset` e `pageYOffset`  
restituiscono in pixel di quanto è stato spostato il documento (rispettivamente in larghezza e altezza)
- `screenX` e `screenY` (solo Firefox e Chrome)  
restituiscono le coordinate dell'angolo in alto a sinistra del browser rispetto alla finestra del pc.
- `top`  
restituisce la finestra più in alto del browser

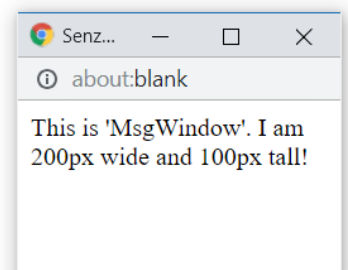
- **Funzioni:**

- `alert(text)`  
già vista
- `confirm(text)`  
già vista

- `prompt(text, value)`  
già vista
- `open(URL, name, specs, replace)`  
per aprire una finestra
  - *URL*: url della pagina da aprire, se non indicato si aprirà una finestra *about:blank*
  - *name*: attributo target o nome della finestra. Sono possibili i seguenti valori
    - *\_blank* - URL is loaded into a new window, or tab. This is default
    - *\_parent* - URL is loaded into the parent frame
    - *\_self* - URL replaces the current page
    - *\_top* - URL replaces any framesets that may be loaded
    - *name* - The name of the window
 (Note: the name does not specify the title of the new window)
  - *specs*: caratteristiche della pagina. Posso avere una lista di caratteristiche, separate da virgole e senza spazi bianchi.
  - *replace*: specifica se l'URL crea una nuova entry nello storico o se sostituisce l'elemento attualmente aperto (in questo caso pongo true, altrimenti false).

Alcuni esempi di proprietà specificabili in <i>specs</i>	
<code>height=pixels</code>	The height of the window. Min. value is 100
<code>left=pixels</code>	The left position of the window. Negative values not allowed
<code>menubar=yes no 1 0</code>	Whether or not to display the menu bar
<code>scrollbars=yes no 1 0</code>	Whether or not to display scroll bars. IE, Firefox & Opera only
<code>status=yes no 1 0</code>	Whether or not to add a status bar
<code>titlebar=yes no 1 0</code>	Whether or not to display the title bar. Ignored unless the calling application is an HTML Application or a trusted dialog box
<code>toolbar=yes no 1 0</code>	Whether or not to display the browser toolbar. IE and Firefox only
<code>top=pixels</code>	The top position of the window. Negative values not allowed
<code>width=pixels</code>	The width of the window. Min. value is 100

```
var myWindow =
window.open("", "MsgWindow", "width=200,height=100");
myWindow.document.write("<p>This is 'MsgWindow'. I am 200px
wide and 100px tall!</p>");
```



- `close()`  
per chiudere la finestra in cui viene eseguito il codice (cosa valida solo con le finestre aperte con la funzione precedente)
- `print()`  
per stampare il contenuto della finestra
- `moveBy(x, y)`  
permette di muovere la finestra prendendo come riferimento la posizione attuale
- `moveTo(x, y)`  
permette di muovere la finestra in una posizione specifica (indipendentemente da dove si trovava)
- `resizeBy(x, y)`  
permette di ridimensionare la finestra prendendo come riferimento le dimensioni attuali



- o `resizeTo(x, y)`  
permette di stabilire le dimensioni della finestra indipendentemente da quelle attuali
  - o `scrollBy(x, y)`  
permette di spostare il contenuto del documento all'interno della finestra prendendo come riferimento la posizione attuale del documento all'interno della finestra
  - o `scrollTo(x, y)`  
permette di spostare il contenuto del documento all'interno della finestra indipendentemente dalla posizione attuale del documento all'interno della finestra.
- `for(proprieta in window)`  
`document.write('<b>' + proprieta + '</b>: ' + window[proprieta] + '<br>');`

```

postMessage: function () { [native code] }
blur: function () { [native code] }
focus: function () { [native code] }
close: function () { [native code] }
frames: [object Window]
self: [object Window]
window: [object Window]
parent: [object Window]
opener: null
top: [object Window]
length: 0
closed: false
location: http://localhost/js.html
document: [object HTMLDocument]
origin: http://localhost
name:
history: [object History]
locationbar: [object BarProp]
menubar: [object BarProp]
personalbar: [object BarProp]
scrollbars: [object BarProp]
statusbar: [object BarProp]
toolbar: [object BarProp]
status:
frameElement: null
navigator: [object Navigator]
customElements: [object CustomElementRegistry]
external: [object External]
screen: [object Screen]
innerWidth: 1524
innerHeight: 714
scrollX: 0
pageXOffset: 0
scrollY: 0
pageYOffset: 0
screenX: 387
screenY: 664
outerWidth: 1538
outerHeight: 832
devicePixelRatio: 1.25
clientInformation: [object Navigator]
screenLeft: 387
screenTop: 664
defaultStatus:
defaultstatus:

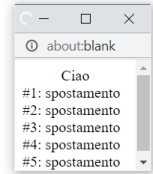
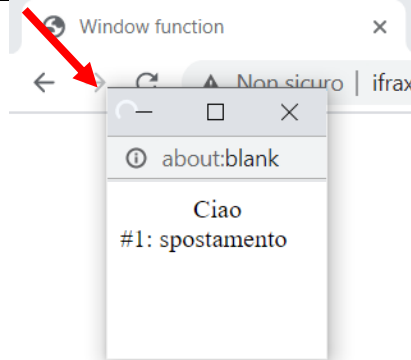
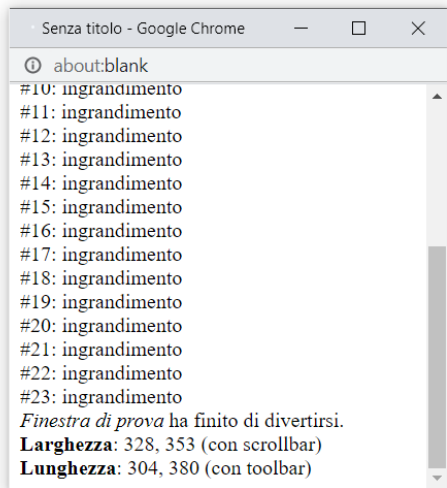
```

```
styleMedia: [object StyleMedia]
onanimationend: null
onanimationiteration: null
onanimationstart: null
onsearch: null
ontransitionend: null
onwebkitanimationend: null
onwebkitanimationiteration: null
onwebkitanimationstart: null
onwebkittransitionend: null
isSecureContext: true
onabort: null
onblur: null
oncancel: null
oncanplay: null
oncanplaythrough: null
onchange: null
onclick: null
onclose: null
oncontextmenu: null
oncuechange: null
ondblclick: null
ondrag: null
ondragend: null
ondragenter: null
ondragleave: null
ondragover: null
ondragstart: null
ondrop: null
ondurationchange: null
onemptied: null
onended: null
onerror: null
onfocus: null
oninput: null
oninvalid: null
onkeydown: null
onkeypress: null
onkeyup: null
onload: null
onloadeddata: null
onloadedmetadata: null
onloadstart: null
onmousedown: null
onmouseenter: null
onmouseleave: null
onmousemove: null
onmouseout: null
onmouseover: null
onmouseup: null
onmousewheel: null
onpause: null
onplay: null
onplaying: null
onprogress: null
onratechange: null
onreset: null
onresize: null
onscroll: null
onseeked: null
onseeking: null
```

```
onselect: null
onstalled: null
onsubmit: null
onsuspend: null
ontimeupdate: null
ontoggle: null
onvolumechange: null
onwaiting: null
onwheel: null
onauxclick: null
ongotpointercapture: null
onlostpointercapture: null
onpointerdown: null
onpointermove: null
onpointerup: null
onpointercancel: null
onpointerover: null
onpointerout: null
onpointerenter: null
onpointerleave: null
onafterprint: null
onbeforeprint: null
onbeforeunload: null
onhashchange: null
onlanguagechange: null
onmessage: null
onmessageerror: null
onoffline: null
ononline: null
onpagehide: null
onpageshow: null
onpopstate: null
onrejectionhandled: null
onstorage: null
onunhandledrejection: null
onunload: null
performance: [object Performance]
stop: function stop() { [native code] }
open: function open() { [native code] }
alert: function alert() { [native code] }
confirm: function confirm() { [native code] }
prompt: function prompt() { [native code] }
print: function print() { [native code] }
requestAnimationFrame: function requestAnimationFrame() { [native code] }
cancelAnimationFrame: function cancelAnimationFrame() { [native code] }
requestIdleCallback: function requestIdleCallback() { [native code] }
cancelIdleCallback: function cancelIdleCallback() { [native code] }
captureEvents: function captureEvents() { [native code] }
releaseEvents: function releaseEvents() { [native code] }
getComputedStyle: function getComputedStyle() { [native code] }
matchMedia: function matchMedia() { [native code] }
moveTo: function moveTo() { [native code] }
moveBy: function moveBy() { [native code] }
resizeTo: function resizeTo() { [native code] }
resizeBy: function resizeBy() { [native code] }
getSelection: function getSelection() { [native code] }
find: function find() { [native code] }
webkitRequestAnimationFrame: function webkitRequestAnimationFrame() {
[native code] }
webkitCancelAnimationFrame: function webkitCancelAnimationFrame() {
[native code] }
```

```
fetch: function fetch() { [native code] }
btoa: function btoa() { [native code] }
atob: function atob() { [native code] }
setTimeout: function setTimeout() { [native code] }
clearTimeout: function clearTimeout() { [native code] }
setInterval: function setInterval() { [native code] }
clearInterval: function clearInterval() { [native code] }
createImageBitmap: function createImageBitmap() { [native code] }
scroll: function scroll() { [native code] }
scrollTo: function scrollTo() { [native code] }
scrollBy: function scrollBy() { [native code] }
onappinstalled: null
onbeforeinstallprompt: null
crypto: [object Crypto]
ondevicemotion: null
ondeviceorientation: null
ondeviceorientationabsolute: null
indexedDB: [object IDBFactory]
webkitStorageInfo: [object DeprecatedStorageInfo]
sessionStorage: [object Storage]
localStorage: [object Storage]
chrome: [object Object]
visualViewport: [object VisualViewport]
speechSynthesis: [object SpeechSynthesis]
webkitRequestFileSystem: function () { [native code] }
webkitResolveLocalFileSystemURL: function () { [native code] }
openDatabase: function () { [native code] }
applicationCache: [object ApplicationCache]
caches: [object CacheStorage]
TEMPORARY: 0
PERSISTENT: 1
addEventListener: function addEventListener() { [native code] }
removeEventListener: function removeEventListener() { [native code] }
dispatchEvent: function dispatchEvent() { [native code] }
```

## Esempio di esercizio (mio)



// Con la funzione open possiamo aprire sia pagine web esistenti sia finestre vuote che riempiremo noi.

// Col primo parametro si può indicare l'URL, col secondo il nome (non si intende il nome solitamente posto nell'el. title)

// Col terzo parametro si pongono le proprietà della finestra. Le più importanti sono la width e la height.

// Ogni secondo compio qualcosa

// Per i primi 5 secondi sposto ogni volta la finestra di 55 px in basso e 55 px a destra

// Successivamente ingrandisco ogni secondo di 10px sia in grandezza che in larghezza la finestra

// L'espansione continua finché non supero 300px sia in grandezza che in larghezza

// A quel punto blocco l'esecuzione della funzione (l'identificativo e' quello restituito dalla setInterval) e faccio scroll in fondo alla window

// Stampo anche un saluto e le dimensioni finali della finestra

// Imposto l'esecuzione ritardata (di 50 secondi) di una funzione che mi chiuderà la finestra e mi restituirà un booleano

// Col booleano si può verificare se la finestra è stata chiusa o meno. Ricordiamoci che si possono chiudere solo finestre aperte col comando open. Cosa interessante è la possibilità di recuperare informazioni su una finestra anche dopo averla chiusa.

```
<script type="text/javascript">
    var nuovaFinestra = open('', 'Finestra di prova', 'width=100px, height=100px');
    nuovaFinestra.document.write('<center>Ciao</center>');

    var i = 0;
    var id1 = setInterval(function() {
        if(i < 5) {
            nuovaFinestra.moveBy(55, 55);
            nuovaFinestra.document.write('#' + parseInt(i+1) + ': spostamento <br>');
        }
        else if(i >= 5) {
            nuovaFinestra.resizeBy(10,10);
            nuovaFinestra.document.write('#' + parseInt(i+1) + ': ingrandimento <br>');
            if(nuovaFinestra.innerHeight > 300 && nuovaFinestra.innerWidth > 300) {
                clearInterval(id1);
                nuovaFinestra.document.write('<i>' + nuovaFinestra.name + '</i> ha
finito di divertirsi.<br>');
                nuovaFinestra.document.write('<b>Larghezza</b>: ' +
nuovaFinestra.innerWidth + ', ' + nuovaFinestra.outerWidth + ' (con scrollbar)<br>');
                nuovaFinestra.document.write('<b>Lunghezza</b>: ' +
nuovaFinestra.innerHeight + ', ' + nuovaFinestra.outerHeight + ' (con toolbar)');
                nuovaFinestra.scrollTo(0, 350);
            }
        }
        i++;
    }, 1000);

    setTimeout(function() { nuovaFinestra.close(); alert(nuovaFinestra.closed); },
50000);
</script>
```

**Oggetto navigator** (non quello di cui si parlava fino a qualche mese fa, cit.)

- Informazioni relative al browser di tipo read-only.
- **Proprietà interessanti** (anteprime nell'output del for):
  - o `appName`  
restituisce il nome in codice del browser
  - o `appVersion`  
Restituisce la informazioni relative alla versione usata del browser.
  - o `cookieEnabled`  
booleano che determina se i cookie sono abilitati sul browser.
  - o `platform`  
Restituisce la piattaforma per cui il browser è realizzato.
  - o `userAgent`  
Restituisce l'header user-agent inviato dal browser al server.

```
for (var proprietyName in navigator)
    document.write("<strong>" + proprietyName + ":</strong>" +
        navigator[proprietyName] + "<br>");
```

