Michele Arcadipane

JAVASCRIPT

GUIDA DI RIFERIMENTO

arcadp@neomedia.it
arcadp@inwind.it
arcadp@tiscalinet.it

AVVERTENZE LEGALI

Realizzata con StarOffice 5.2. Il formato PDF è stato realizzato con i metodi descritti nella guida «Creare documenti PDF senza utilizzare Acrobat» reperibile sul sito http://www.arcaonline.cjb.net o http://spazioweb.inwind.it/arcaonline

Prima pubblicazione Palermo - Italia, maggio 2001
Scaricabile dal sito http://www.arcaonline.cjb.net o http://spazioweb.inwind.it/arcaonline

INTRODUZIONE

Questa guida di riferimento nasce da un mia esigenza: comprendere gli scripts di cui Internet è provvida fornitrice ed il loro funzionamento.

Infatti dopo avere scaricato uno script (o sbirciato tra quelli che corredano le varie pagine) utilizzarlo così com'è non è difficile. Ma se si vuole modificarlo per adattarlo alle proprie esigenze bisogna comprendere esattamente ciò che fa ciascuna istruzione, e come lo fa.

Ho quindi scaricato (non ricordo da dove) la Javascript Guide e il Reference Javascript 1.3 di Netscape, nonché il (meno ricco) Jscript Language Reference di Microsoft.

Altra esigenza che si poneva, a questo punto, era capire quali fossero gli elementi del linguaggio (istruzioni, eventi, oggetti, metodi, proprietà, etc.) che funzionano in maniera simile sia sotto Internet Explorer che sotto Netscape Navigator, per cercare di utilizzare solo quelli che possano consentire il funzionamento di uno script sotto l'uno e l'altro.

E ancora: nello scrivere o modificare un programma javascript mi occorreva sapere quali istruzioni ed oggetti fossero propri a questo linguaggio e quale fosse la esatta sintassi, per non confondersi con istruzioni o sintassi di altri linguaggi.

L'ultima esigenza, soddisfatte le altre, era raccogliere i risultati delle varie letture in una sintesi che non mi costringesse a mantenere sottomano oltre mille pagine e che non esigesse, ogni volta, uno sforzo per comprendere in italiano ciò che su quelle pagine è scritto in inglese: lingua che ho autoappreso (male) in una lotta continua, prima con gli archeomanuali di informatica e di computers (Pet Commodore; C 64), poi con i più vari contenuti di Internet.

Ho, pertanto, scritto questo reference: non vi sono gli esempi di cui il Reference di Netscape è zeppo; manca qualche elemento (ma di quelli mancanti -e di altri che, invece, ci sono- non ne ho mai incontrati negli scripts). Per quel che mi riguarda, ha soddisfatto le mie esigenze e mi ha aiutato moltissimo. Spero che serva anche a chi mi legge!

USARE JAVASCRIPT

SCRIPT (TAG)

Normalmente lo javascript va inserito nella parte di testata della pagina, tra i tags <HEAD> ed </HEAD>, in modo che venga caricato subito e quindi sia disponibile per le chiamate successive.
Lo javascript va inserito entro i tag di marcatura:

```
<SCRIPT language="JavaScript">
<!-- //
Istruzioni
// -->
</SCRIPT>
```

L'inclusione tra <!--//' e ' //-->serve a nascondere lo javascript ai vecchi browsers come se fosse un commento

Per specificare un contenuto alternativo per i browser che non supportano Javascript occorre inserire una pagina alternativa tra i tags
<NOSCRIPT> ..... </NOSCRIPT>.

VERSIONI DI JAVASCRIPT

Le versioni di Netscape Navigator precedenti alla 2.0 **non** possono interpretare Javascript

Netscape Navigator 2.0 ed Explorer 3 eseguono istruzioni contenute tra i tag <SCRIPT LANGUAGE="Javascript"> e </SCRIPT> (Javascript versione 1.0).
Pero' ignorano le istruzioni contenute tra i tag <SCRIPT LANGUAGE="Javascript1.1"> e </SCRIPT> e tra <SCRIPT LANGUAGE="Javascript1.2"> e </SCRIPT>
Netscape Navigator 3.0 ed Explorer 4 eseguono istruzioni tra i tag <SCRIPT LANGUAGE="Javascript1.1"> e </SCRIPT> ed ignorano le istruzioni tra i tag <SCRIPT LANGUAGE="Javascript1.2"> e </SCRIPT>
Netscape Navigator 4.0 ed Explorer 5 (o forse anche il 4.1) eseguono tutte le versioni di Javascript fino alla 1.2 (<SCRIPT LANGUAGE="Javascript1.2"> e </SCRIPT>)

Per sapere quale versione di browser leggera' la pagina HTML:

```
<SCRIPT LANGUAGE="Javascript">
    if (navigator.userAgent.indexOf ("4.0" != -1)
        jsVersion = "1.2";
    else
        if (navigator.userAgent.indexOf ("3.0" != -1)
            jsVersion = "1.1";
        else
            jsVersion = "1.0";
</SCRIPT>
```

FILE .JS

Il sorgente javascript, anziché dentro il file HTML, può essere contenuto in un file ASCII che avrà estensione ".js" e che **non dovrà contenere** i tag di apertura e chiusura script o altri tags HTML!!
Nell'HTML poi va richiamato con

<SCRIPT src="*nome_del_file.js*">
.....
</SCRIPT>

nome del file js può essere un'URL completa come
"http://www.nomedelsito.it/directory/nomefile.js"
Le istruzioni javascript contenute in una coppia di tag <SCRIPT> </SCRIPT> in cui è inclusa la chiamata ad un sorgente .js, vengono ignorate sino a che tale inclusione non genera un errore.

ESPRESSIONI JAVASCRIPT IN ATTRIBUTI HTML

E' possibile inserire in una istruzione HTML **dove occorre un valore** una espressione letterale (i.e. tra virgolette) javascript inserita tra bracket {} e preceduta dall'ampersand (&) e terminante con un punto e virgola (;), come se fosse un carattere speciale, alla pari di   (spazio hard).
Ad esempio potreste scrivere
HR WIDTH "**&**{larghezzariga}**;** %" ALIGN="CENTER"
per creare una riga la cui altezza, invece di esser predefinita da un numero inserito come valore costante, venga definita attraverso la variabile javascript *larghezzariga*.

VIRGOLETTE ED APICI

I valori letterali sono inclusi tra virgolette ("). Ma se un valore letterale deve contenere a sua volta un valore letterale, per consentire a javascript di distinguerlo occorre racchiuderlo tra apici (').
Ciò in quanto i delimitatori di stringa possono essere sia le virgolette che gli apici, purchè il tipo di delimitatore corrisponda in apertura e chiusura.
Esempio: document.write ("HR **'**&{larghezzariga}; %**'** ALIGN=**'**CENTER**'** ")

ISTRUZIONI: BRACKET E PUNTI E VIRGOLA

Le istruzioni vanno separate l'una dall'altra con punto e virgola (;), indipendentemente dal fatto che possano venir scritte su diverse righe per comodità di lettura del sorgente.
I gruppi di istruzioni, appartenenti, cioè, allo stesso costrutto (funzione, istruzione condizionale, etc.) vanno raggruppate tra bracket (**{ }**).