**vendorSub:**

**productSub:**20030107

**vendor:**Google Inc.

**maxTouchPoints:**0

**userActivation:**[object UserActivation]

**doNotTrack:**null

**geolocation:**[object Geolocation]

**connection:**[object NetworkInformation]

**plugins:**[object PluginArray]

**mimeTypeTypes:**[object MimeTypeArray]

**webkitTemporaryStorage:**[object DeprecatedStorageQuota]

**webkitPersistentStorage:**[object DeprecatedStorageQuota]

**hardwareConcurrency:**8

**cookieEnabled:**true

**appName:**Mozilla

**appVersion:**5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML,

like Gecko) Chrome/86.0.4240.111 Safari/537.36

**platform:**Win32

**product:**Gecko

**userAgent:**Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36

(KHTML, like Gecko) Chrome/86.0.4240.111 Safari/537.36

**language:**it-IT

**languages:**it-IT,it,en-US,en

**onLine:**true

**getBattery:**function getBattery() { [native code] }

**getGamepads:**function getGamepads() { [native code] }

**javaEnabled:**function javaEnabled() { [native code] }

**sendBeacon:**function sendBeacon() { [native code] }

**vibrate:**function vibrate() { [native code] }

**xr:**[object XRSystem]

**mediaCapabilities:**[object MediaCapabilities]

**permissions:**[object Permissions]

**locks:**[object LockManager]

```

wakeLock:[object WakeLock]
usb:[object USB]
mediaSession:[object MediaSession]
clipboard:[object Clipboard]
credentials:[object CredentialsContainer]
keyboard:[object Keyboard]
mediaDevices:[object MediaDevices]
storage:[object StorageManager]
serviceWorker:[object ServiceWorkerContainer]
deviceMemory:8
presentation:[object Presentation]
bluetooth:[object Bluetooth]
registerProtocolHandler:function registerProtocolHandler() { [native code]
}
unregisterProtocolHandler:function unregisterProtocolHandler() { [native
code] }
getUserMedia:function getUserMedia() { [native code] }
requestMIDIAccess:function requestMIDIAccess() { [native code] }
requestMediaKeySystemAccess:function requestMediaKeySystemAccess() {
[native code] }
webkitGetUserMedia:function webkitGetUserMedia() { [native code] }
getInstalledRelatedApps:function getInstalledRelatedApps() { [native code]
}
clearAppBadge:function clearAppBadge() { [native code] }
setAppBadge:function setAppBadge() { [native code] }

```

#### Oggetto screen

- Oggetto contenente informazioni read-only relativamente allo schermo del nostro computer.
  - ```
for(proprieta in screen)
    document.write('<b>' + proprieta + '</b>: ' + screen[proprieta] + '<br>');
```
- availWidth:** 1536 *// Larghezza dello schermo (esclusa la Windows taskbar)*  
**availHeight:** 824 *// Altezza dello schermo (esclusa la Windows taskbar)*  
**width:** 1536 *// Larghezza totale dello schermo*  
**height:** 864 *// Altezza totale dello schermo*  
**colorDepth:** 24 *// Numero di bit utilizzati mostrare un colore<sup>1</sup>*  
**pixelDepth:** 24 *// Uguale a colorDepth nei computer moderni*  
**availLeft:** 0  
**availTop:** 0  
**orientation:** [object ScreenOrientation]

#### Oggetto history

- Oggetto contenente gli URL visitati dall'utente.
- Importante perché ci permette di gestire la navigazione: con la proprietà `length` possiamo recuperare la lunghezza del nostro storico.  
`alert(history.length) //6`
- L'oggetto non contiene elementi che riflettono url reali: quindi non possiamo stampare o recuperare gli URL ma possiamo scegliere quale visitare sfruttando **le funzioni** a nostra disposizione:
  - o `back()`  
visito la pagina precedente.

<sup>1</sup> All modern computers use 24 bit or 32 bit hardware for color resolution:

24 bits = 16,777,216 different "True Colors"

32 bits = 4,294,967,296 different "Deep Colors"

Older computers used 16 bits: 65,536 different "High Colors" resolution.

Very old computers, and old cell phones used 8 bits: 256 different "VGA colors".

- o `forward()`  
visito la pagina successiva (se siamo tornati indietro).
- o `go(offset)`  
visito la pagina che si trova indietro o avanti di `|offset|` posizioni.  
`offset` è un numero che può essere positivo o negativo.
- o `go(substring)`  
Individuo l'indirizzo più recente che contiene la stringa specificata.  
`history.go('w3schools');`

**Esempio:**

```
<div>
  Go to the page:<br>
  <button onclick="history.back();">Back</button>
  <button onclick="history.go(-1);">Back (alternativa)</button>
  <button onclick="history.go(0);">Current</button>
  <button onclick="history.go(+1);">Forward</button>
  <button onclick="history.forward();">Forward (alternativa)</button>
</div>
```

**Queste funzioni non restituiscono alcun valore ma reindirizzano verso la pagina indicata.**

- `for(proprieta in history)`  
    `document.write('<b>' + proprieta + '</b>: ' + history[proprieta] + '<br>');`

**length:** 3

**scrollRestoration:** auto

**state:** null

**go:** function go() { [native code] }

**back:** function back() { [native code] }

**forward:** function forward() { [native code] }

**pushState:** function pushState() { [native code] }

**replaceState:** function replaceState() { [native code] }

**Oggetto location**

- Contiene informazioni sull'URL attuale.
- Attraverso le proprietà è possibile manipolare l'URL (basta modificare le proprietà dell'oggetto location).
- **Proprietà disponibili:**
  - o `href`, contiene in una stringa il valore dell'intero URL. Si può scrivere  
`location.href = 'https://google.it';`  
l'utente verrà reindirizzato sulla home di Google.
  - o `host`, contiene l'host name e la porta dell'attuale url
  - o `port`, contiene il numero della porta dell'attuale url
  - o `hostname`, restituisce la porzione hostname dell'url.
  - o `pathname`, restituisce la porzione pathname dell'url
  - o `protocol`, restituisce la parte di protocollo dell'url
  - o `hash`, restituisce il valore che segue il cancelletto.  
`location.hash = 'aggiunta';`
  - o `search`, restituisce il contenuto dopo il punto interrogativo (importante quando parleremo di PHP)  
`location.search = '?ciao=prova';`
- **Funzioni disponibili:**
  - o `reload()`  
si ricarica il contenuto della pagina [Mi pare funzioni solo su Chrome]

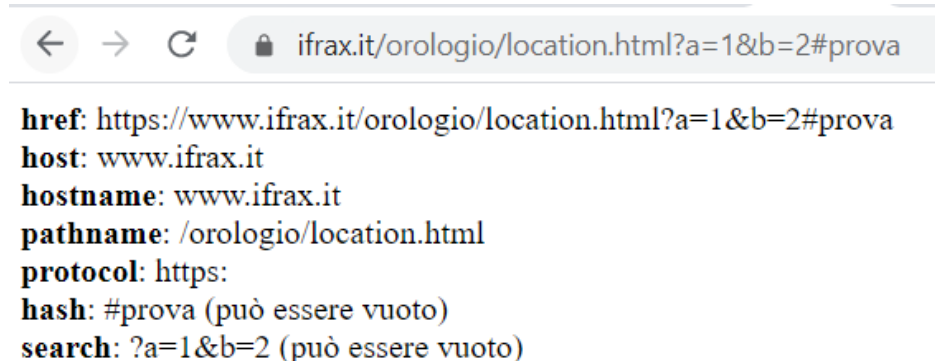


- o `assign(newurl)`  
si carica un nuovo documento
- o `replace(newurl)`  
si sostituisce l'attuale documento con uno nuovo.

**Differenza tra `assign` e `replace`:** la prima funzione aggiunge il nuovo documento allo storico, la seconda rimuove l'URL dell'attuale documento dallo storico. Questo significa che non è recuperabile né dal tasto del browser, né dall'oggetto `history`.

- **Esempio:**

```
<!DOCTYPE HTML>
<html lang="it">
  <head>
    <title>Location.html</title>
  </head>
  <body>
    <script type="text/javascript">
      document.writeln('<b>href</b>: '+location.href + '<br>');
      document.writeln('<b>host</b>: '+location.host+ '<br>');
      document.writeln('<b>hostname</b>: '+location.hostname+ '<br>');
      document.writeln('<b>pathname</b>: '+location.pathname+ '<br>');
      document.writeln('<b>protocol</b>: '+location.protocol+ '<br>');
      document.writeln('<b>hash</b>: '+location.hash + ' (può essere
vuoto)<br>');
      document.writeln('<b>search</b>: '+location.search + ' (può essere
vuoto)<br>');
    </script>
  </body>
</html>
```



## Oggetto document

- Ogni documento HTML caricato in una finestra del browser sarà associato a un oggetto document. Questo oggetto permette di accedere a tutti gli elementi HTML del documento.
- **L'oggetto document è parte dell'oggetto window.**
- Questo oggetto contiene una serie di array di oggetti:
  - o forms, a sua volta contiene l'array *elements* dei seguenti oggetti:
    - button
    - checkbox
    - fileupload
    - hidden
    - password
    - radio
    - reset
    - select, con al suo interno un array di oggetti options.
    - submit
    - text
    - textarea
  - o anchors
  - o links
  - o images
  - o applets
  - o embeds
- L'ordine di tutti questi elementi nell'albero dipende dall'ordine di caricamento degli stessi.
- **Proprietà presenti** alla diapositiva 205: si osservi in particolare la...
  - o `activeElement`  
riferimento all'elemento su cui si ha attualmente *focus*.
  - o `defaultView`  
riferimento all'oggetto window. <sup>2</sup>
  - o `cookie`  
restituisce una lista separate da punti e virgola dei cookie salvati in quel documento. Possiamo anche inizializzare un set di cookies.
  - o `body`  
restituisce l'elemento body dell'attuale documento (un oggetto apparentemente superfluo, molte proprietà sono deprecate in HTML5)
  - o `head`  
restituisce l'elemento head del documento (un oggetto).
  - o `lastModified`  
viene restituita la data di ultima modifica del documento.  
**Esempio:** 11/03/2020 10:34:31
  - o `domain`  
restituisce il nome del dominio del server che ospita il documento.  
**Esempio:** google
  - o `dir`  
restituisce il percorso per arrivare al documento che stiamo leggendo (nel percorso non si include il

<sup>2</sup> Per i fan dei fan delle funzioni ricorsive:

```
alert(document.defaultView.document.defaultView.innerHeight);
```

nome del file con la sua estensione)

- `location`  
restituisce l'URI del documento.  
**Esempio:** <http://localhost/js.html>
- `readyState`  
restituisce lo stato del documento. Viene restituito una delle seguenti stringhe:
  - *uninitialized* - Has not started loading yet
  - *loading* - Is loading
  - *loaded* - Has been loaded
  - *interactive* - Has loaded enough and the user can interact with it
  - *complete* - Fully loaded
- `style`, proprietà da utilizzare per modificare lo style dell'elemento. Le proprietà utilizzabili sono le stesse viste in CSS. Tuttavia dobbiamo tener conto che dobbiamo eliminare i trattini e i caratteri immediatamente successivi devono essere posti maiuscolo.  
`background-color -> backgroundColor`

```
element.style.backgroundColor = 'yellow'
```

- **Funzioni:**

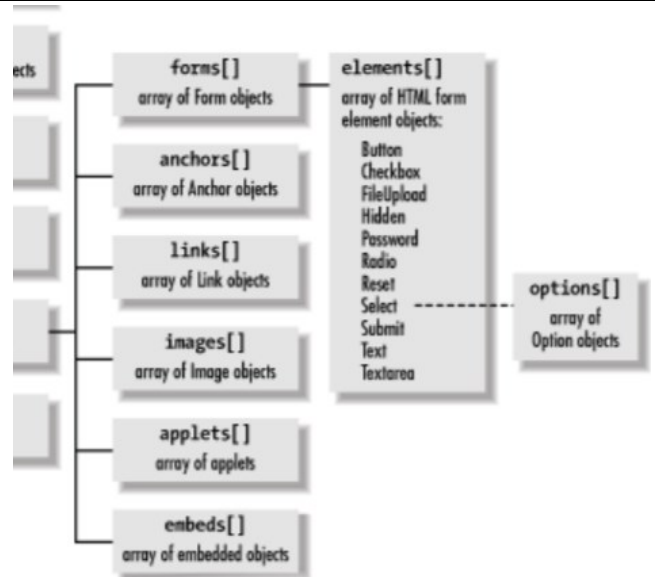
- `write()` e `writeln()`  
funzioni già viste, permettono di scrivere espressioni HTML o codice javascript in un documento. La seconda aggiunge una newline per ogni statement inserito.
- Ho omesso due funzioni presenti alla diapositiva 206 considerando il loro utilizzo in approcci mal digeriti dal docente di laboratorio Tesconi.

**Individuare oggetti con un certo nome**

- **Strategia in DOM 0:**

- Immaginiamoci di avere un array che si ramifica su più fronti: quanto presente nella foto a destra consiste negli array presenti nell'oggetto `document`.
- Ricordandoci che Javascript è case-sensitive.
- Adottiamo una notazione ereditata dagli array in cui si tiene conto del percorso da svolgere per arrivare a un certo elemento.
- **Attenzione:** non è possibile raggiungere i `paragraph`.

```
document.forms[i].elements[i].value
```



- L'approccio appena visto è estremamente inflessibile: per arrivare a un elemento dobbiamo conoscere tutto il percorso.
- **Esempio:**

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>A Simple Document</title>
  </head>
  <body>
    <p><a name="top" id="top">This is the top of the page</a></p>
    <hr>
    <form method="post" id="myform" action="mailto:nobody@dev.null">
```

```

    <p>
    Enter your name: <input type="text" name="me" size="70">
    <input type="Submit" id="prova2" value="OK">
    <input type="Reset" value="Oops">
    </p>
  </form>
  <hr>
  <p>Click here to go to the <a href="#top">top</a> of the page</p>
</body>
</html>

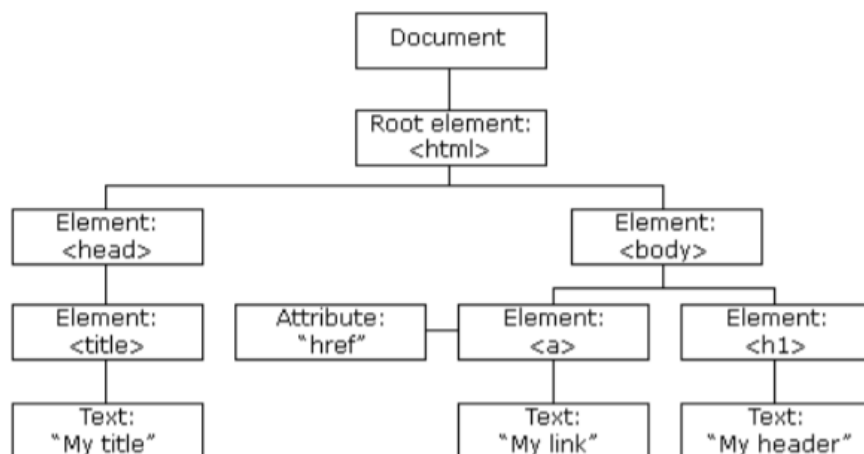
```

Supponiamo di voler raggiungere le seguenti aree di codice

- `<title>A Simple Document</title>`  
raggiungibile con `document.title`
- `<a name="top">This is the top of the page</a>`  
valori raggiungibili con `document.anchors[0].text` e `document.anchors[0].name`
- `<form method="post" action="mailto:nobody@dev.null">`  
Valori raggiungibili con `document.forms[0].method` e `document.forms[0].action`
- `<input type="Submit" value="OK">`  
Valori raggiungibili con `document.forms[0].elements[1].value` e `document.forms[0].elements[1].type`

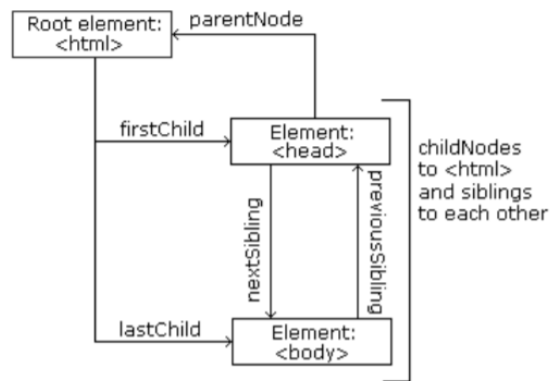
#### - Versione attuale di DOM 3 (ci interessa in particolare lo standard DOM HTML):

- Ricordiamo la formazione dell'albero DOM al momento del caricamento della pagina: questo albero permetterà di muoverci in modo agile da un elemento a un altro.
- Si crea una struttura ad oggetti (un albero) che rappresenta il documento e dove ogni cosa consiste in un nodo di un albero. Possiamo modificare la struttura del documento alterando la gerarchia presente nell'albero DOM.
  - L'intero documento è un nodo *document*
  - Ogni elemento HTML è un nodo *element*
  - **Errore comune:** un tipico errore nel comprendere la processazione del DOM è aspettarsi che un elemento nodo contenga del testo. Ciò non avviene: il testo di un elemento è contenuto in un *text node*. Precisamente, se abbiamo il seguente elemento  
`<title>DOM Tutorial</title>`  
 avremo un text node con valore *DOM Tutorial*. Questo non è il valore dell'elemento `title`!
  - Ogni attributo HTML è un nodo *attribute*
  - I commenti sono nodi *comment*



- Presente, all'interno dei vari oggetti, un oggetto text (tutti, incluso dove apparentemente non sembrerebbe necessario). La cosa sarà chiara più avanti.
- L'HTML DOM è un W3C standard ed è indipendente dai linguaggi.

- La cosa interessante è che avendo un albero possiamo navigare dal padre al figlio, da un figlio a un fratello, da un figlio al padre:



- Presenti dalla diapositiva 234 una lista di attributi che possiamo usare. Si premette che x consiste nel nome di un qualunque oggetto nodo presente nell'albero:
  - `x.innerHTML` (text value di x, sconsigliato)
    - Sconsigliato poiché il contenuto passato per questo attributo viene analizzato dal parser dell'HTML. Questo significa che se poniamo codici Javascript questi saranno eseguiti! La cosa è caldamente sconsigliata per evitare possibili attacchi al visitatore.
    - **Conclusion:** non deve essere usato se vogliamo inserire solo testo.
  - `x.childNodes` ("array" nodi figli di x)
  - `x.parentNode` (riferimento al nodo padre di x)
  - `x.firstChild` (riferimento primo figlio di x)
  - `x.lastChild` (riferimento all'ultimo figlio di x)
  - `x.namespaceURI` (il namespace URI del nodo x)
  - `x.nodeName` (nome del nodo x: p, a, span...)
  - `x.nodeValue` (valore del nodo x)
  - `x.nodeType` (tipo del nodo x)
  - `x.nextSibling` (riferimento al fratello successivo di x)
  - `x.previousSibling` (riferimento al fratello precedente di x)
  - `x.textContent` (contenuto testo di x e dei suoi discendenti)
    - Da usare al posto di innerHTML quando si vuole semplicemente introdurre testo.
- Funzioni messe a disposizione da DOM 3:
  - `x.getElementById(id)`  
per ottenere un elemento avente uno specifico id.
  - `x.getElementsByTagName(name)`  
per ottenere tutti gli elementi che hanno uno specifico tag name. Si ottiene un "array".
  - `x.appendChild(node)`  
per aggiungere un nodo figlio alla fine di una lista di figli di x
  - `newx = x.cloneNode(deep)`  
si clona x per creare un nuovo oggetto. Con deep (booleano) si indica se copiare solo

l'elemento x o tutto l'albero (anche i discendenti di x)

- `x.getAttribute(string_name)`  
per ottenere il valore dell'attributo di x avente nome `string_name`.
- `x.setAttribute(string_name, string_value)`  
per assegnare il valore `string_value` all'attributo `string_name` dell'elemento x.

○ **Ritorniamo sulla modifica:**

- Il metodo più semplice per modificare il contenuto di un elemento è utilizzare `innerHTML`.  
*Ribadiamo che l'uso di questa proprietà è estremamente pericoloso.*
- Due soluzioni alternative in cui teniamo conto che il primo figlio di un qualunque nodo è il nodo contenente il testo.
  - Usare le proprietà `childNodes` e `nodeValue`

```
<body>
<p id="intro">Hello World!</p>
...
function change() {
    txt=document.getElementById("intro").childNodes[0].nodeValue;
    document.getElementById("intro").childNodes[0].nodeValue="hi";
    document.write("<p>Text from the paragraph: " + txt + "<\p>");
}
```

- Usare le proprietà `firstChild` e `nodeValue`

```
function change() {
    txt=document.getElementById("intro").firstChild.nodeValue;
    document.getElementById("intro").firstChild.nodeValue="hi";
    document.write("<p>Text from the paragraph: " + txt + "<\p>");
}
```

○ **Conclusione sullo scorrere gli elementi: possiamo farlo**

- Utilizzando la `getElementById()`

```
var text = document.getElementById("mypar1").firstChild.nodeValue;
window.alert("The paragraph has the text '" + text + "'");
var body = document.getElementById("mybody");
var textNode = body.childNodes[0].firstChild;

//Explorer (old versions)
if (!textNode)
    window.alert("Mozilla Firefox");
else
    window.alert("Microsoft Explorer");
text = textNode ? textNode.nodeValue :
body.childNodes[1].firstChild.nodeValue; //Mozilla
window.alert("The paragraph has the text '" + text + "'");
```

- Utilizzando la `getElementsByTagName()`

```
var elems = document.getElementsByTagName("p");
var text = elems[0].firstChild.nodeValue;
window.alert("The paragraph has the text '" + text + "'");
```

- Navigare l'albero DOM  
(esempi alle diapositive 250 e 251, rispettivamente versione iterativa e ricorsiva)

Abbiamo maggiore flessibilità rispetto al DOM 0: nella versione iniziale era necessario conoscere il percorso per arrivare all'elemento.

### Lista completa di proprietà e funzioni dell'oggetto *document*

```
for(proprieta in document)
    document.write('<b>'+proprieta + '</b>: ' + document[proprieta] + '<br>');

location: http://localhost/js.html?ciao=prova
implementation: [object DOMImplementation]
URL: http://localhost/js.html?ciao=prova
documentURI: http://localhost/js.html?ciao=prova
origin: http://localhost
compatMode: CSS1Compat
characterSet: windows-1252
charset: windows-1252
inputEncoding: windows-1252
contentType: text/html
doctype: [object DocumentType]
documentElement: [object HTMLHtmlElement]
xmlEncoding: null
xmlVersion: null
xmlStandalone: false
domain: localhost
referrer: http://localhost/js.html?ciao=prova
cookie:
lastModified: 11/02/2020 17:34:19
readyState: loading
title: prova
dir:
body: [object HTMLBodyElement]
head: [object HTMLHeadElement]
images: [object HTMLCollection]
embeds: [object HTMLCollection]
plugins: [object HTMLCollection]
links: [object HTMLCollection]
forms: [object HTMLCollection]
scripts: [object HTMLCollection]
currentScript: [object HTMLScriptElement]
defaultView: [object Window]
designMode: off
onreadystatechange: null
anchors: [object HTMLCollection]
applets: [object HTMLCollection]
fgColor:
linkColor:
vlinkColor:
alinkColor:
bgColor:
all: [object HTMLAllCollection]
scrollingElement: [object HTMLHtmlElement]
onpointerlockchange: null
onpointerlockerror: null
hidden: false
visibilityState: visible
webkitVisibilityState: visible
webkitHidden: false
onbeforecopy: null
onbeforecut: null
onbeforepaste: null
oncopy: null
oncut: null
onpaste: null
onsearch: null
onselectionchange: null
```

**onselectstart:** null  
**onvisibilitychange:** null  
**fonts:** [object FontFaceSet]  
**activeElement:** [object HTMLBodyElement]  
**styleSheets:** [object StyleSheetList]  
**pointerLockElement:** null  
**onabort:** null  
**onblur:** null  
**oncancel:** null  
**oncanplay:** null  
**oncanplaythrough:** null  
**onchange:** null  
**onclick:** null  
**onclose:** null  
**oncontextmenu:** null  
**oncuechange:** null  
**ondblclick:** null  
**ondrag:** null  
**ondragend:** null  
**ondragenter:** null  
**ondragleave:** null  
**ondragover:** null  
**ondragstart:** null  
**ondrop:** null  
**ondurationchange:** null  
**onemptied:** null  
**onended:** null  
**onerror:** null  
**onfocus:** null  
**oninput:** null  
**oninvalid:** null  
**onkeydown:** null  
**onkeypress:** null  
**onkeyup:** null  
**onload:** null  
**onloadeddata:** null  
**onloadedmetadata:** null  
**onloadstart:** null  
**onmousedown:** null  
**onmouseenter:** null  
**onmouseleave:** null  
**onmousemove:** null  
**onmouseout:** null  
**onmouseover:** null  
**onmouseup:** null  
**onmousewheel:** null  
**onpause:** null  
**onplay:** null  
**onplaying:** null  
**onprogress:** null  
**onratechange:** null  
**onreset:** null  
**onresize:** null  
**onscroll:** null  
**onseeked:** null  
**onseeking:** null  
**onselect:** null  
**onstalled:** null  
**onsubmit:** null  
**onsuspend:** null  
**ontimeupdate:** null



```
ontoggle: null
onvolumechange: null
onwaiting: null
onwheel: null
onauxclick: null
ongotpointercapture: null
onlostpointercapture: null
onpointerdown: null
onpointermove: null
onpointerup: null
onpointercancel: null
onpointerover: null
onpointerout: null
onpointerenter: null
onpointerleave: null
children: [object HTMLCollection]
firstElementChild: [object HTMLHtmlElement]
lastElementChild: [object HTMLHtmlElement]
childElementCount: 1
webkitIsFullScreen: false
webkitCurrentFullScreenElement: null
webkitFullscreenEnabled: true
webkitFullscreenElement: null
onwebkitfullscreenchange: null
onwebkitfullscreenerror: null
rootElement: null
getElementsByTagName: function getElementsByTagName() { [native code] }
getElementsByTagNameNS: function getElementsByTagNameNS() { [native code] }
getElementsByClassName: function getElementsByClassName() { [native code] }
createDocumentFragment: function createDocumentFragment() { [native code] }
createTextNode: function createTextNode() { [native code] }
createCDATASection: function createCDATASection() { [native code] }
createComment: function createComment() { [native code] }
createProcessingInstruction: function createProcessingInstruction() { [native code] }
importNode: function importNode() { [native code] }
adoptNode: function adoptNode() { [native code] }
createAttribute: function createAttribute() { [native code] }
createAttributeNS: function createAttributeNS() { [native code] }
createEvent: function createEvent() { [native code] }
createRange: function createRange() { [native code] }
createNodeIterator: function createNodeIterator() { [native code] }
createTreeWalker: function createTreeWalker() { [native code] }
getElementsByName: function getElementsByName() { [native code] }
open: function open() { [native code] }
close: function close() { [native code] }
write: function write() { [native code] }
writeln: function writeln() { [native code] }
hasFocus: function hasFocus() { [native code] }
execCommand: function execCommand() { [native code] }
queryCommandEnabled: function queryCommandEnabled() { [native code] }
queryCommandIndeterm: function queryCommandIndeterm() { [native code] }
queryCommandState: function queryCommandState() { [native code] }
queryCommandSupported: function queryCommandSupported() { [native code] }
queryCommandValue: function queryCommandValue() { [native code] }
clear: function clear() { [native code] }
captureEvents: function captureEvents() { [native code] }
releaseEvents: function releaseEvents() { [native code] }
```