VARIABILI E TIPI DI DATO

Ogni variabile è connotata da un **nome** che inizia con una lettera o un segno underscore (_), seguito da altre lettere o numeri senza spazi.
Attenzione: javascript è *case sensitive*, per cui vPippo è diversa da vpippo.

Quattro sono i tipi di dati fondamentali:

Numeri

es: 3 - 25 - 1.1467254 .JavaScript supporta sia i numeri interi che a virgola mobile.

Booleani

o valori logici. Questi possono essere di due tipi: true (vero) oppure false (falso). Il valore zero, nullo o stringa vuota è *false*; un qualunque valore diverso da zero o da stringa vuota è *true*.

Stringhe

es: "Questo è un esempio". Le stringhe sono formate da uno o più caratteri di testo racchiusi tra virgolette (“) o tra gli apici (‘), purchè il tipo di delimitatore corrisponda in apertura e chiusura.
.

valore zero

rappresentato dalla parola chiave **null**.

Al contrario di altri linguaggi simili (come C e Java), JavaScript non è un linguaggio tipizzato, ovvero una variabile non deve essere dichiarata e può cambiare il tipo di dati associati in qualsiasi momento. La conversione tra i diversi tipi di dati avviene automaticamente.

Nelle stringhe JavaScript, oltre che ai caratteri alfanumerici, si possono utilizzare una serie di caratteri speciali. Questi caratteri non sono stampabili, ovvero non vengono rappresentati nel layout della pagina. Per inserire questi caratteri, basta aggiungere il backslash (\):

\a	: avvertimento (emette il suono di un campanello)
\b	: back (sposta il cursore indietro di un carattere)
\f	: cambio pagina (indica una nuova pagina alla stampante)
\n	: return (è il ritorno a capo)
\r	: home (sposta il cursore all'inizio della riga)
\t	: tab (sposta il cursore alla tabulazione successiva)
\\	: carattere \ (backslash)
\"	: carattere " ('doppi apici')

DICHIARAZIONE ED AMBITO DI VALIDITÀ DI UNA VARIABILE

Una variabile può essere creata in due modi: o semplicemente identificandola per la prima volta ed assegnandole un valore, ovvero dichiarandola esplicitamente con l'istruzione **var**.

Le variabili interne ad una funzione **devono** essere create con l'istruzione **var**.

Le variabili possono essere

globali

create fuori da una funzione o metodo. Le variabili globali sono disponibili in tutto il documento. E' buona norma di programmazione crearle all'inizio, prima di ogni altra istruzione.
Le variabili globali possono anche essere richiamate anche da una finestra o frame diversa dal contenitore, richiamandola come proprietà dell'oggetto contenitore.
Es. : parent.miavariabile

locali

sono le variabili la cui validità è limitata all'interno della funzione entro la quale sono definite. Non sono accessibili fuori dalla funzione creante.
Tuttavia una variabile globale può essere usata all'interno di una funzione che, peraltro, può anche modificarne il valore.

FUNZIONI

Una funzione è un blocco di istruzioni con un compito definito, identificato con l'istruzione **function** seguita da una coppia di parentesi **()** entro cui possono esser previsti dei parametri.
La funzione può avere degli argomenti (parametri) passati dall'istruzione che la chiama, e può restituire un risultato.
Sintassi: function *nomefunzione* **(***arg0* [,*arg2.....,arg255*]**)**

```
{   istruzioni;
    [return valorediritorno;]
}
```

dove *arg1 .... arg255* (255 è il massimo numero dei parametri) sono i parametri passati
In realtà anche la funzione è un oggetto, i cui parametri sono mantenuti nella proprietà *arguments* che è un array numericamente indicizzato. Per cui si può accedere ad un parametro mediante la proprietà *arguments* e l'indice di posizione nella dichiarazione dei parametri

esempio:

```
miafunzione.arguments [3]
miafunzione.arguments.lenght ritorna il numero dei parametri
```

ELEMENTI DEL LINGUAGGIO

OPERATORI

PRECEDENZE TRA GLI OPERATORI

Dalla priorità più alta alla più bassa:

1) Funzione o indice di matrice () []
2) Negazione (!, ~, -), incremento (++), decremento (--)
3) Moltiplicazione, divisione e modulo (*, /, %)
4) Addizione e sottrazione (+, -)
5) Operatore bit (>>, <<, >>>)
6) Confronto (<, >, <=, >=)
7) Uguaglianza e disuguaglianza (==, !=)
8) Operatore relativo ai bit AND (&)
9) Operatore relativo ai bit XOR (^)
10) Operatore relativo ai bit OR (|)
11) Operatore logico AND (&&)
12) Operatore logico OR (||)
13) Operatori condizionali (?, :)
14) Operatori di assegnazione (=)
15) Virgola (,)

La precedenza può essere superata usando le parentesi.

OPERATORI DI ASSEGNAZIONE

=

assegna alla variabile a sinistra il valore dell'espressione di destra
a=b

+=

somma al valore della variabile a sinistra il valore dell'espressione di destra
a+=b equivale ad a=a+b

-=

sottrae al valore della variabile a sinistra il valore dell'espressione di destra

***=**

assegna alla variabile a sinistra il prodotto del valore della variabile stessa per l'espressione di destra
a*=b equivale a a=a*b

/=

assegna alla variabile a sinistra il quoziente del valore della variabile stessa per l'espressione di destra

%=

assegna alla variabile a sinistra il modulo della divisione della variabile stessa per l'espressione di destra

OPERATORI MATEMATICI

+

addizione

++

incremento. Se usato prima della variabile, prima avviene l'incremento e dopo la valutazione del valore. Se usato dopo, prima viene valutato il valore e poi avviene l'incremento.

a++ equivale ad a=a+1

-

sottrazione o meno unario

--

decremento. Se usato prima della variabile, prima avviene il decremento e dopo la valutazione del valore. Se usato dopo, prima viene valutato il valore e poi avviene il decremento.

a - - equivale ad a=a-1

*****

moltiplicazione

/

divisione

%

modulo (il resto intero della divisione di due numeri)

OPERATORI DI STRINGA

+

concatenazione. "pippo"+"pluto" restituisce "pippopluto"

+=

assegna alla variabile di sinistra il risultato della concatenazione della variabile stessa con il contenuto dell'espressione a destra

a="pippo"; a +="pluto" a conterrà "pippopluto"

OPERATORI DI CONFRONTO

Il risultato è *true* (vero) se la condizione espressa dall'operatore si verifica. Altrimenti il risultato è *false* (falso).

==

(doppio uguale) eguaglianza.

===

(tre segni uguale) gli operatori sono eguali e dello stresso tipo

!=

diseguaglianza (Js 1.2)

<

l'operando di sinistra è minore di quello di destra

>

l' operando di sinistra è maggiore di quello di destra

<=

l'operando di sinistra è minore o uguale rispetto a quello di destra

>=

l'operando di sinistra è maggiore o uguale rispetto a quello di destra

OPERATORI LOGICI

&&

AND logico.

||

OR logico

!