```

exitPointerLock: function exitPointerLock() { [native code] }
createElement: function createElement() { [native code] }
createElementNS: function createElementNS() { [native code] }
caretRangeFromPoint: function caretRangeFromPoint() { [native code] }
getSelection: function getSelection() { [native code] }
elementFromPoint: function elementFromPoint() { [native code] }
elementsFromPoint: function elementsFromPoint() { [native code] }
getElementById: function getElementById() { [native code] }
prepend: function prepend() { [native code] }
append: function append() { [native code] }
querySelector: function querySelector() { [native code] }
querySelectorAll: function querySelectorAll() { [native code] }
webkitCancelFullScreen: function webkitCancelFullScreen() { [native code] }
}
webkitExitFullscreen: function webkitExitFullscreen() { [native code] }
createExpression: function createExpression() { [native code] }
createNSResolver: function createNSResolver() { [native code] }
evaluate: function evaluate() { [native code] }
wasDiscarded: false
onfreeze: null
onresume: null
registerElement: function () { [native code] }
pictureInPictureElement: null
pictureInPictureEnabled: false
exitPictureInPicture: function () { [native code] }
ELEMENT_NODE: 1
ATTRIBUTE_NODE: 2
TEXT_NODE: 3
CDATA_SECTION_NODE: 4
ENTITY_REFERENCE_NODE: 5
ENTITY_NODE: 6
PROCESSING_INSTRUCTION_NODE: 7
COMMENT_NODE: 8
DOCUMENT_NODE: 9
DOCUMENT_TYPE_NODE: 10
DOCUMENT_FRAGMENT_NODE: 11
NOTATION_NODE: 12
DOCUMENT_POSITION_DISCONNECTED: 1
DOCUMENT_POSITION_PRECEDING: 2
DOCUMENT_POSITION_FOLLOWING: 4
DOCUMENT_POSITION_CONTAINS: 8
DOCUMENT_POSITION_CONTAINED_BY: 16
DOCUMENT_POSITION_IMPLEMENTATION_SPECIFIC: 32
nodeType: 9
nodeName: #document
baseURI: http://localhost/js.html?ciao=prova
isConnected: true
ownerDocument: null
parentNode: null
parentElement: null
childNodes: [object NodeList]
firstChild: [object DocumentType]
lastChild: [object HTMLHtmlElement]
previousSibling: null
nextSibling: null
nodeValue: null
textContent: null
hasChildNodes: function hasChildNodes() { [native code] }
getRootNode: function getRootNode() { [native code] }
normalize: function normalize() { [native code] }
cloneNode: function cloneNode() { [native code] }

```

```
isEqualNode: function isEqualNode() { [native code] }  
isSameNode: function isSameNode() { [native code] }  
compareDocumentPosition: function compareDocumentPosition() { [native  
code] }  
contains: function contains() { [native code] }  
lookupPrefix: function lookupPrefix() { [native code] }  
lookupNamespaceURI: function lookupNamespaceURI() { [native code] }  
isDefaultNamespace: function isDefaultNamespace() { [native code] }  
insertBefore: function insertBefore() { [native code] }  
appendChild: function appendChild() { [native code] }  
replaceChild: function replaceChild() { [native code] }  
removeChild: function removeChild() { [native code] }  
addEventListener: function addEventListener() { [native code] }  
removeEventListener: function removeEventListener() { [native code] }  
dispatchEvent: function dispatchEvent() { [native code] }
```

## Events

- L'utente, che interagisce con la pagina HTML, può generare degli eventi.
- Questi eventi vengono catturati da Javascript, che lancia degli handler (cioè dei gestori degli eventi)
- La cattura è gestita direttamente da Javascript: noi dobbiamo solo scrivere gli handler e associarli agli eventi.

### Esempio di evento

- Un esempio di evento è il *mouse-down event*.
- Premere il bottone del mouse genera un oggetto event che contiene
  - o Il tipo di evento (MouseDown)
  - o La posizione del cursore (x, y)
  - o Un numero che rappresenta il bottone del mouse premuto
  - o Un campo che contiene i tasti che possono essere usati come modificatori.
- Le proprietà contenute nell'oggetto, ovviamente, variano da evento a evento.

### Associare handler ad eventi

- Per associare codice ad eventi abbiamo a disposizione diversi approcci.
- **Primo approccio:** attraverso gli attributi HTML.
  - o L'evento è determinato dal nome dell'attributo: ogni eventName inizia con on (ripensare agli attributi globali visti all'inizio)
  - o Come valore dell'attributo poniamo una o più funzioni (se poniamo più funzioni le separiamo mediante punto e virgola)

```
<div style="position:absolute; left:300px; top:200px;" id="but">
  <button onmouseover="moveMouse()">Click</button>
</div>
```

```
<script type="text/javascript">
  var but = document.getElementById('but');
  var diff = [150, 0, -150, 0];
  var i = 0;
  function moveMouse() {
    but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
    but.style.left=(parseInt(but.style.left)+ diff[(i++)%4]) + "px";
  }
</script>
```

- **Secondo approccio:** utilizzare la proprietà dell'oggetto che corrisponde all'evento. Si pone come valore il nome della funzione (senza parentesi, contrariamente al primo approccio)

```
<div style="position:absolute; left:300px; top:200px;" id="but">
  <button >Click</button>
</div>
<script type="text/javascript" src="./myscript11.js"> </script>
```

```
// myscript11.js
but.onmouseover = moveMouse; //Event handlers are function references
function moveMouse() {
  but.style.top= (parseInt(but.style.top)+ diff[(i+3)%4]) + "px";
  but.style.left=(parseInt(but.style.left)+ diff[(i++)%4]) + "px";
}
```

- **Terzo approccio:** specifichiamo l'handler dell'evento attraverso un oggetto Function e assegniamo l'evento a una serie di elementi attraverso un ciclo for.

Per vedere questo approccio leggere il **Laboratorio 4 del prof.Tesconi**.

- **Quarto approccio:** utilizzo della funzione `addEventListener(type, listener, useCapture)`. Abbiamo i seguenti parametri:
  - o `type`, stringa che indica l'evento da captare
  - o `listener`, oggetto che riceve la notifica quando l'evento specificato si verifica
  - o `useCapture`, booleano. Spiegato più avanti nella sezione relativa all'ordine degli eventi.

```
// Function to change the content of t2
function modifyText() {
    var t2 = document.getElementById("t2");
    if (t2.firstChild.nodeValue == "three") {
        t2.firstChild.nodeValue = "two";
    }
    else {
        t2.firstChild.nodeValue = "three";
    }
}
// add event listener to table
var el = document.getElementById("outside");
el.addEventListener("click", modifyText, false);
```

- **Osservazioni sugli eventi:**
  - o L'evento `onBlur` si applica solo ad oggetti window e a tutti gli elementi di un form.
  - o L'evento `onChange` si applica solo ai campi input, textarea e select di un form.
  - o L'evento `onMouseOut` si applica a elementi links e areas.
  - o L'evento `onResize` è applicabile a document, frame, e window.

### Ordine degli eventi

- Supponiamo di avere un elemento `element2` contenuto all'interno di un altro elemento `element1`.
- Entrambi gli elementi sono associati a degli eventi. Mi chiedo: in che ordine considero gli eventi?
- Esistono due modelli:
  - o **Capturing** (Netscape), dove eseguo prima gli eventi di `element1` e poi quelli di `element2`
  - o **Bubbling** (Microsoft), eseguo prima gli eventi associati ad `element2`
- W3C, che ha definito lo standard, ha intrapreso una via di mezzo.
  - o Gli eventi vengono catturati finché non si raggiunge l'elemento target.
  - o Quando l'elemento è stato trovato si ha "un rimbalzo" (bubbling up)
- Grazie all'ultimo argomento della funzione `addEventListener()` il programmatore può decidere se adottare un modello o un altro.

```
element1.addEventListener('click', doSomething2, true)
element2.addEventListener('click', doSomething, false)
```

If the user clicks on `element2` the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of `element2` has a `onclick` event handler for the capturing phase.
2. The event finds one on `element1`. `doSomething2()` is executed.
3. The event travels down to the target itself, no more event handlers for the capturing phase are found. The event moves to its bubbling phase and executes `doSomething()`, which is registered to `element2` for the bubbling phase.
4. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase. This is not the case, so nothing happens.

```
element1.addEventListener('click',doSomething2,false)
element2.addEventListener('click',doSomething,false)
```

Now if the user clicks on element2 the following happens:

1. The `click` event starts in the capturing phase. The event looks if any ancestor element of element2 has a `onclick` event handler for the capturing phase and doesn't find any.
2. The event travels down to the target itself. The event moves to its bubbling phase and executes `doSomething()`, which is registered to element2 for the bubbling phase.
3. The event travels upwards again and checks if any ancestor element of the target has an event handler for the bubbling phase.
4. The event finds one on element1. Now `doSomething2()` is

### **Blocco della propagazione degli eventi**

- La propagazione degli eventi può essere interrotta!
- In Explorer si effettua il seguente assegnamento  
`window.event.cancelBubble = true;`
- Nel W3C model si chiama la funzione `a.stopPropagation()`;
- La compatibilità si può risolvere con la seguente funzione  

```
function stopPropagazione(e) {
    if(!e) var e = window.event;
    e.cancelBubble = true;
    if(e.stopPropagation) e.stopPropagation();
}
```

### **Current target**

- Durante la fase di *capturing* e quella di *bubbling* il target non cambia: è sempre quello relativo all'elemento2 (ripeniamo all'esempio di prima).
- **Come riconosciamo l'elemento HTML che in quel momento sta gestendo l'evento?** Abbiamo detto che `target/srcElement` non aiutano, poiché restituiscono l'elemento2. Si risolve utilizzando la keyword `this`, che si riferisce all'oggetto che sta eseguendo il proprio handler.

### **Oggetto event**

- Quando avviene un evento il browser crea un oggetto evento disponibile agli handler.
- L'oggetto fornisce informazioni relative agli eventi.
- L'oggetto `event` presenta le seguenti proprietà/funzioni

Property/Method	Description
<a href="#">bubbles</a>	Returns whether or not a specific event is a bubbling event
<a href="#">cancelBubble</a>	Sets or returns whether the event should propagate up the hierarchy or not
<a href="#">cancelable</a>	Returns whether or not an event can have its default action prevented
<code>composed</code>	Returns whether the event is composed or not
<a href="#">createEvent()</a>	Creates a new event
<a href="#">currentTarget</a>	Returns the element whose event listeners triggered the event
<a href="#">defaultPrevented</a>	Returns whether or not the <code>preventDefault()</code> method was called for the event
<a href="#">eventPhase</a>	Returns which phase of the event flow is currently being evaluated
<a href="#">preventDefault()</a>	Cancella l'evento se questo è cancellabile. Questo significa che le azioni eseguite di default con l'evento non avverranno ( <b>Esempio:</b> sottomissione di un form)
<a href="#">stopImmediatePropagation()</a>	Prevents other listeners of the same event from being called

<a href="#">stopPropagation()</a>	Prevents further propagation of an event during event flow
<a href="#">target</a>	Returns the element that triggered the event
<a href="#">timeStamp</a>	Returns the time (in milliseconds relative to the epoch) at which the event was created
<a href="#">type</a>	Returns the name of the event

- Esistono altri oggetti evento, ciascuno con specifiche proprietà e funzioni. Tutti questi oggetti hanno accesso alle proprietà e alle funzioni del classico oggetto `Event`. I più importanti sono i seguenti:

Event Object	Description
<a href="#">AnimationEvent</a>	For CSS animations
<a href="#">ClipboardEvent</a>	For modification of the clipboard
<a href="#">DragEvent</a>	For drag and drop interaction
<a href="#">FocusEvent</a>	For focus-related events
<a href="#">HashChangeEvent</a>	For changes in the anchor part of the URL
<a href="#">InputEvent</a>	For user input
<a href="#">KeyboardEvent</a>	For keyboard interaction
<a href="#">MouseEvent</a>	For mouse interaction
<a href="#">PageTransitionEvent</a>	For navigating to, and away from, web pages
<a href="#">PopStateEvent</a>	For changes in the history entry
<a href="#">ProgressEvent</a>	For the progress of loading external resources
<a href="#">StorageEvent</a>	For changes in the window's storage area.
<a href="#">TouchEvent</a>	For touch interaction
<a href="#">TransitionEvent</a>	For CSS transitions
<a href="#">UiEvent</a>	For user interface interaction
<a href="#">WheelEvent</a>	For mousewheel interaction

**Per le proprietà relative a tutti questi oggetti e per una lista molto ricca di eventi captabili andare in fondo alla sezione del Javascript**

#### **Compatibilità tra i browser**

- Tipicamente l'oggetto evento viene passato all'handler
- In Explorer ciò non avviene: l'oggetto viene reso disponibile come una proprietà dell'oggetto `Window`.
- Risolviamo la questione attraverso una funzione handler

```
function handler(e) {
    e = (!e) ? window.event : e;
}
```

Se `e` è nullo allora il parametro viene inizializzato con `window.event` (ciò avviene su IE)
- Altra differenza si ha relativamente ai nomi delle proprietà: se vogliamo trovare il riferimento all'elemento dove l'evento si è generato dovremo
  - o Utilizzare l'attributo `target` (in Firefox)
  - o Utilizzare l'attributo `srcElement` (in Explorer)
- La questione si risolve anche in questo caso mediante operatore condizionale

```
obj = (e.target != null) ? e.target : e.srcElement;
```
- Queste questioni sono poste a scopo esclusivamente informativo. Concretamente non ci interfaceremo con questi problemi di compatibilità (il pratico e il progetto dovranno essere fatti con i browser portable del pacchetto All-in-one)

#### **Uso degli event handlers**

- Gli event handlers possono essere usati in tre modi diversi per triggerare una funzione
  - o **Associando l'handler a un evento nelle ancore (nei link)**

```
<a href="#" onClick="alert('Ooo, do it again!');">Click on me!</a>
<a href="javascript:void('')" onClick="alert('Ooo, do it again!');">
    Click on me!
</a>
```

```
<a href="javascript:alert('Ooo, do it again!')" >Click on me!</a>
```

Non serve il tag script: tutto ciò che si trova all'interno dell'attributo onClick (in questo caso questo attributo) viene interpretato come Javascript. Inoltre:

- href="#"  
riferisce al browser di cercare una certa ancora, ma questa non viene trovata e quindi il browser va in cima alla pagina
- javascript:void('')  
riferisce al browser di non andare da nessuna parte

○ **Azioni all'interno delle form per verificare la validità degli input**

```
<html>
<head>
<script type="text/javascript">
function checkField(fld){
    if (fld.checkValidity() && fld.value>100) {
        alert("Correct number");
        fld.style.backgroundColor="white";
    }
    else {
        alert("Please enter a number greater than 100");
        fld.value=null;
        fld.style.backgroundColor="red";
        setTimeout(function(){
            document.getElementById('idname').focus();},1000);
    }
}

function setStyle(fld) {
    fld.style.backgroundColor="yellow"
}
</script>
</head>

<body>
<form id="frm1" action="form_action.asp">
    <p>
        Insert a Number:
        <input type="input" required pattern="^[0-9]+$" id="idname"
        onfocus="setStyle(this)" onchange="checkField(this)">
    </p>
</form>
</body>
</html>
```

### **Timer events**

- Gli eventi possono essere generati utilizzando un timer. Abbiamo, a tal proposito, una serie di funzioni:
  - setInterval("function()", delay)  
chiama una funzione in modo ripetitivo a intervalli di tempo impostati dal secondo parametro. Delay è un parametro espresso in millisecondi. La funzione sarà chiamata ripetutamente finchè non si chiamerà la clearInterval (o finchè non sarà chiusa la pagina).
  - clearInterval(id\_of\_setinterval)  
l'unico argomento è l'identificatore restituito dalla funzione setInterval. Permette di bloccare l'esecuzione ripetuta di funzioni indicata con la setInterval.
  - setTimeout("function()", delay)  
chiama una funzione dopo un numero specifico di millisecondi.



- `clearTimeout(id_of_settimeout)`  
azzerà il contatore impostato con la `setTimeout`. Questa funzione permette di impedire l'esecuzione di una funzione dopo averne programmato l'esecuzione con la `setTimeout`. L'argomento è l'identificatore restituito dalla `setTimeout`.

## Effetti dinamici

- Possiamo ottenere effetti dinamici combinando eventi e proprietà di oggetti differenti.
- Vedremo alcuni esempi:
  - **Rollover**, cioè il cambio di un'immagine quando l'utente passa sopra di essa con il cursore del mouse.

```
<body>
  <div>
    
    
  </div>
  <script type="text/javascript" src="./myscript13.js"></script>
</body>
```

### //myscript13.js

```
var img1 = new Array(2); //
img1[0] = new Image(); img1[1] = new Image();
var img2 = new Array(2); img2[0] = new Image();
img2[1] = new Image();
img1[0].src="a.jpg"; img1[1].src="b.gif"; //default images
img2[0].src="b.gif"; img2[1].src="a.jpg"; //rollover images
```

```
function modify(i, type) {
  if (type == "over")
    document.images[i].src = img2[i].src; //mouseover
  else
    document.images[i].src = img1[i].src; //mouseout
}
```

- Gestisco le due immagini attraverso due array aventi per elementi le immagini (non si pone direttamente il link dell'immagine, ma l'oggetto `Image`).
- La funzione `modify` fa il resto: viene chiamata sia quando ci poniamo sopra l'immagine che quando ne usciamo: ha per parametri l'indicativo dell'immagine (in questo caso ci comportiamo come in DOM 0, l'indice consiste nella posizione dell'immagine rispetto alle altre) e l'azione che stiamo eseguendo (*over* se andiamo sopra e *out* se ne usciamo)
- Si modificano le proprietà delle immagini, in particolare la proprietà `src` che contiene l'URL dell'immagine.

- **Testo scorrevole**

```
<html>
<head>
<style type="text/css">
  #slidText { color : red; background:black}
</style>
</head>
<body onload="setInterval('slide()',100);">
  <form action="#">
    <p>
      <input type="text" size="30" name="mytext"
        id = "slidText" style="border: solid;">
    </p>
  </form>
```

```

        <script type="text/javascript" src="./myscript12.js"></script>
    </body>
</html>

```

#### //myscript14.js

```

var text = "Example of Sliding Text .....";
function slide() {
    var firstchar = text.charAt(0);
    text = text.slice(1,text.length) + firstchar;
    document.forms[0].mytext.value = text
}

```

- Attraverso la `onLoad` e la `setInterval` stabilisco l'esecuzione della funzione `slide()` ogni 100 millisecondi.
- La funzione `slide` si occupa di rendere il testo scorrevole e lo fa in una maniera molto rustica: il testo da far scorrere è posto in una variabile, ogni volta prendo il primo carattere e lo sposto in fondo (un po' come la SHIFT LEFT THROUGH CARRY)

#### ○ Gestione dei *layer*.

- Con *layers* intendiamo strati, quindi la possibilità per gli sviluppatori di imporre un contenuto sopra un altro contenuto.
- Abbiamo già visto quali sono le proprietà che permettono di gestire i *layers* e sovrapporre elementi `div` dell'HTML
  - *position*, che identifica la posizione dell'elemento nella pagina (il suo comportamento relativamente al flusso)
  - *left, right, top, bottom*: identificano la posizione dell'elemento relativamente alla distanza dai margini
  - *height, width*: lunghezza e larghezza dell'elemento
  - *z-index*: posizione dell'elemento relativamente all'asse z (chi sta più in alto?)
  - *visibility*: determina se l'elemento è visibile o meno
- Presente un esempio sostanzioso dalla diapositiva 301 delle dispense sul Javascript.

#### ○ Validazioni dei dati inseriti mediante form

- Quanto segue sono strategie di validazione dei dati utilizzate in passato, soprattutto prima della nascita di HTML5.
- Javascript permette di verificare in lato client se i dati posti nelle form sono corretti.
- La cosa è vantaggiosa perché ci permette di evitare il caricamento della pagina e un controllo lato server molto più dispendioso e fastidioso.

#### ▪ Esempio 1:

```

<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Validation of modules</title>
</head>
<body>
    <h1>Validation</h1>
    <form action="#" name="mymodule" id="mymodule">
        <p>
            Name (*):<br>
            <input type="text" name="Name" maxlength="30" value="Paul"><br>
            Surname (*):<br>
            <input type="text" name="Surname" maxlength="30" value="Red"><br>
            Age:<br>
            <input type="text" name="Age" maxlength="3" value="64"><br>
            City:<br>
            <input type="text" name="City" maxlength="30" value="Pisa"><br>
            Zip Code (*):<br>
            <input type="text" name="ZIP" maxlength="5" value="56125"><br>

```

```

        <input type="button" value="Validate" onclick="validate()">
    </p>
</form>
<script type="text/javascript">
    var fields = document.getElementsByTagName("INPUT");
    var shouldBeALPHANUMERICAL = [0, 1, 4];    //(1)
    var shouldBeNUMERICAL      = [2, 4];
    var shouldBeALPHABETICAL   = [0, 1, 3];
    var shouldBe5NUMBERS       = [4];

    var MESSAGES = [
        "The following field does not have valid symbols: ",
        "The following field is not exclusively numerical: ",
        "The following field must have five digits: ",
        "The following field must have only letters: ",
        "...",
        "OK"];

    var existALPHANUMERICAL     = /\w/; //(2)
    var existNONALPHANUMERICAL = /\W/; //(3)
    var existNONNUMERICAL       = /\D/; //(4)
    var existNUMERICAL          = /\d/; //(5)
    var exist5NUMERICAL         = /\d{5}/; //(6)

    function error(idmess, field) {
        window.alert(MESSAGES[idmess] + field.name);
        field.focus();    field.select();
    }

    function isTrue(COND, ELEM, BOOL, MESS) { //(7)
        for (var i=0; i<ELEM.length; i++) {
            var j = ELEM[i];
            field = fields[j];
            if (COND.test(field.value) == BOOL) {    //(8)
                error(MESS, field);
                return true;
            }
        }
        return false;
    }

    function validate() {
        if (isTrue(existALPHANUMERICAL, shouldBeALPHANUMERICAL, false, 0))
            return; //(9)
        if (isTrue(existNONALPHANUMERICAL, shouldBeALPHANUMERICAL, true, 0))
            return; //(10)
        if (isTrue(existNONNUMERICAL, shouldBeNUMERICAL, true, 1))
            return; //(11)
        if (isTrue(exist5NUMERICAL, shouldBe5NUMBERS, false, 2))
            return; //(12)
        if (isTrue(existNUMERICAL, shouldBeALPHABETICAL, true, 3))
            return; //(13)
        if (isTrue(existNONALPHANUMERICAL, shouldBeALPHABETICAL, true, 3))
            return; //(14)

        window.alert(MESSAGES[MESSAGES.length-1]);
    }
</script>
</body>
</html>

```

## Validation

Name (\*):

Surname (\*):

Age:

City,:

Zip Code (\*):



Il codice contiene

- Una funzione `error` che permette di gestire in modo agile gli errori da mostrare. Oltre a segnalare l'errore la funzione imposta il focus sull'elemento con l'errore e lo seleziona.

Name (\*):

Paulcbchcvb1

Surname (\*):

*Focus e selezione dopo aver tentato di inviare i dati*

- Una funzione `isTrue`.
  - Una condizione `COND`. La condizione è rappresentata da un `RegExp`. I vari `RegExp` utilizzati sono in cima al codice Javascript.
  - Una lista di elementi `ELEM` (ogni input è identificato da un indice numerico che dipende dalla sua posizione nel codice). Le liste (cioè gli array) sono salvati in cima al codice Javascript.
  - Un booleano `BOOL`, che consiste nel risultato che deve restituire la verifica della condizione.
  - L'identificativo `MESS` di un messaggio di errore da stampare. L'array con i messaggi è salvato in cima al codice Javascript.

La funzione analizza tutti gli input posti nella lista `ELEM` utilizzando la funzione `test()` tipica degli oggetti `RegExp` (se si ha dubbi a riguardo tornare indietro). La funzione `isTrue()` chiama subito la `error()` e restituisce `true` se individua un input con la condizione soddisfatta.

Questa pagina dice

The following field must have only letters: Name

Se ciò non avviene arriva alla fine del codice e restituisce `false`.

- Una funzione `validate()` che introduce tutte le condizioni da verificare e chiama diverse volte la funzione `isTrue()`
  - Se individua una condizione soddisfatta l'esecuzione della funzione viene fermata con `return`.
  - Se tutte le condizioni non sono soddisfatte arriva alla fine e stampa l'ultimo messaggio presente nell'array `MESSAGES` ("OK")

La funzione viene *triggerata* dal click dell'input di tipo button.

#### ▪ Esempio 2:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Form per Quiz</title>
  <script type="text/javascript">
    var Queries = ["What is the keyword in Javascript for defining a function?",
                  "What is not a valid comment in Javascript?",
                  "What is the handler for the 'loss of active state' event?",
                  "Which operator can be used to instance an object?",
                  "To periodically call a function you use the method:"];

    var Options = [ ["var", "function", "script"],
                    ["\\*...*\\", "**/.../*", " //...", "///...//"],
                    ["onFocus", "onBlur", "onClick"],
                    ["create", "new", "add"],
                    ["window.setInterval", "window.setTimeout", "date.setTime"] ];

    var Answers = [ 0,1,0, 0,1,0,0, 0,1,0, 0,1,0, 1,0,0];

    function check() {
      var score = 0;
      var answers = document.getElementsByTagName("INPUT");
      for (var i = 0; i < answers.length-1; i++)
        if (answers[i].checked)
          if (Answers[i]==1)
            score++;
    }
```

```

        else
            score--;
        window.alert("Your score is: " + score);
        document.mymodule.reset();
    }
</script>
</head>

<body>
    <h1>Quiz</h1>
    <form action="#" name="mymodule" id="mymodule">
        <script type="text/javascript">
            for (var i=0; i<Queries.length; i++) {
                document.writeln("<div>" + (i+1) + ") " + Queries[i] + "<br>");
                for (var j=0; j < Options[i].length; j++)
                    document.writeln("<input type='radio' name='q" + i + "'" + Options[i][j] +
"<br>");
                    document.writeln("<hr></div>");
            }
        </script>
        <p>
            <input type="button" value="VERIFY" onclick="check()" ">
        </p>
    </form>
</body>

</html>

```

- Il codice contiene

- Degli array, posti in cima al codice, che definiscono la struttura del quiz:
  - Queries: domande rivolte all'utente
  - Options: risposte possibili per ciascuna domanda
  - Answers: risposte corrette

Un'altra parte di codice genera la struttura del quiz (codice HTML con div ed input) sfruttando le informazioni presenti negli array.

- Una funzione `check()` che viene triggerata dal click dell'input di tipo button.
  - Con la `getElementsByName` si pongono tutti gli elementi input del documento in un array.
  - Con un `for` scorro tutti gli input. Per ciascuno:
    - Verifico se l'utente ha indicato una risposta
    - Se l'ha indicata verifico se è corretta
      - Se è corretta incremento lo score
      - Se è scorretta decremento lo score (quindi posso ottenere un punteggio negativo)
    - Restituisco all'utente il punteggio complessivo
    - Resetto tutti gli input.

# WebStorage

Con WebStorage intendiamo tutto ciò che può essere salvato lato client in modo permanente.

**Premessa:** per lavorare con gli strumenti di questa parte è necessario avere il pacchetto All-in-one e attivare Apache (protocollo http).

## Cookie

- Un cookie consiste in un pezzo di testo memorizzato dal browser dell'utente.
- Abbiamo già detto che un web server, dopo aver inviato la pagina web al browser, chiude la connessione dimentica qualunque cosa relativa a colui che ha richiesto il contenuto. I cookie sono stati ideati per risolvere questo problema.
- Un cookie può essere usato per gestire autenticazioni, ospitare preferenze dell'utente relative al sito (per esempio l'aspetto grafico), ricordarsi quali sono i contenuti visitati in passato dall'utente e offrire contenuti migliori per lui durante le future visite.
- Il supporto ai cookie può essere disattivato dal browser, attenzione.
- Un cookie consiste in una o più coppie nome-valore che contengono informazioni. Solitamente il cookie non contiene grandi informazioni, ma si limita a cose semplici.
- Il cookie viene inviato attraverso l'instestazione http da un server al browser e viene rimandato indietro dal browser ogni volta che accediamo al server. Vedremo i meccanismi precisi nell'ultima parte del corso sul protocollo http.
- I cookie possono essere impostati con o senza una data di scadenza
  - o Cookie senza data di scadenza esistono finchè il browser non viene chiuso.
  - o Cookie con una data di scadenza vengono ospitati dal browser finchè quella data non viene superata.

È buona abitudine per un utente cancellare periodicamente i cookie salvati. Essere tracciati certe volte è vantaggioso, ma altre volte può risultare estremamente pericoloso per la sicurezza dei nostri dati.