NOT logico

OPERATORI SUI BIT

&

AND sui bit

|

OR sui bit

^

XOR sui bit

~

NOT sui bit

>>

SHIFT a destra dei bit dell'operando di sinistra del numero di bit indicati dall'operando di destra. I bits meno significativi sono persi

<<

SHIFT a sinistra dei bit dell'operando di sinistra del numero di bit indicati dall'operando di destra. I bits più significativi sono persi

>>>

SHIFT a destra dei bit dell'operando di sinistra del numero di bit indicati dall'operando di destra. I bits meno significativi sono persi. Gli elementi introdotti sono posti a zero

OPERATORI SPECIALI

?

selettore condizionale.
Sintassi: condizione **?** espresspervero **:** espressperfalso
Se la *condizione* è valutata per *true* ritorna *espresspervero*, altrimenti restituisce *espressperfalso*
esempio:

```
document.write ("Il pollo è " + (zampe==2 ? "bipede" : "non bipede"))
```

,

(virgola). Serve a separare diverse espressioni l'una dall'altra in una posizione che prevede una singola espressione. Tutte le espressioni vengono valutate, ma il valore che viene preso in considerazione è quello dell'ultima espressione.
Esempio:

```
for (a=3, i=1; i<100; i++) .....
```

.

(punto). Serve a richiamare una proprietà o un metodo associato ad un oggetto
esempio:

```
document.write("Chiamo il metodo write dell'oggetto document")
```

delete

(Js.1.2)elimina un oggetto, una proprietà definita dall'utente, un elemento di un array o una variabile dichiarata implicitamente (senza l'uso dell'istruzione **var**).
Sintassi: delete nomeoggetto | oggetto.proprietà | array[ind]
La cancellazione di un elemento di un array non refluisce sugli altri elementi e sulla lunghezza dell'array ritornata da *lenght*. Per cui i restanti elementi successivi continueranno ad avere lo stesso indice di prima. Ma l'elemento

cancellato non esiste più. Se si vuole mantenere l'elemento con valore indefinito, occorre assegnare all'elemento il valore **undefined**

new

crea una nuova istanza di un oggetto predefinito o definito dall'utente.
Sintassi: nomeoggetto=**new** tipooggetto(par1[, par2 ....parx])
Vedi: Oggetti

this

riferisce le operazioni all'oggetto corrente.
esempio: alfa= this.voto anziché alfa=studente.voto
L'oggetto deve essere quello che correntemente viene in uso, altrimenti si riferirà all'oggetto di più alta gerarchia.
Vedi: Oggetti, Eventi

typeof

ritorna una stringa contenente il tipo dell'operando. L'operando può essere anche racchiuso tra parentesi.
Sintassi: typeof *operando* o **typeof(***operando***)**
operando può essere una stringa, una variabile, una keyword o un oggetto
Le stringhe di ritorno possono essere:

object	oggetti e valore *null*
string	
number	
boolean	per *true* e *false*
function	per funzioni, metodi ed oggetti predefiniti
undefined	l'operando non esiste

Per le proprietà viene restituito il tipo di valore contenuto

void

valuta un'espressione senza ritornare alcun valore. L'espressione può anche esser richiusa tra parentesi.
Sintassi: void *espressione* o **void(***espressione***)**
L'uso più frequente è di fornire un valore ad un link di ipertesto senza che venga caricato alcunchè.
esempio

```
<A HREF="javascript: void(0)">Anche se clicchi qui non succede nulla</A>
```


ISTRUZIONI

// oppure /* ..... */

Per inserire un commento usare
// per una singola riga (tutto il testo dopo il doppio slash è commento
/* per iniziare un testo di commento che finirà al */

BREAK

Serve ad uscire da un ciclo in un'istruzione **loop**, **switch** o da una serie di istruzioni contrassegnate da un'**etichetta (label)** e trasferire il controllo all'istruzione immediatamente seguente il loop o al set di istruzioni sotto un'etichetta.

Sintassi: **break** [***etichetta***]

etichetta è opzionale. Se usato, il controllo viene trasferito all'istruzione successiva all'ultima di quelle sotto l'etichetta identificata.

Vedi anche: continue, do ..while, for, switch, **while**

CONTINUE

Serve ad ricominciare un ciclo in un'istruzione di loop, la serie di istruzioni contrassegnate da un'**etichetta (label).**
In un loop di tipo while ritorna a valutare la condizione; in un ciclo for aggiorna il contatore.

Sintassi: **continue** [***etichetta***]

etichetta è opzionale. Se usato, il controllo viene trasferito all'istruzione successiva all'ultima di quelle sotto l'etichetta identificata.

Vedi anche: break, do ..while, for, switch, **while**

DO …WHILE

Esegue le istruzioni tra **do** e **while** finchè la condizione è vera.

Sintassi: do
blocco di istruzioni;
while(condizione**);**

Se la condizione dà *vero* il blocco di istruzioni è rieseguito. Il blocco di istruzioni, comunque, viene eseguito almeno una volta, prima della valutazione della condizione.

N.B. A differenza di questa, nell'istruzione **while** la condizione è valutata all'inizio del blocco di istruzioni, per cui tale blocco non viene eseguito se la condizione è *false*.

EXPORT (Js 1.2)

Serve ad esportare funzioni da un documento ad un altro. Da approfondire

ETICHETTA

Non è una vera istruzione. Javascript dalla versione 1.2 dà la possibilità di porre un'etichetta (**label** in inglese) in un punto dello script per consentire di riferirsi ad un punto del programma e, quindi uscire da quel punto, con un'istruzione **break** ovvero di saltarvi con **continue**.

Sintassi: ***etichetta* :**

etichetta è una qualunque parola, purchè non riservata, da utilizzare come etichetta. Va seguita dai due punti.

N.B. **Attenzione:** l'istruzione **switch** contiene valori identificati come etichetta : (label :). In essa il significato è diverso, in quanto tale 'etichetta' serve afornire un valore di riferimento per eseguire le istruzioni sotto 'etichetta' che matchhanno corrispondenza con l'espressione di selezione.

FOR

Esegue cicli di istruzioni.

Sintassi: **for (***espressione*; *condizione*; *incremento* o *decremento espressione***)**
{
istruzioni;
}

espressione inizializza la variabile di appoggio, che verrà poi incrementata per eseguire il ciclo

condizione viene valutata ad ogni ciclo. Se *true* il ciclo viene rieseguito, altrimenti termina

incremento o decremento serve ad aggiornare l'espressione.

Tipicamente *espressione* inizializza una variabile di appoggio (che resta locale alla funzione in cui il loop è contenuto; *condizione* verifica che la variabile non abbia raggiunto il livello voluto; *incremento o decremento* aggiorna il valore della variabile ad ogni ciclo.

esempio:

```
for (var i= 1; i<=100;i++)
{
    faiqualcosa();
    .....
    altre istruzioni
}
```

FOR … IN

Esegue le istruzioni di ciclo per ciascuna proprietà di un oggetto.