- **Parametri relativi a un cookie:**
  - o name: nome del cookie
  - o value: valore del cookie
  - o expire: data di scadenza (espressa in formato UTC)
  - o path: percorso sul server dove il cookie sarà disponibile. Il valore di default è "/": si rende disponibile il cookie sull'intero dominio.
  - o domain: il dominio dove il cookie sarà reso disponibile. Se vogliamo rendere disponibile il dominio anche ai sottodomini dovremo indicare un punto prima del dominio. Esempio: .example.com
  - o secure: indica se il cookie dovrà essere trasmesso solo mediante una connessione sicura HTTPS col server. Booleano.
- **Funzioni per la gestione dei cookie:**
  - o Javascript può creare, leggere ed eliminare cookies utilizzando la proprietà `document.cookie`. La stringa si comporta in modo strano: tutte le volte andiamo a inserire un nuovo cookie utilizzando un'operazione di assegnamento (e non di concatenazione). Il nuovo assegnamento non va a sovrascrivere quelli precedenti: se si stampa la `document.cookie` dopo una serie di assegnamenti troveremo una lista di coppie nome-valore separate da punto e virgola.
  - o Per fare ciò utilizzeremo le seguenti funzioni (non sono funzioni built-in offerte da Javascript). Le funzioni proposte sono di w3schools (tranne la delete, che è una versione modificata di quella di Marcelloni). Preferisco usare queste poiché più semplici e intuitive (con le cose che ci servono).

```
function setCookie(cname, cvalue, exdays) {  
    var d = new Date();  
    d.setTime(d.getTime() + (exdays*24*60*60*1000));  
    var expires = "expires="+ d.toUTCString();  
    document.cookie = cname + "=" + cvalue + ";" + expires + ";path=/";  
}
```

```

function getCookie(cname) {
    var name = cname + "=";
    var decodedCookie = decodeURIComponent(document.cookie);
    var ca = decodedCookie.split(';');
    for(var i = 0; i <ca.length; i++) {
        var c = ca[i];
        while (c.charAt(0) == ' ') {
            c = c.substring(1);
        }
        if (c.indexOf(name) == 0) {
            return c.substring(name.length, c.length);
        }
    }
    return "";
}

function deleteCookie( name) {
    if ( getCookie( name ) )
document.cookie = name + "=" + ";path=/" + ";expires=Thu, 01-Jan-1970 00:00:01 GMT";
}

```

- La prima funzione svolge un'operazione di assegnamento indicando tre dati: nome, valore e data di scadenza. Relativamente alla `path` si pone un valore di default (si rende il cookie disponibile su tutto il dominio).
- La seconda funzione attraverso un lavoro di ritaglio estrae il cookie che ci interessa. Ricordiamo come si struttura la proprietà `document.cookie`.
  - Si utilizza la funzione `split` per dividere ogni coppia nome valore.
  - Si scorre l'array generato
  - Si ignorano eventuali caratteri vuoti iniziali (li cancelliamo)
  - Si verifica con la `indexOf` se il nome della coppia coincide col parametro in ingresso.
  - Se coincide restituiamo il valore
  - Se scorrendo tutto il `for` non troviamo niente si restituisce una stringa vuota.
- La terza funzione permette la rimozione di cookie: molto semplicemente sovrascriviamo il cookie già esistente indicando come data di scadenza una data anteriore a quella attuale. Questo comporta la cancellazione del cookie.

- **Esempio di applicazione:** creiamo la seguente funzione

```

function checkCookie() {
    var user = getCookie("username");
    if (user != "") {
        alert("Welcome again " + user);
    } else {
        user = prompt("Please enter your name:", "");
        if (user != "" && user != null) {
            setCookie("username", user, 365);
        }
    }
}

```

- Recupero dai cookie l'username.
- Se l'username è già stato impostato restituisco un avviso di bentornato
- Altrimenti richiedo con la `prompt` di indicare un username. A quel punto la `setCookie` salva l'username per una durata di 365 giorni.

## SessionStorage e LocalStorage

- Prima dell'HTML5 i dati delle applicazioni erano ospitati esclusivamente in cookies, inclusi in ogni singola richiesta al server.
- Il `sessionStorage` e `localStorage` è molto più sicuro (i dati non saranno mai trasferiti al server), non influenza la performance di caricamento del sito e permette di ospitare informazioni di dimensione maggiore.
- **Abbiamo i seguenti oggetti:**
  - o `window.localStorage`, i dati possono essere consultati all'interno delle finestre o tab del browser e non hanno data di scadenza: saranno preservati anche dopo la chiusura del browser o della pagina relativa.
  - o `window.sessionStorage`, i dati persistono all'interno di una finestra o di un tab (sono visibili solo lì). Saranno eliminati con la chiusura del browser o della pagina relativa.
- **Compatibilità:** prima di lavorare con questi oggetti dobbiamo verificare la loro compatibilità. Per fare ciò prendiamo come riferimento le seguenti funzioni, che segnalano il supporto o meno del `sessionStorage` e del `localStorage`

```
function checkStorageSupport() {  
    //sessionStorage  
    if (window.sessionStorage) {  
        alert('This browser supports sessionStorage');  
    } else {  
        alert('This browser does NOT support sessionStorage');  
    }  
  
    //localStorage  
    if (window.localStorage) {  
        alert('This browser supports localStorage');  
    } else {  
        alert('This browser does NOT support localStorage');  
    }  
}
```

- **Funzioni e proprietà utilizzabili:**
  - o `length`, specifica quante coppie chiave-valore sono memorizzate nell'oggetto storage.
  - o `key(index)`  
restituisce una chiave memorizzata all'interno dello storage. Se si pone `index = 0`, per esempio, restituiamo la chiave relativa alla prima coppia memorizzata nello storage (come un array)
  - o `getItem(key)`  
restituisce il valore di una coppia data una chiave in ingresso
  - o `setItem(key, value)`  
permette di impostare un valore nello storage. Possiamo indicare una key già usata, quindi aggiornare, ma anche una key nuova, quindi aggiungere qualcosa di nuovo.
  - o `removeItem(key)`  
permette di rimuovere un valore dallo storage ponendo in ingresso la chiave identificativa. Ovviamente non si fa niente se la chiave non è utilizzata
  - o `clear()`  
rimuove tutti i valori dallo storage.
- **Evento:** l'evento `storage` si ha in caso di cambiamenti nel web storage utilizzando gli strumenti appena introdotti. Possiamo scrivere, per esempio, quanto segue



```
function displayStorageEvent(e) {
    var logged = "key:" + e.key + ", newValue:" + e.newValue + ", + oldValue:" +
    e.oldValue + ", url:" + e.url + ", storageArea:" + e.storageArea;
    alert(logged);
}
```

```
window.addEventListener("storage", displayStorageEvent, true);
```

Poco più avanti sono descritti gli attributi relativi all'oggetto `storageEvent` (utilizzati nell'esempio)

### Esempio di localStorage

Key:

Value:

Contents of Local Storage:  
'ciao\_Marco' = 'Come stai?'

```
<!DOCTYPE html>
<html>
<head>
    <title>Web Storage Example</title>
    <meta charset="UTF-8">
    <script>
        window.addEventListener("load", function(event) {
            var key = document.getElementById("key"); // input key
            var value = document.getElementById("value"); // input value
            var add = document.getElementById("add"); // bottone "Add to Storage"
            var remove = document.getElementById("remove"); // bottone "Remove from Storage"
            var clear = document.getElementById("clear"); // bottone "Clear Storage"
            var content = document.getElementById("content"); // "Contents of Local Storage..."

            add.addEventListener("click", function(event) {
                if (key.value !== "") {
                    try {
                        localStorage.setItem(key.value, value.value);
                    }
                    catch (e) {
                        alert("Exceeded Storage Quota!");
                    }
                    refreshContents();
                }
            });

            remove.addEventListener("click", function(event) {
                if (key.value !== "") {
                    localStorage.removeItem(key.value);
                    refreshContents();
                }
            });

            clear.addEventListener("click", function(event) {
                localStorage.clear();
                refreshContents();
            });

            window.addEventListener("storage", function(event) {
                var k = event.key;
                var newValue = event.newValue;
                var oldValue = event.oldValue;
```

```

    var url = event.url;
    var storageArea = event.storageArea;

    alert("EVENT: " + k + " newValue= " + newValue + " oldValue= " +
oldValue + " url= " + url + " storageArea= " + storageArea);
    refreshContents();
});

function refreshContents() {
    while (content.firstChild) {
        content.removeChild(content.firstChild);
    }

    var k;
    var txt, linebreak;
    for (var i = 0, len = localStorage.length; i < len; i++) {
        k=localStorage.key(i);
        str = "'" + k + "' = '" + localStorage.getItem(k) + "'";
        txt=document.createTextNode(str);
        content.appendChild(txt);
        linebreak=document.createElement('br');
        content.appendChild(linebreak);
    }
    key.value = "";
    value.value = "";
}
refreshContents();
});
</script>
</head>
<body>
    Key: <input type="text" id="key"><br>
    Value: <input type="text" id="value"><br>
    <input type="button" id="add" value="Add to Storage">&nbsp;
    <input type="button" id="remove" value="Remove from Storage">&nbsp;
    <input type="button" id="clear" value="Clear Storage"><br>
    Contents of Local Storage:<br>
    <div id="content"></div>
</body>
</html>

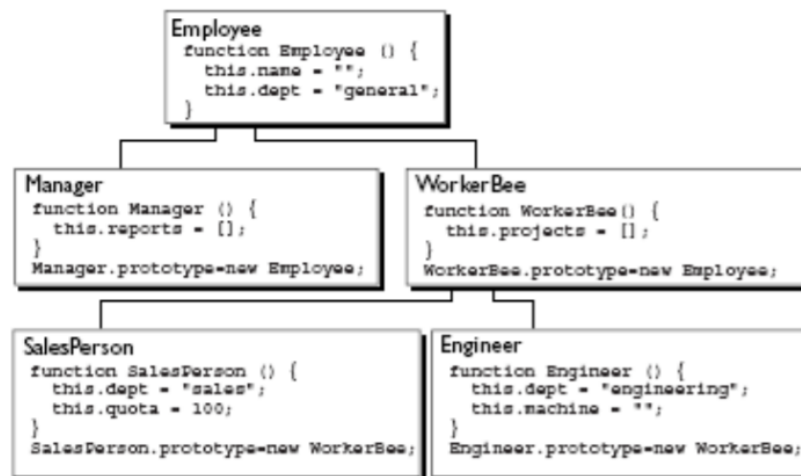
```

- Il codice Javascript sarà “eseguibile” dopo il caricamento completo della pagina.
- Con le `document.getElementById` si recuperano tutti gli input utilizzati nel body.
- Analizziamo le funzioni triggerate dai vari eventi:
  - o `add.addEventListener("click" ...`  
 Verifico che l’input key non sia vuoto, se non lo è aggiorno lo storage utilizzando i valori degli input key e value. Con la try catch gestisco un eventuale fallimento nell’inserimento (dovuto allo storage pieno). Alla fine eseguo la `refreshContents()`
  - o `remove.addEventListener("click" ...`  
 Verifico che l’input key non sia vuoto, se non lo è procedo all’eliminazione utilizzando il valore dell’input stesso. Alla fine eseguo la `refreshContents()`
  - o `clear.addEventListener("click" ...`  
 Non svolgo controlli particolari: elimino il contenuto dello storage ed eseguo la `refreshContents()`.
  - o `window.addEventListener("storage" ...`  
 Invio un alert contenente le informazioni relative ai cambiamenti compiuti nello storage.

- Funzione `refreshContents()`
  - o Chiamata dalle funzioni precedentemente introdotte
  - o Sfruttando un meccanismo alternativo all'uso della `innerHTML` svuota il div con id `content` (quello che contiene quanto salvato nello storage): con un `while` verifico ogni volta se ho un primo figlio, se lo ho lo cancello.

## Inheritance

- Abbiamo già detto che Javascript è un linguaggio **object-based**, in contrapposizione al C++ che è **object-oriented**: in Javascript abbiamo oggetti come contenitori da riempire, non classi come in C++.
- Formalmente i meccanismi relativi all'ereditarietà visti ad *Algoritmi e strutture dati* non sono disponibili.
- Esistono, tuttavia, dei meccanismi che permettono di stabilire un'ereditarietà (quindi è possibile stabilire che un oggetto erediti le proprietà e le funzioni di un altro oggetto).
- Prendiamo il seguente schema



- o *Employee* è l'oggetto padre.
- o *Manager* e *WorkerBee* ereditano proprietà e funzioni di *Employee*.
- o *SalesPerson* ed *Engineer* ereditano proprietà e funzioni di *WorkerBee*, quindi anche le proprietà e le funzioni di *Employee*.
- L'operatore `new` crea un oggetto generico. L'oggetto appena creato viene passato alla funzione costruttore.
- Javascript si comporta nel seguente modo quando si richiede il valore di una proprietà di un oggetto:
  - o Verifica l'esistenza di quella proprietà nell'oggetto. Se esiste ne restituisce il valore.
  - o Se la proprietà non esiste nell'oggetto Javascript controlla la catena `prototype`.
  - o **Se in un oggetto di quella catena esiste la proprietà indicata allora ne viene restituito il valore.**
  - o Se arrivati a questo punto non abbiamo trovato niente allora non esiste la proprietà richiesta.
- Costruttori con argomenti: come gestiamo gli argomenti del costruttore? Abbiamo due strade, di cui una il proseguo dell'altra.
  - o **Argomenti inizializzati con valori default**  
Si introduce una sorta di operatore logico OR nelle righe in cui inizializziamo i valori delle proprietà del nostro oggetto. Questo ci permette di indicare un valore di default che sarà utilizzato in caso di chiamata del costruttore senza gli argomenti.

```

function Employee(name, dept) {
  // Pongo il valore passato mediante l'argomento name o lo spazio vuoto
  this.name = name || ' ';

  // Pongo il valore passato mediante l'argomento dept o la stringa "general"
  this.dept = dept || 'general';
}

```

```

function Manager() {
    this.reports = []; // Array vuoto
}
//Stabilisco l'ereditarietà (Manager ← Employee)
Manager.prototype=new Employee;

function WorkerBee(projs) {
    // Pongo il valore (si presume un array) passato con l'argomento projs oppure un array vuoto
    this.projects = projs || [];
}
//Stabilisco l'ereditarietà (WorkerBee ← Employee)
WorkerBee.prototype=new Employee;

function SalesPerson() {
    this.dept = 'sales';
    this.quota = 100;
}
//Stabilisco l'ereditarietà (SalesPerson ← WorkerBee ← Employee)
SalesPerson.prototype=new WorkerBee;

function Engineer(mach) {
    this.dept = 'engineering';

    // Pongo il valore passato con l'argomento match oppure una stringa vuota
    this.machine = mach || '';
}
//Stabilisco l'ereditarietà (Engineer ← WorkerBee ← Employee)
Engineer.prototype=new WorkerBee;

```

- **Argomenti inizializzati utilizzando i parametri del costruttore dell'oggetto figlio**  
Per indicare i valori del costruttore dell'oggetto padre indichiamo, nel costruttore dell'oggetto figlio, una base.