Sintassi: for (*variabile* **in** *oggetto***)**

```
{
    istruzioni;
}
```

variabile è una variabile (contatore) che si incrementa percorrendo tutte le proprietà dell'oggetto

Le istruzioni verranno eseguite con ogni proprietà.

esempio:

Con un oggetto *studente*, le cui proprietà sono *nomeecognome*, *annodinascita*, *luogodinascita*, e *giudizio*

```
var risultato=""
for (var pippo in studente)
{   risultato =+ pippo +"=" + studente[pippo]+";  "
}
```

risultato conterrà
nomeecognome=Michele Arcadipane; annodinascita=1951; luogodinascita=Palermo; giudizio=scarsissimo;

FUNCTION

Dichiara una funzione. Se le funzioni appartengono ad oggetti vengono chiamate metodi.

Sintassi: function *nome***(**[*arg0*] [,*arg2*] .....[,*arg254*]**)**

{ *istruzioni*;
[return *valorediritorno*;]
}

nome	è il nome che si assegna alla funzione o metodo
arg0 ....arg 254	parametri passati alla funzione. I parametri (al massimo 255) sono opzionali, possono essere stringhe, numeri o oggetti
valorediritorno	se si vuole che la funzione restituisca un valore occorre far restituire *valorediritorno* preceduto dall'istruzione **return**

N.B.: La funzione utente da voi creata è un oggetto function e di questo condivide metodi e proprietà

IF …ELSE

La classica istruzione condizionale, con istruzione **else**

Sintassi: if (*condizione***)**
{ *istruzioni*
}
[**else**
{ *istruzioni alternative*
}]

condizione una condizione qualunque che dà come risultato *true* o *false*. Le istruzioni correlate ad **if** vengono eseguite se *condizione* dà vero; altrimenti vengono eseguite quelle correlate ad **else,** se previsto.

Vedi anche: switch

IMPORT (JS 1.2)

Serve ad esportare funzioni da un documento ad un altro. Da approfondire

RETURN

Usato nel corpo di una **function** ritorna il valore (espressione) della funzione

Vedi: function

SWITCH ...CASE (JS 1.2)

Corrisponde ad istruzioni select..case, select ..., etc. di altri linguaggi e consente di operare una scelta di istruzioni da compiere in relazione al valore dell'espressione di selezione.

Sintassi: switch (*espressione***)**

```
{
case espressionecaso1 :
    istruzioni;
    [break;]
case espressionecaso2 :
    altre istruzioni;
    [break;]
.....
.....
[default :
    istruzioni di default;]
}
```

espressione è l'espressione o il valore che deve corrispondere ad uno dei valori *espressionecaso* e viene confrontata con ciascuno di tali valori.

espressionecaso è il valore di selezione. Dopo lo stesso devono esser posti i due punti (:). Solo se viene individuata una corrispondenza verranno eseguite le istruzioni successive a **case** *espressionecaso* **:** In tale ipotesi, però, la valutazione delle altre *espressionecaso* continua a meno che dopo le istruzioni non vi sia un **break**

default : è opzionale e contiene istruzioni per il caso che non venga trovata alcuna corrispondenza

break è opzionale. Ma se non viene messo, dopo l'esecuzione delle istruzioni previste per il caso di corrispondenza l'istruzione **switch** continua con la valutazione delle ulteriori *espressionicaso*.

esempio:

```
var studente = "Arcadipane"
switch (studente)
    {
    case "Rossi" :
        document.write ("Meritevole! <BR>");
        break;
    case "Gates" :
        document.write ("Eccezionale/Deplorevole.<BR>");
        break;
    case "Arcadipane" :
        document.write ("Deplorevole.<BR>");
        break;
    default :
        document.write ("Non in elenco.<BR>)
    }
```


VAR

Dichiara una variabile. Può anche assegnarle un valore iniziale fornito di seguito.

Sintassi: **var** *nomevar* [= *valore*] [**,** *nomevar2* [= *valore*] .....]

nomevar	ogni nome consentito, che inizia con una lettera o un segno underscore (_), seguito da altre lettere o numeri senza spazi. Attenzione: javascript è *case sensitive*, per cui vPippo è diversa da vpippo.
valore	ogni valore consentito (numero, stringa, booleano o *null*).

In javascript le variabili non sono tipizzate, per cui a ciascuna è possibile assegnare ogni tipo di valore.

WHILE

Crea un ciclo condizionale in cui le istruzioni vengono eseguite solo se la condizione è vera.

Sintassi: **while (***condizione***)**
{ *blocco di istruzioni*
}

condizione	Qualunque espressione che dà come risultato *true* o *false*. E' valutata all'inizio del ciclo. Se è vera le istruzioni verranno eseguite, altrimenti il controllo passa all'istruzione successiva al blocco di istruzioni di while.
N.B.	A differenza di questa, nell'istruzione **do ... while** la condizione è valutata solo alla fine del blocco di istruzioni, per cui tale blocco viene eseguito almeno una volta

WITH

Serve ad identificare un oggetto sul quale operare la serie di istruzioni raggruppate nelle bracket {}. In altri termini, nelle istruzioni sotto with non è necessario referenziare ulteriormente l'oggetto su cui with opera.

Sintassi: **with(***oggetto***)**
{ *istruzioni*
.....
}

MODELLI DI RICERCA (REGULAR EXPRESSION)

In Javascript 1.2 sono state introdotte le **regular expressions**, modellate sul Perl. Queste sono modelli che servono a ricercare e confrontare combinazioni di caratteri in una stringa.
Per esempio, se si vogliono cercare tutte le occorrenze di 'geni' in una stringa, occorre creare un modello che consiste in 'geni' e usarlo per trovare le corrispondenze nella stringa.
Il modello di ricerca (la regular expression) può essere creato in due modi:

1) **var** *nomeVarRegExp* **= /*modello*/ [*modificatore*]**
2) **var** *nomeVarRegExp* **= new RegExp ("*modello*" [, "*modificatore*"])**

la differenza essenziale è che se creato con la prima sintassi, il modello viene compilato al caricamento di Javascript e viene mantenuto per qualunque utilizzazione futura. Se, invece, creato con la seconda sintassi, all'oggetto sarà applicabile il metodo **compile**, cosicchè il modello viene ricompilato al volo ogni volta che viene usato. Per cui la sintassi 1 si userà quando *modello* e *modificatore* sono noti in sede di programmazione e non devono cambiare durante l'esecuzione; la seconda quando *modello* o *modificatore* vengono creati da programma o immessi dall'utente.

modello è la combinazione di caratteri (o sottostringa) che si vuole cercare. Nella sintassi 1 va delimitato da caratteri slash (/). Nella sintassi 2 va racchiuso tra virgolette. Quindi, ad es. /geni/ o **"**geni**"**.

modificatore parametro opzionale che può assumere i seguenti valori:

- **g** ricerca globale per tutte le occorrenze di *modello*
- **i** ignora differenze maiuscolo/minuscolo
- **gi** entrambe le opzioni precedenti.

CARATTERI SPECIALI PER LA REGULAR EXPRESSION

La sottostringa usata per *modello* può contenere anche caratteri speciali che servono per particolari direttive di ricerca. I caratteri sono i seguenti:

Carattere	Descrizione
\	è il carattere backslash che viene usato come carattere di escape per dare un particolare significato al carattere che lo segue immediatamente. Ad esempio, **n** corrisponde al carattere n, ma **\n** corrisponde al carattere di nuova riga. Inoltre caratteri speciali possono essere identificati letteralmente se preceduti dal backslash; ad esempio **\n** corrisponde al fine riga, ma **\\n** corrisponde alla sequenza di caratteri \n; + è un carattere speciale, ma se si vuole inserire nella stringa di ricerca il carattere più occorre scriverlo preceduto dal backslash \+
^	la corrispondenza è l'inizio del testo o della linea. Ad esempio il *modello* /^S/ troverà corrispondenza in "Soltanto" ma non in " Lui Solo".
$	la corrispondenza è la fine del testo o della linea. Ad esempio il *modello* /po**$/** troverà corrispondenza in "pippo" ma non in "pollo".

*****	la corrispondenza è il carattere che precede l'asterisco, trovato zero o più volte. Ad es . /zo***/** trova corrispondenza in "z" , in "zo", in "zoo".
+	la corrispondenza è il carattere che precede il segno più, trovato una o più volte. Ad es . /zo***/** trova corrispondenza in "zo" e in "zoo" ma non in "z"
?	la corrispondenza è il carattere che precede il punto interrogativo, trovato zero o una volta. Ad es . /zo***/** trova corrispondenza in "z" e in "zo" ma non in "zoo".
.	il punto equivale ad ogni singolo carattere, eccetto al carattere di nuova riga (\n)
(*modello***)**	la corrispondenza è sempre rispetto alla stringa *modello*. Ma il risultato del confronto (la sottostringa trovata nella stringa oggetto della scansione) viene conservato negli elementi [0] .....[N] dell'array che viene creato con **exec** o **match** o nelle proprietà **$1 ...$9** dell'oggetto predefinito **RegExp**.
|	equivale ad un or, in quanto la corrispondenza viene cercata o con il testo prima o con il testo dopo il carattere | . Ad esempio /pi|po/ troverà corrispondenza sia in "pino" che in "tipologia"
{ *n* **}**	*n* è un numero intero positivo. La corrispondenza si ha per una sottostringa formata dal numero di occorrenze *n* del carattere immediatamente precedente. /z**{3}/** trova corrispondenza nelle prime tre z di "zzzzz" ma non in quella di "zoo".
{ *n* **, }**	*n* è un numero intero positivo. La corrispondenza si ha per una sottostringa formata da un numero di occorrenze almeno *n* del carattere immediatamente precedente. /z**{3,}/** trova corrispondenza nelle z di "zzzzz" ma non in "zoo" o "uzzo".
{ *n* **,** *m* **}**	*n* ed *m* sono interi positivi. Il confronto è positivo per un numero di occorrenze non inferiore ad *n* e non superiore ad *m* del carattere immediatamente precedente. /z**{2,3}/** trova corrispondenza nelle z di "zzzzz" e in "uzzo" ma non in "zoo".
[*xyz* **]**	i caratteri tra le parentesi quadre costituiscono un insieme di caratteri; la corrispondenza si ha per uno dei caratteri dell'insieme. /gatt**[**oi**]/** troverà corrispondenza in "gatto" e "gatti" ma non in "gatta" e "gatte"
[^ *xyz* **]**	i caratteri tra le parentesi quadre costituiscono un insieme negativo di caratteri; la corrispondenza si ha per qualunque dei caratteri non inclusi dell'insieme. /gatt**[^**ae**]/** troverà corrispondenza in "gatto", "gatti" e "gattuccio", ma non in "gatta" e "gatte".
[a-c **]**	i caratteri tra le parentesi quadre costituiscono i limiti inferiore e superiore di un range costituente l'insieme di caratteri adiacenti.
[^ a-c **]**	i caratteri tra le parentesi quadre costituiscono i limiti inferiore e superiore di un range costituente l'insieme negativo di caratteri adiacenti.
\b	corrisponde a "fine parola". /o**\b/** corrisponde alla o di "atto", ma non trova corrispondenza in "ottanta".
\B	corrisponde ad un punto di delimitazione che non è "fine parola". /o**\B/** corrisponde alla o di "ottanta", ma non trova corrispondenza in "atto".
\d	corrisponde ad un carattere rappresentante un numero. Equivale a **[0-9]**

\D corrisponde a qualunque carattere ad eccezione di caratteri numerici. Equivale a **[^0-9]**

\f carattere form feed (fine pagina)

\n carattere di nuova riga

\r carattere ritorno carrello

\s qualunque spazio bianco, compresi caratteri.

\S qualunque carattere eccetto spazio, tabulatore, form feed, nuovariga.

\t carattere tabulatore orizzontale

\v carattere tabulatore verticale

\w qualunque carattere alfanumerico compreso l'underscore.

Equivale ad **[**A-Za-z0-9**_]**

\W qualunque carattere **non** alfanumerico. Equivale ad **[**^A-Za-z0-9**_]**

numero numero è un intero positivo. E' un riferimento alla precedente sottostringa trovata e memorizzata, scritta nella precedente *numero* coppia di parentesi. Ad esempio il *modello* è /pipp**([oi])(\s)**paol**2/** e in esso vi sono due coppie di parentesi. Il riferimento **2** opera alla ricorrenza trovata secondo il modello memorizzato **([oi])**. Per cui in complesso il confronto sarà riuscito se verrà trovato "pippo paolo" ovvero "pippi paoli" ma non "pippi paolo", in quanto la memorizzazione determinata dalle parentesi avrà ad oggetto non il modello tra parentesi ma il risultato dell'applicazione di quel modello, e, quindi, la sottostringa trovata che corrispondeva al modello proposto. In altri termini viene memorizzata la sottostringa trovata, non il modello di ricerca.

Attenzione se *numero* è superiore alle coppie di parentesi precedenti, viene interpretato come ottale.

numeroottale corrisponde al carattere ASCII corrispondente al valore di *numeroottale* che è un numero in notazione ottale. Serve per utilizzare nelle espressioni di ricerca anche caratteri ASCII particolari. Tuttavia, dal momento che il modello è ambiguo, potendo sorgere conflitto con il modello *numero* sopra descritto, è meglio usare la notazione esadecimale.

xnumeroesadecimale* corrisponde al carattere ASCII corrispondente al valore di *numeroesadecimale* che è un numero di due cifre in notazione esadecimale preceduto dal carattere **x**. Serve per utilizzare nelle espressioni di ricerca anche caratteri ASCII particolari

UTILIZZO DEI MODELLI DI RICERCA

Le regular expression create dall'utente sono affiancate da un oggetto predefinito **RegExp**, statico, che appartiene al processo javascript in esecuzione, che non può essere creato ma è sempre disponibile per l'uso. Nelle proprietà di tale oggetto predefinito vengono memorizzati i risultati dell'ultima ricerca.
I modelli vengono utilizzati con i metodi di regular expression **test** e **exec** e con i metodi **match, replace, search** e **split** di **String**.

exec(*stringa***)** applica la ricerca del modello definito nella regular expression alla *stringa* oggetto della ricerca. Se la ricerca ha successo ritorna un array e aggiorna le proprietà dell'oggetto **Regular Expression** e dell'oggetto statico predefinito **RegExp**. Se la ricerca non ha successo torna **null**.

test(*stringa***)** applica la ricerca del modello definito nella regular expression alla *stringa* oggetto della ricerca, tornando *true* o *false* a seconda se la ricerca ha successo o meno.

match(*regexpr***)** applica alla stringa referenziata la **regular expression** *regexpr*, tornando un array con i risultati della ricerca, se questa ha avuto successo. L'elemento zero dell'array contiene l'ultima sequenza di caratteri corrispondente al *modello*. Gli elementi da 1 a *array***.lenght** -1, contengono le ricorrenze delle sottostringhe corrispondenti alla parte di *modello identificata* tra parentesi tonde.