- Prima si indica il nome dell'oggetto  
this.base = Employee;
- Successivamente si indicano i parametri  
this.base(name, dept);

**Attenzione:** fare questo non è un'alternativa all'utilizzo della catena di prototype. Non includere l'operatore new non è problematico. L'oggetto figlio viene creato (e possiede le proprietà/funzioni dell'oggetto padre) ma non si stabilisce l'ereditarietà: significa che se aggiungerò in futuro nuove proprietà nell'oggetto padre queste non saranno ereditate dall'oggetto figlio creato prima.

```

function Employee(name, dept) {
    this.name = name || ' ';
    this.dept = dept || 'general';
}

function Manager() {
    this.reports = [];
}
Manager.prototype=new Employee;

function WorkerBee(name, dept,projs) {
    this.base = Employee; //Indico l'oggetto padre
    this.base(name,dept); //Indico i valori in ingresso nel costruttore
    this.projects = projs || [];
}
WorkerBee.prototype=new Employee;

function SalesPerson() {
    this.dept = 'sales';
    this.quota = 100;
}
SalesPerson.prototype=new WorkerBee;

```

```
function Engineer(name, projs, mach) {
    this.base = WorkerBee; //Indico l'oggetto padre
    this.base(name, 'engineering', projs); //Indico i valori in ingresso nel costruttore
    this.machine = mach || '';
}
Engineer.prototype=new WorkerBee;
```

### Oggetto evento AnimationEvent

Property/Method	Description
<a href="#">animationName</a>	Returns the name of the animation
<a href="#">elapsedTime</a>	Returns the number of seconds an animation has been running
<a href="#">pseudoElement</a>	Returns the name of the pseudo-element of the animation

### Oggetto evento ClipboardEvent

Property/Method	Description
<a href="#">clipboardData</a>	Returns an object containing the data affected by the clipboard operation

### Oggetto evento DragEvent

Property/Method	Description
<a href="#">dataTransfer</a>	Returns the data that is dragged/dropped

### Oggetto evento FocusEvent

Property/Method	Description
<a href="#">relatedTarget</a>	Returns the element related to the element that triggered the event

### Oggetto evento HashChangeEvent

Property/Method	Description
<a href="#">newURL</a>	Returns the URL of the document, after the hash has been changed
<a href="#">oldURL</a>	Returns the URL of the document, before the hash was changed

### Oggetto evento InputEvent

Property/Method	Description
<a href="#">data</a>	Returns the inserted characters
<a href="#">dataTransfer</a>	Returns an object containing information about the inserted/deleted data
<a href="#">getTargetRanges()</a>	Returns an array containing target ranges that will be affected by the insertion/deletion
<a href="#">inputType</a>	Returns the type of the change (i.e "inserting" or "deleting")
<a href="#">isComposing</a>	Returns whether the state of the event is composing or not

### Oggetto evento KeyboardEvent

Property/Method	Description
<a href="#">altKey</a>	Returns whether the "ALT" key was pressed when the key event was triggered
<a href="#">charCode</a>	Returns the Unicode character code of the key that triggered the event
<a href="#">code</a>	Returns the code of the key that triggered the event
<a href="#">ctrlKey</a>	Returns whether the "CTRL" key was pressed when the key event was triggered
<a href="#">getModifierState()</a>	Returns true if the specified key is activated
<a href="#">isComposing</a>	Returns whether the state of the event is composing or not
<a href="#">key</a>	Returns the key value of the key represented by the event

<a href="#">keyCode</a>	Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event
<a href="#">location</a>	Returns the location of a key on the keyboard or device
<a href="#">metaKey</a>	Returns whether the "meta" key was pressed when the key event was triggered
<a href="#">repeat</a>	Returns whether a key is being hold down repeatedly, or not
<a href="#">shiftKey</a>	Returns whether the "SHIFT" key was pressed when the key event was triggered
<a href="#">which</a>	Returns the Unicode character code of the key that triggered the onkeypress event, or the Unicode key code of the key that triggered the onkeydown or onkeyup event

### Oggetto evento MouseEvent

Property/Method	Description
<a href="#">altKey</a>	Returns whether the "ALT" key was pressed when the mouse event was triggered
<a href="#">button</a>	Returns which mouse button was pressed when the mouse event was triggered
<a href="#">buttons</a>	Returns which mouse buttons were pressed when the mouse event was triggered
<a href="#">clientX</a>	Returns the horizontal coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered
<a href="#">clientY</a>	Returns the vertical coordinate of the mouse pointer, relative to the current window, when the mouse event was triggered
<a href="#">ctrlKey</a>	Returns whether the "CTRL" key was pressed when the mouse event was triggered
<a href="#">getModifierState()</a>	Returns true if the specified key is activated
<a href="#">metaKey</a>	Returns whether the "META" key was pressed when an event was triggered
<a href="#">movementX</a>	Returns the horizontal coordinate of the mouse pointer relative to the position of the last mousemove event
<a href="#">movementY</a>	Returns the vertical coordinate of the mouse pointer relative to the position of the last mousemove event
<a href="#">offsetX</a>	Returns the horizontal coordinate of the mouse pointer relative to the position of the edge of the target element
<a href="#">offsetY</a>	Returns the vertical coordinate of the mouse pointer relative to the position of the edge of the target element
<a href="#">pageX</a>	Returns the horizontal coordinate of the mouse pointer, relative to the document, when the mouse event was triggered
<a href="#">pageY</a>	Returns the vertical coordinate of the mouse pointer, relative to the document, when the mouse event was triggered
<a href="#">relatedTarget</a>	Returns the element related to the element that triggered the mouse event
<a href="#">screenX</a>	Returns the horizontal coordinate of the mouse pointer, relative to the screen, when an event was triggered
<a href="#">screenY</a>	Returns the vertical coordinate of the mouse pointer, relative to the screen, when an event was triggered
<a href="#">shiftKey</a>	Returns whether the "SHIFT" key was pressed when an event was triggered
<a href="#">which</a>	Returns which mouse button was pressed when the mouse event was triggered

### Oggetto evento PageTransitionEvent

Property/Method	Description
<a href="#">persisted</a>	Returns whether the webpage was cached by the browser

### Oggetto evento PopStateEvent

Property/Method	Description
<a href="#">state</a>	Returns an object containing a copy of the history entries

## Oggetto evento ProgressEvent

Property/Method	Description
lengthComputable	Returns whether the length of the progress can be computable or not
loaded	Returns how much work has been loaded
total	Returns the total amount of work that will be loaded

## Oggetto evento StorageEvent

Property/Method	Description
<a href="#">key</a>	Restituisce la chiave dell'elemento modificato dello storage. (Ricordare: coppie key-value)
<a href="#">newValue</a>	Restituisce il nuovo valore dell'elemento modificato nello storage.
<a href="#">oldValue</a>	Restituisce il vecchio valore dell'elemento modificato nello storage.
<a href="#">storageArea</a>	Restituisce l'oggetto storage coinvolto.
<a href="#">url</a>	Restituisce l'URL del documento dove abbiamo modificato l'elemento dello storage.

## Oggetto evento TouchEvent

Property/Method	Description
<a href="#">altKey</a>	Returns whether the "ALT" key was pressed when the touch event was triggered
changedTouches	Returns a list of all the touch objects whose state changed between the previous touch and this touch
<a href="#">ctrlKey</a>	Returns whether the "CTRL" key was pressed when the touch event was triggered
<a href="#">metaKey</a>	Returns whether the "meta" key was pressed when the touch event was triggered
<a href="#">shiftKey</a>	Returns whether the "SHIFT" key was pressed when the touch event was triggered
<a href="#">targetTouches</a>	Returns a list of all the touch objects that are in contact with the surface and where the touchstart event occurred on the same target element as the current target element
<a href="#">touches</a>	Returns a list of all the touch objects that are currently in contact with the surface

## Oggetto evento TransitionEvent

Property/Method	Description
<a href="#">propertyName</a>	Returns the name of the transition
<a href="#">elapsedTime</a>	Returns the number of seconds a transition has been running
pseudoElement	Returns the name of the pseudo-element of the transition

## Oggetto evento UiEvent

Property/Method	Description
<a href="#">detail</a>	Returns a number with details about the event
<a href="#">view</a>	Returns a reference to the Window object where the event occurred

## Oggetto evento WheelEvent

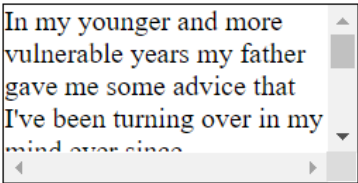
Property/Method	Description
deltaX	Restituisce un valore numerico che consiste nello scroll orizzontale della rotella del mouse. <ul style="list-style-type: none"><li>- Il valore è positivo quando si scorre a destra</li><li>- Il valore è uguale a 0 se non si hanno spostamenti orizzontali</li><li>- Il valore è negativo quando si scorre a sinistra</li></ul> <b>NB.</b> La maggior parte dei mouse non permette di fare scroll orizzontale.
deltaY	Restituisce un valore numerico che consiste nello scroll verticale della rotella del mouse. <ul style="list-style-type: none"><li>- Il valore è positivo quando si scorre in basso</li><li>- Il valore è uguale a 0 se non si hanno spostamenti verticali</li><li>- Il valore è negativo quando si scorre in alto</li></ul>
deltaZ	Restituisce un valore numerico che consiste nello scroll della rotella del mouse sull'asse z.

	<ul style="list-style-type: none"> <li>- Il valore è positivo quando si scorre verso il dentro</li> <li>- Il valore è uguale a 0 se non si hanno spostamenti lungo l'asse z</li> <li>- Il valore è negativo quando si scorre verso l'esterno</li> </ul> <p><b>NB.</b> La maggior parte dei mouse non permette di fare scroll lungo l'asse z.</p>
deltaMode	<p>Restituisce un numero che rappresenta l'unità di misura per le proprietà precedenti:</p> <ul style="list-style-type: none"> <li>- <b>0:</b> <i>pixels</i></li> <li>- <b>1:</b> <i>lines</i></li> <li>- <b>2:</b> <i>pages</i></li> </ul> <p>Proprietà <i>read-only</i>.</p>

Eventi captabili		
Evento captabile	Descrizione (in inglese)	Oggetti evento (proprietà e funzioni a cui avrà accesso)
<a href="#">abort</a>	The event occurs when the loading of a media is aborted	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">afterprint</a>	L'evento occorre se la pagina è in corso di stampa o se siamo usciti dalla schermata di stampa senza aver avviato operazioni.	<a href="#">Event</a>
<a href="#">animationend</a>	The event occurs when a CSS animation has completed	<a href="#">AnimationEvent</a>
<a href="#">animationiteration</a>	The event occurs when a CSS animation is repeated	<a href="#">AnimationEvent</a>
<a href="#">animationstart</a>	The event occurs when a CSS animation has started	<a href="#">AnimationEvent</a>
<a href="#">beforeprint</a>	L'evento occorre quando si richiede di fare una stampa (prima che appaia l'apposita schermata raggiungibile con CTRL+P)	<a href="#">Event</a>
<a href="#">beforeunload</a>	The event occurs before the document is about to be unloaded	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">blur</a>	L'evento occorre quando un elemento perde il focus.	<a href="#">FocusEvent</a>
<a href="#">canplay</a>	The event occurs when the browser can start playing the media (when it has buffered enough to begin)	<a href="#">Event</a>
<a href="#">canplaythrough</a>	The event occurs when the browser can play through the media without stopping for buffering	<a href="#">Event</a>
<a href="#">change</a>	L'evento occorre quando il valore del controllo di un form (input, select, textarea) viene cambiato. Funziona anche con radiobuttons e checkboxes: l'evento occorre quando lo stato selezionato viene cambiato. Molto simile all'evento <i>input</i> : la differenza è che l'evento <i>input</i> occorre in modo immediato, mentre l'evento <i>change</i> occorre solo dopo la perdita di focus dell'elemento.	<a href="#">Event</a>
<a href="#">click</a>	L'evento occorre quando l'utente fa click su un certo elemento.	<a href="#">MouseEvent</a>
<a href="#">contextmenu</a>	L'evento occorre quando l'utente clicca il tasto destro su un certo elemento.	<a href="#">MouseEvent</a>
<a href="#">copy</a>	L'evento occorre quando l'utente <i>copia</i> il contenuto di un certo elemento.	<a href="#">ClipboardEvent</a>
<a href="#">cut</a>	L'evento occorre quando l'utente <i>taglia</i> il contenuto di un certo elemento.	<a href="#">ClipboardEvent</a>
<a href="#">dblclick</a>	L'evento occorre quando l'utente fa doppio-click su un certo elemento.	<a href="#">MouseEvent</a>
<a href="#">drag</a>	The event occurs when an element is being dragged	<a href="#">DragEvent</a>
<a href="#">dragend</a>	The event occurs when the user has finished dragging an element	<a href="#">DragEvent</a>
<a href="#">dragenter</a>	The event occurs when the dragged element enters the drop target	<a href="#">DragEvent</a>
<a href="#">dragleave</a>	The event occurs when the dragged element leaves the drop target	<a href="#">DragEvent</a>
<a href="#">dragover</a>	The event occurs when the dragged element is over the drop target	<a href="#">DragEvent</a>