Se si vuole una ricerca globale o non case-sensitive occorre formulare la **regular expression** *regexpr* fornendole i modificatori **g** e/o **i**.

replace(*regexpr* **,** *sottostringa***)** ritorna una nuova stringa che è la risultante della stringa referenziata con le occorrenze trovate tramite *regexpr* sostituite dalla *sottostringa*. La stringa referenziata non subisce modifiche.

search(*regexpr***)** applica alla stringa referenziata la regular expression *regexpr*, tornando la posizione nella stringa referenziata (partendo da zero) se ha avuto successo, oppure -1 (*false*)

split(*separatore***)** ritorna un array i cui elementi sono le sottostringhe che nella stringa referenziata sono separate l'una dall'altra dal *separatore*, che può essere un carattere o stringa o una **regular expression** (ovviamente da Js 1.2). Il *separatore* non viene riportato nelle sottostringhe.

I metodi **exec** e **match** sono meno veloci dei metodi **test** e **search**. Per cui è consigliabile usare questi ultimi se serve soltanto sapere se la stringa contiene il modello.

Se non serve accedere alle proprietà di una **regular expression** e non serve riutilizzarla, un metodo sintetico per ottenere un array con le occorrenze trovate è:

mioArray = /*modello*/ **[***modificatore***].exec(***stringa***);**

OGGETTI

Un oggetto javascript è un costrutto che possiede proprietà (variabili semplici, array, variabili oggetto) e metodi (funzioni).
Javascript possiede oggetti predefiniti ed oggetti definibili dall'utente.
Nella strutturazione del linguaggio javascript, in realtà un oggetto è un array (vettore); le proprietà sono contenute in elementi di tale array, i cui indici, anziché numerici come in una variabile array, sono letterali, cioè identificati con il nome della proprietà. (In realtà nelle precedenti versioni di javascript si poteva accedere a proprietà dell'oggetto, sia con l'indice *nomeproprietà* sia usando l'indice numerico identificante la posizione della proprietà nell'array oggetto. Con js 1.2 tale possibilità è esclusa. Ed è buona norma, quindi, riferirsi alle proprietà solo con il nome, anche per futura compatibilità
I metodi sono dei blocchi di istruzioni identificati in una funzione definita come **function** esternamente alla dichiarazione dell'oggetto; nella dichiarazione dell'oggetto occorrerà associare il nome del metodo con la **function**.

TIPI DI OGGETTO E VARIABILI OGGETTO

Non bisogna confondere l'oggetto come tipo, dal reale oggetto da manipolare (variabile oggetto)
Il tipo di oggetto o esiste già, in quanto predefinito, o va strutturato dall'utente prima di usarne le caratteristiche.
Il tipo di oggetto è definito dall'utente tramite il costruttore, una funzione (**function**) il cui compito è solo quello di definire proprietà e metodi.

Per usare un oggetto, occorre prima creare un'istanza (un oggetto reale), assegnando ad un nome di variabile (oggetto) un tipo di oggetto. Solo dopo si può operare sull'oggetto reale (variabile oggetto) identificato con il nome assegnato.

CREAZIONE DI UN OGGETTO PERSONALIZZATO

Occorre prima definire il tipo di oggetto

Sintassi:
function *nometipooggetto* **(***proprietà1,* [*proprietà2, ...proprietàN*]**)**
{ **this.***nomeproprietà1 =proprietà1*
[**this.***nomeproprietà2 = proprietà2*
.....
this.*nomeproprietàN = proprietàN*]
[**this.***nomemetodo1 = nomefunzionedefinita*
.....
this.*nomemetodoN = nomealtrafunzionedefinita*]
}

Questo è il costruttore del tipo che definisce soltanto la struttura dell'oggetto e la maniera di costruirlo.
La parola chiave **this** identifica il corrente oggetto sui cui si opera.
Esempio:

```
function giudizio (tribunale, sezione, istruttore, stato)
{
this.tribunale=tribunale
this. sezione =sezione
this. istruttore = istruttore
this. stato = stato
this.stampainfo= stampainfo            //questo è un metodo
}
```

Una volta "disegnato" il tipo, quando si vorrà costruire un nuovo oggetto si dovrà crearlo da tale tipo con il costruttore **new**:

varoggetto **=** **new** *nometipooggetto***(***proprietà1* [,*proprietà2 ... ,proprietàN*]**)**

In javascript versione 1.2 è stato aggiunto un inizializzatore di oggetti, che consente di creare direttamente una istanza di oggetto senza prima predefinirne il tipo, e senza doverlo creare con **new**. Ciò può servire quando occorre un unico oggetto di quel tipo.
L'oggetto unico viene inizializzato assegnando ad una variabile oggetto un blocco racchiuso tra bracket { } all'interno del quale vengono definite le proprietà cui vengono assegnati relativi valori con i due punti (:)
La **sintassi** è
varoggetto = **{***propr1* **:** *valore1* [**,** *propr2* **:** *valore2 .....*, *proprN* **:** *valoreN*]**}**

AGGIUNGERE PROPRIETÀ E METODI AD UN OGGETTO DEFINITO

Per aggiungere proprietà e/o metodi ad un oggetto già definito, usare la proprietà **prototype**.

*Nometipooggetto***.prototype.***nomenuovaproprietà* = null
*Nometipooggetto***.prototype.***nomenuovometodo = nomealtrafunzione*

La nuova proprietà (o il nuovo metodo) saranno aggiunti anche agli oggetti già creati con un costruttore che non prevedeva prima la proprietà o il metodo.
Assegnando un valore diverso da null ad una proprietà aggiunta tramite la proprietà **prototype** è possibile anche definire il valore di default che tale proprietà assume in tutti gli oggetti già creati e da creare.
I nuovi oggetti creati con il costruttore (che non prevedeva la proprietà poi aggiunta) avranno sì la nuova proprietà, ma il relativo valore non può essere assegnato dal costruttore che non prevedeva la nuova proprietà. Per cui tale nuova proprietà potrà avere il valore di default previsto, ma se si vuole attribuirle un valore occorrerà attribuirlo esternamente al costruttore con un'assegnazione ordinaria
*varoggettoesistente***.***nuovaproprietà = valore*

Ad esempio vogliamo aggiungere all'oggetto **Array** un metodo che restituisca il valore più grande dell'array, dobbiamo dichiarare la funzione:

```
function array_max( )
{
  var i, max = this[0];
  for (i = 1; i < this.length; i++)
  {
     if (max < this[i])
          max = this[i];
  }
  return max;
}
```

aggiungerla al prototipo di classe

```
Array.prototype.max = array_max;
```

dopodichè potremo usarla

```
var mioarray = new Array(1, 2, 3, 4, 5, 6);
var massimo = mioarray.max( );
```

RICHIAMARE METODI E PROPRIETÀ

Metodi e proprietà appartengono ad oggetti. Quindi per richiamarli occorre riferirli ad un oggetto

esempi:

```
miafinestra.close()
varnome=miafinestra.name
miafinestra.status=varmessaggio
miafinestra.miodocumento.miaform.miacasellatesto.value="Ciao"
```

Esistono anche parole chiave che si riferiscono ad un oggetto senza conoscerne il nome, ma considerandone solo la gerarchia (parent, top) o si riferiscono all'oggetto chiamante (this, self).

Tuttavia l'utilizzo del nome dell'oggetto (o degli oggetti di gerarchia superiore) non è necessario per riferire la proprietà o il metodo all'oggetto corrente; basta specificare la discendenza dall'oggetto corrente in avanti.

Vedi: this, with

OGGETTI PREDEFINITI DI JAVASCRIPT

Sono gli oggetti propri del linguaggio, senza riferimento al browser, e servono come involucri per i tipi di dato relativi.

Gli oggetti predefiniti sono **Array, Boolean, Date, function, Math, Number, RegExp** e **String,** oltre **Object**, l'oggetto progenitore di tutti.
Inoltre javascript possiede alcune funzioni e metodi predefiniti, che cioè non appartengono ad oggetti.

ARRAY

L'array è un insieme ordinato di valori, cui riferirsi con il suo nome e con il relativo indice posto tra parentesi quadre.
E' la struttura cardine di javascript, in quanto tutti i dati strutturati sono in realtà degli array.
Per costruire un nuovo array vi sono tre modi:

var *mioarray***= new Array()**

crea un array vuoto di grandezza indefinita

var *mioarray***= new Array(***dimensione***)**

crea un array di grandezza *dimensione*, che deve essere uguale al numero di elementi + 1 (ma partendo da zero). Quindi *dimensione* sarà un numero uguale al numero reale degli elementi.

var *mioarray***= new Array(***elemento0* [**, ...***elementoN*]**)**

crea un array i cui elementi assumono direttamente i valori dei parametri passati (e separati tra loro da virgole). La grandezza dell'array è data dal numero degli elementi creati, e la relativa dimensione sarà uguale ad *elementoN* +1.

Non è possibile creare in tale maniera un array con un solo elemento numerico, poiché il numero verrebbe interpretato come *numeroelementi*.

E' possibile costruire array multidimensionali, e cioè array i cui elementi sono a loro volta array.
Per riferirsi ad un elemento dell'array occorre richiamarne l'indice tra parentesi quadre, indicando o il numero della posizione nell'array ovvero utilizzando come indice il valore assegnato all'elemento.
Ad esempio mioarray [4] ovvero mioarray ["blu"].
Ad un elemento in un array multidimensionale si fa riferimento usando tanti indici quante sono le dimensioni. Es.: mioarray [4] [3] [6].

Gli array non sono tipizzati, per cui ciascun elemento può essere di diverso tipo.
Inoltre la lunghezza dell'array non implica che esistano tutti gli elementi tra zero e l'intero rappresentante la lunghezza, in quanto alcuni elementi intermedi possono non esistere neppure.
Infine è possibile estendere la lunghezza di un array attribuendo un valore ad un elemento con un indice che va oltre la attuale dimensione dell'array stesso.

ARRAY: PROPRIETÀ

lenght

è un intero che specifica la lunghezza (numero degli elementi) dell'array + 1, partendo da zero.
Poiché gli elementi dell'array non devono essere necessariamente contigui, non esprime il numero di elementi definiti dell'array.
Impostando **lenght** ad un numero inferiore all'attuale dimensione l'array viene troncato; impostandolo ad un numero superiore l'array viene espanso ed ogni ulteriore elemento ha valore **undefined**, sino a che non viene assegnato un valore.

prototype

ritorna il riferimento al prototipo per la classe dell'oggetto.
Vedi: aggiungere proprietà e metodi ad un oggetto definito

ARRAY: METODI

array1. concat (*array2*)

torna un nuovo array risultante dall'unione di *array1* e *array2*.
Es. *nuovoarray= array1*.**concat(***array2***)**

join (*separatore*)

ritorna una stringa che contiene gli elementi dell'array, convertiti in formato stringa, separati dal carattere *separatore* specificato nel parametro.

Attenzione: il *separatore* sarebbe opzionale. Tuttavia è necessario specificarlo, in quanto, in sua assenza, il motore javascript Netscape utilizza come separatore una virgola, mentre quello di Explorer usa una stringa vuota.

pop()

(Js 1.2) ritorna l'ultimo elemento dell'array, rimuovendolo dall'array stesso modificando la dimensione dell'array

push(*elem1* [,....*elemN*])

(Js 1.2) il contrario di **pop()**. Aggiunge gli elementi indicati come parametri alla fine dell'array, maggiorandone la dimensione. Non è chiaro se ritorni l'ultimo elemento aggiunto all'array (l'*elemN*) o se torni la dimensione dell'array. Nell'incertezza non usare il valore di ritorno, ma ottenere le informazioni volute in altro modo.

reverse()

inverte gli elementi dell'array che chiama il metodo. Non costruisce un nuovo array ma inverte i relativi elementi (l'ultimo diviene il primo e viceversa).
Se l'array non ha elementi contigui, vengono creati elementi vuoti per riempire l'intervallo, con valore **undefined**.

shift()

(Js 1.2) Il complemento di **pop().** Ritorna il primo elemento dell'array, rimuovendolo dall'array stesso modificando la dimensione dell'array

slice(*inizio* [*fine*])

ritorna un nuovo array costituito dagli elementi compresi tra *inizio* e *fine*-1.
I parametri indicano la posizione indice, partendo da zero.
L'elemento posto a *fine* non è ricompreso.
Il parametro *fine* può essere omesso. In tal caso ritornerà gli elementi da *inizio* alla fine dell'array.
Se il parametro *fine* è un numero negativo indica lo scostamento dalla fine; saranno tornati gli elementi tra *inizio* e (*array*.**lenght** - *fine*).

sort([*funzioneSort*])

riordina l'array utilizzando la funzione utente *funzioneSort* che va scritta a parte.
Se *funzioneSort* è omesso, l'array verrà ordinato nell'ordine ascendente dei caratteri ASCII.
funzioneSort deve tornare uno dei seguenti valori:

numero negativo	se il primo parametro è minore del secondo
numero positivo	se il primo parametro è maggiore del secondo
zero	se i due parametri sono eguali

Esempio:

```
function sortcrescente (a,b)
{
if  (a<b) { return -1}
if (a>b)  { return 1}
return 0
}
```

Attenzione: per array non contigui il risultato di **sort** può essere diverso a seconda delle versioni javascript e delle piattaforme di utilizzo. Per cui è consigliabile applicare il metodo ad array contigui.

toString(...)

il metodo ha effetti assolutamente diversi a seconda della versione Javascript e il motore (netscape o explorer). Meglio non utilizzarlo, anche perché si tratta di un oggetto ereditato dal prototipo **object** e dovrebbe avere un utilizzazione interna al motore javascript.
Utilizzare, eventualmente il metodo **join** se si vuole ottenere la stringa contenente gli elementi dell'array.