<a href="#">dragstart</a>	The event occurs when the user starts to drag an element	<a href="#">DragEvent</a>
<a href="#">drop</a>	The event occurs when the dragged element is dropped on the drop target	<a href="#">DragEvent</a>
<a href="#">ended</a>	The event occurs when the media has reach the end (useful for messages like "thanks for listening")	<a href="#">Event</a>
<a href="#">error</a>	L'evento occorre se si ha un errore nel caricamento di un file esterno (per esempio un'immagine)	<a href="#">ProgressEvent</a> , <a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">focus</a>	L'evento occorre quando si pone focus su un certo elemento.	<a href="#">FocusEvent</a>
<a href="#">hashchange</a>	L'evento occorre quando viene modificata la parte dell'URL relativa alle ancore (La parte dopo il cancelletto).	<a href="#">HashChangeEvent</a>
<a href="#">input</a>	L'evento occorre quando il valore degli input viene cambiato. Molto simile all'evento <i>change</i> : la differenza è che l'evento input occorre in modo immediato, mentre l'evento <i>change</i> occorre solo dopo la perdita di focus dell'elemento.	<a href="#">InputEvent</a> , <a href="#">Event</a>
<a href="#">invalid</a>	L'evento occorre quando input che può essere sottomesso non ha successo (per esempio se è presente l'attributo required).	<a href="#">Event</a>
<a href="#">keydown</a>	L'evento occorre quando l'utente sta premendo un tasto. Funziona con qualunque tasto.  <b>Ordine di esecuzione degli eventi legati alla pressione di un tasto:</b> 1. <i>onkeydown</i> 2. <i>onkeypress</i> 3. <i>onkeyup</i>	<a href="#">KeyboardEvent</a>
<a href="#">keypress</a>	L'evento occorre quando l'utente preme un tasto. L'evento in questione non viene attivato per tutti i tasti (attenzione ad ALT, CTRL, MAIUSC, ESC). Per rilevare se un utente ha premuto un certo tasto conviene utilizzare l'evento <i>keydown</i> .  <b>Ordine di esecuzione degli eventi legati alla pressione di un tasto:</b> 1. <i>onkeydown</i> 2. <i>onkeypress</i> 3. <i>onkeyup</i>	<a href="#">KeyboardEvent</a>
<a href="#">keyup</a>	L'evento occorre quando l'utente rilascia un tasto. Attenzione a mettere questo evento assieme a uno dei due precedenti: viene eseguito solo con una lunga pressione del tasto.  <b>Ordine di esecuzione degli eventi legati alla pressione di un tasto:</b> 1. <i>onkeydown</i> 2. <i>onkeypress</i> 3. <i>onkeyup</i>	<a href="#">KeyboardEvent</a>
<a href="#">load</a>	L'evento occorre quando un certo oggetto è stato caricato. Questo evento è usato molto frequentemente nell'elemento body per eseguire uno script dopo il caricamento completo della pagina (Formazione dell'albero DOM, in documenti complessi può richiedere tempo – cit.Tesconi).	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">loadeddata</a>	The event occurs when media data is loaded	<a href="#">Event</a>
<a href="#">loadedmetadata</a>	The event occurs when meta data (like dimensions and duration) are loaded	<a href="#">Event</a>
<a href="#">loadstart</a>	The event occurs when the browser starts looking for the specified media	<a href="#">ProgressEvent</a>
<a href="#">message</a>	The event occurs when a message is received through the event source	<a href="#">Event</a>

<a href="#">mousedown</a>	L'evento occorre quando l'utente preme un bottone del mouse col cursore sopra un certo elemento.	<a href="#">MouseEvent</a>
<a href="#">mouseenter</a>	L'evento occorre quando il cursore viene portato all'interno di un elemento.	<a href="#">MouseEvent</a>
<a href="#">mouseleave</a>	L'evento occorre quando il cursore viene portato fuori dall'elemento.	<a href="#">MouseEvent</a>
<a href="#">mousemove</a>	L'evento occorre quando il cursore si muove rimanendo all'interno dell'elemento.	<a href="#">MouseEvent</a>
<a href="#">mouseover</a>	The event occurs when the pointer is moved onto an element, or onto one of its children	<a href="#">MouseEvent</a>
<a href="#">mouseout</a>	The event occurs when a user moves the mouse pointer out of an element, or out of one of its children	<a href="#">MouseEvent</a>
<a href="#">mouseup</a>	L'evento occorre quando un utente rilascia un bottone del mouse col cursore sopra un certo elemento.	<a href="#">MouseEvent</a>
<a href="#">offline</a>	L'evento occorre quando il browser inizia a lavorare offline.	<a href="#">Event</a>
<a href="#">online</a>	L'evento occorre quando il browser inizia a lavorare online.	<a href="#">Event</a>
<a href="#">open</a>	The event occurs when a connection with the event source is opened	<a href="#">Event</a>
<a href="#">pagehide</a>	The event occurs when the user navigates away from a webpage	<a href="#">PageTransitionEvent</a>
<a href="#">pageshow</a>	The event occurs when the user navigates to a webpage	<a href="#">PageTransitionEvent</a>
<a href="#">paste</a>	L'evento occorre quando l'utente <i>incolla</i> del contenuto in un certo elemento.	<a href="#">ClipboardEvent</a>
<a href="#">pause</a>	The event occurs when the media is paused either by the user or programmatically	<a href="#">Event</a>
<a href="#">play</a>	The event occurs when the media has been started or is no longer paused	<a href="#">Event</a>
<a href="#">playing</a>	The event occurs when the media is playing after having been paused or stopped for buffering	<a href="#">Event</a>
<a href="#">popstate</a>	The event occurs when the window's history changes	<a href="#">PopStateEvent</a>
<a href="#">progress</a>	The event occurs when the browser is in the process of getting the media data (downloading the media)	<a href="#">Event</a>
<a href="#">ratechange</a>	The event occurs when the playing speed of the media is changed	<a href="#">Event</a>
<a href="#">resize</a>	L'evento occorre quando la finestra del browser, quella con cui stiamo visualizzando il documento, viene ridimensionata.	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">reset</a>	L'evento occorre quando un certo form viene resettato.	<a href="#">Event</a>
<a href="#">scroll</a>	L'evento occorre quando la scrollbar di un certo elemento viene utilizzata. 	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">search</a>	The event occurs when the user writes something in a search field (for <input="search">)	<a href="#">Event</a>
<a href="#">seeked</a>	The event occurs when the user is finished moving/skipping to a new position in the media	<a href="#">Event</a>
<a href="#">seeking</a>	The event occurs when the user starts moving/skipping to a new position in the media	<a href="#">Event</a>
<a href="#">select</a>	L'evento occorre quando l'utente seleziona del testo. Evento utilizzato soprattutto per gli input di testo e le textarea.	<a href="#">UiEvent</a> , <a href="#">Event</a>

<a href="#">show</a>	The event occurs when a <menu> element is shown as a context menu	<a href="#">Event</a>
<a href="#">stalled</a>	The event occurs when the browser is trying to get media data, but data is not available	<a href="#">Event</a>
storage	The event occurs when a Web Storage area is updated.	<a href="#">StorageEvent</a>
<a href="#">submit</a>	L'evento occorre quando un certo form viene sottomesso.	<a href="#">Event</a>
<a href="#">suspend</a>	The event occurs when the browser is intentionally not getting media data	<a href="#">Event</a>
<a href="#">timeupdate</a>	The event occurs when the playing position has changed (like when the user fast forwards to a different point in the media)	<a href="#">Event</a>
<a href="#">toggle</a>	The event occurs when the user opens or closes the <details> element	<a href="#">Event</a>
<a href="#">touchcancel</a>	L'evento occorre quando si interrompe il contatto tra il dito e un certo elemento del documento. <b>Esempio nell'iPhone X:</b> mantenere il dito su un certo elemento e fare swipe per tornare alla home.	<a href="#">TouchEvent</a>
<a href="#">touchend</a>	L'evento occorre quando togliamo il dito dal touch screen, e il dito si trovava sopra un certo elemento	<a href="#">TouchEvent</a>
<a href="#">touchmove</a>	L'evento occorre quando spostiamo il dito mantenendo il contatto col touch screen. Il dito si trova sopra un certo elemento	<a href="#">TouchEvent</a>
<a href="#">touchstart</a>	L'evento occorre quando si pone un dito sul touch screen, precisamente su un certo elemento.	<a href="#">TouchEvent</a>
<a href="#">transitionend</a>	The event occurs when a CSS transition has completed	<a href="#">TransitionEvent</a>
<a href="#">unload</a>	The event occurs once a page has unloaded (for <body>)	<a href="#">UiEvent</a> , <a href="#">Event</a>
<a href="#">volumechange</a>	The event occurs when the volume of the media has changed (includes setting the volume to "mute")	<a href="#">Event</a>
<a href="#">waiting</a>	The event occurs when the media has paused but is expected to resume (like when the media pauses to buffer more data)	<a href="#">Event</a>
<a href="#">wheel</a>	L'evento occorre quando la rotella del mouse si muove sopra un certo elemento.	<a href="#">WheelEvent</a>

## Esercizio: ingrandimento del carattere in base alla rotella del mouse

### Prima versione:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Ingrandimento/Riduzione carattere</title>
</head>
<body>
  <div id="output" onwheel="carattere(event);" style="text-align:center; font-size:25px;">Hello World</div>
  <script type="text/javascript">
    var output = document.getElementById("output");
    function carattere (e){
      output.innerHTML = "Hello World";

      if(e.deltaY > 0)
        var new_value = parseInt(output.style.fontSize) + 1;
      else if(e.deltaY < 0)
        var new_value = parseInt(output.style.fontSize) - 1;

      output.style.fontSize = new_value + 'px';
    }
  </script>
</body>
</html>
```

### Seconda versione:

```
[...]
  <div id="output" style="text-align:center; font-size:25px;">Hello World</div>
  <script type="text/javascript">
    var output = document.getElementById("output");

    function carattere(e) {
      output.innerHTML = "Hello World";

      if(e.deltaY > 0)
        var new_value = parseInt(output.style.fontSize) + 1;
      else if(e.deltaY < 0)
        var new_value = parseInt(output.style.fontSize) - 1;

      output.style.fontSize = new_value + 'px';
    }

    output.onwheel = carattere;
  </script>
[...]
```

### Terza versione:

```
[...]
  <div id="output" style="text-align:center; font-size:25px;">Hello World</div>
  <script type="text/javascript">
    var output = document.getElementById("output");
    output.addEventListener("wheel", function(e){
      output.innerHTML = "Hello World";

      if(e.deltaY > 0)
        var new_value = parseInt(output.style.fontSize) + 1;
      else if(e.deltaY < 0)
        var new_value = parseInt(output.style.fontSize) - 1;

      output.style.fontSize = new_value + 'px';
    });
  </script>
[...]
```

## Esercizio: cambio di colore con click del mouse e rilascio

```
<!DOCTYPE html>
<html>
<body>
<p id="myP" onmousedown="mouseDown()" onmouseup="mouseUp()">
Click the text! The mouseDown() function is triggered when the mouse button
is pressed down over this paragraph, and sets the color of the text to red.
The mouseUp() function is triggered when the mouse button is released, and
sets the color of the text to green.
</p>

<script>
function mouseDown() {
    document.getElementById("myP").style.color = "red";
}

function mouseUp() {
    document.getElementById("myP").style.color = "green";
}
</script>

</body>
</html>
```

## Esercizio stile Algoritmi & Strutture dati: contaDispari

Si scriva una funzione `contaDispari(T)` che dato un albero binario (i cui nodi sono implementati come visto a lezione come oggetti con chiavi `val`, `sx` e `dx`) restituisca il numero di nodi contenente un valore dispari.

### Esempi<sup>3</sup>:

- `contaDispari({val:7,sx:{val: 4, sx: {val: 3}, dx: {val:12, sx: {val: 4, dx:{val:3}, sx:{val: 8}}}}, dx:{val: 11, dx: {val: 3}, sx: {val:8, sx: {val: 6}}}}) → 5`
- `contaDispari({val:8,sx:{val: -4, sx: {val: 33}, dx: {val:13, sx: {val: 4, dx:{val:-3}, sx:{val: 81}}}}, dx:{val: 11, dx: {val: 3}, sx: {val:8, sx: {val: 63}}}}) → 7`
- `contaDispari({val:8,sx:{val: -4, sx: {val: 32}, dx: {val:12, sx: {val: 2, dx:{val:-2}, sx:{val: 812}}}}, dx:{val: 112, dx: {val: 32}, sx: {val:82, sx: {val: 632}}}}) → 0`

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Conta Dispari</title>
  </head>
  <body>
    [...]
    <script type="text/javascript">
      function contaDispari(T) {
        if(T == null)
          return 0;

        var n_sx = contaDispari(T.sx);
        var n_dx = contaDispari(T.dx);

        return ((T.val%2 != 0) ? 1 : 0) + n_sx + n_dx;
      }
    </script>
  </body>
</html>
```

**Osservazione:** possiamo scrivere funzioni ricorsive e creare strutture dati logicamente simili a quelle viste in C++.

<sup>3</sup> Promemoria: le parentesi graffe rappresentano oggetti: all'interno delle graffe si scrivono le proprietà.

## Esercizio stile Algoritmi & Strutture dati: carriere scolastiche

Si scriva una funzione `gby(data)`, che prende come argomento un array di dizionari (oggetti). Ogni dizionario rappresenta la carriera scolastica di uno studente. In particolare, ogni dizionario ha come chiave il nome di un insegnamento, e come valore il voto preso all'esame (che si assume essere  $\geq 18$ ).

Ad esempio, il seguente array contiene le carriere di 3 studenti<sup>4</sup>:

```
data = [ {lab1:20, fi: 30}, {analisi:28, lab1:30}, {algebraL:28, ProgAlgo:30}]
```

La funzione `gby(data)` restituisce un dizionario avente le cui chiavi sono i nomi degli insegnamenti, e come valore la media dei voti presi da tutti gli studenti per quell'insegnamento.

```
<!DOCTYPE HTML>
<html>
  <head>
    <title>Carriere scolastiche</title>
  </head>
  <body>
    [...]
    <script type="text/javascript">
      var array = gby([ {lab1:20, analisi: 22}, {analisi:28, lab1:30},
{algebraL:28, ProgAlgo:30}]);

      function gby(D) {
        var array = new Array;
        var dati = {};

        for(carriera in D) {
          for(materia in D[carriera]) {
            if(dati[materia] == null)
              dati[materia] = { somma : D[carriera][materia], num : 1};
            else {
              dati[materia].somma += D[carriera][materia];
              dati[materia].num++;
            }
          }
        }

        for(materia in dati) {
          var nuovo_oggetto = {};
          nuovo_oggetto[materia]= dati[materia].somma / dati[materia].num;
          array.push(nuovo_oggetto);
        }

        return array;
      }
    </script>
  </body>
</html>
```

---

<sup>4</sup> Promemoria: le parentesi quadre rappresentano array.