BOOLEAN

E' solo un involucro per il tipo di dati booleano.

DATE

Consente di lavorare con date e orari.

Sintassi

var *miaData* = **new Date()**

crea un oggetto con la data e orario attuali ricavati dall'orologio del computer locale

var *miaData* = **new Date(***millisecondi***)**

crea un oggetto con la data e tempo espressi in *millisecondi* a partire dal 1° gennaio 1970 (tempo UTC)

var *miaData* = **new Date(***stringa***)**

crea un oggetto secondo le indicazioni contenute in *stringa*. La *stringa* dev'essere in un formato riconosciuto dal metodo **date.parse()**

var *miaData* = **new Date(***anno, mese, giorno* **[,** *ora* **[,** *minuti* **[,** *secondi* **]]])**

crea un oggetto con data e tempo secondo i parametri numerici passati.

anno deve essere un numero completo (1999 e non 99): *mese* è un numero tra zero ed 11; *giorno* un numero tra 1 e 31. *Ora, minuti, secondi* sono numeri rispettivamente tra 0 e 23, 0 e 59, 0 e 59; se omessi vengono impostati a zero.

L'oggetto **Date** è internamente rappresentato con un numero intero che rappresenta i *millisecondi* rispetto a mezzanotte del 1° gennaio 1970.

Il range di anni esprimibili dall'oggetto è compreso circa 274.000 anni dal 1° genn. 1970. Nelle versioni javascript dalla 1.3 il numero, se negativo, può esprimere una data precedente al 1 gennaio 1970. In quelle precedenti non è possibile.

DATE:METODI

getDate()

ritorna il giorno (1-31) dell'oggetto cui è applicato, usando il tempo locale.

Esempio

```
var oggi= new Date ();
var giorno =oggi.getDate()
```

getDay()

ritorna un numero (0-6) che rappresenta il giorno della settimana, a partire da domenica, dell'oggetto cui è applicato, usando il tempo locale.

getFullYear()

ritorna l'anno di 4 numeri dell'oggetto cui è applicato, usando il tempo locale.

getHours()

ritorna l'ora (0-23) dell'oggetto cui è applicato, usando il tempo locale.

getMilliseconds()

(Js 1.3) ritorna i millisecondi (0-999) dell'oggetto cui è applicato, usando il tempo locale.

getMinutes()

ritorna i minuti (0-59) dell'oggetto cui è applicato, usando il tempo locale

getMonth()

ritorna un numero (0-11 che rappresenta il mese, a partire da Gennaio (=0), dell'oggetto cui è applicato, usando il tempo locale.

getSeconds()

ritorna i secondi (0-59) dell'oggetto cui è applicato, usando il tempo locale

getTime()

ritorna un numero che rappresenta i millisecondi trascorsi dal 1° gennaio 1970, a mezzanotte, e l'oggetto cui è applicato.

GetTimezoneOffset()

ritorna i minuti di differenza tra il tempo locale, sulla macchina in cui è eseguito lo script, e il tempo UTC (Greenwich)

getUTCDate()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCDay()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCFullYear()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCHours()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCMilliseconds()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCMinutes()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCMonth()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getUTCSeconds()

come l'analogo metodo, ma riferito al tempo universale (UTC).

getYear()

ritorna l'anno dell'oggetto cui è applicato, usando il tempo locale. E' un metodo obsoleto e va sostituito con **getFullYear()**, in quanto ritorna solo gli ultimi due numeri dell'anno, e, talvolta, il 2000 viene ritornato come 100

parse(*stringa*)

ritorna il numero di millisecondi intercorrenti tra il 1° gennaio 1970 e la data indicata nella *stringa*.

Il metodo è utile per impostare valori data partendo da date espresse in stringa, con i metodi **set** dell'oggetto **date**.

Il tempo è quello locale, tranne che non si utilizzi nella *stringa* la specifica **GMT** o **UTC**.

La data in *stringa* va espressa con le seguenti regole:

- le date in formato abbreviato possono usare i separatori "/" o "-" e devono essere scritte in formato mese/giorno/anno. Es. 01/20/2000
- le date in formato esteso possono essere scritte con l'anno il mese (per intero o abbreviato) e il giorno (in inglese) in qualunque ordine. Es "Jan 20, 2000 07:30:00", "January 20, 2000", "2000 Jan 20". Come separatori possono essere usati virgole e spazi, anche più d'uno. Il giorno della settimana (in inglese) può essere espresso, ma se non coincide effettivamente viene ignorato. I mesi vanno espressi con almeno due caratteri, ma se vi può essere incertezza viene considerato il valore più alto (ad es "Ju" viene considerato come "July" e non come "June".
- ore, minuti e secondi vanno separati da due punti; non è necessario che vi siano tutti. Sono orari validi "12:" "12:30" "12:30:56".
- Può essere usato il formato orario in 12 ore AM/PM. Se usato il suffisso AM o PM con un orario superiore a 12 viene generato un errore.
- Una stringa con una data non valida (ad es. due mesi o due anni) genera un errore.

metodi date.set

impostano i valori di data e ora. Se come valori dei parametri vengono forniti valori maggiori dell'ammissibile (ad es. 32 come giorno, o 26 come ora) la data viene incrementata coerentemente ai valori forniti. Ad esempio se viene settato il giorno 32 di gennaio, la data verrà settata al primo febbraio.

setDate(*numero*)

imposta il giorno dell'oggetto data con il *numero*(1-31).

setFullYear(*anno* [, *nummese* [,*numgiorno*]])

imposta l'*anno* (espresso in 4 cifre). Può impostare il mese e il giorno se vengono forniti i parametri. La data è riferita al tempo locale

setHours(*ora* [, *minuti* [,*secondi* [, *millisec*]]])

imposta l'ora (tempo locale). *minuti*, *secondi* e *millisec*ondi sono parametri opzionali, e in versioni precedenti alla 1.3 potrebbero non avere esito.

setMilliseconds(*millisec*)

setMinutes(*minuti* [,*secondi* [, *millisec*]])

imposta i *minuti* (tempo locale). *secondi* e *millisec*ondi sono parametri opzionali, e in versioni precedenti alla 1.3 potrebbero non avere esito.

setMonth(*numMese* [, *giorno*])

imposta il mese (numero compreso tra 0-11); *giorno* (1-31) è un parametro opzionale, e in versioni precedenti alla 1.3 potrebbe non avere esito.

setSeconds(*secondi* [, *millisec*])

imposta i *secondi* (tempo locale). *millisec*ondi è un parametro opzionale, e in versioni precedenti alla 1.3 potrebbe non avere esito.

setTime(*millisecTotali*)

imposta il tempo locale (data e orario) in base a *millisecTotali* che rappresenta il numero di millisecondi trascorsi dalla mezzanotte del 1° gennaio 1970.

setUTCDate(*numero*)

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCFullYear(*anno* [, *nummese* [,*numgiorno*]])

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCHours(*ora* [, *minuti* [,*secondi* [, *millisec*]]])

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCMilliseconds(*millisec)*

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCMinutes(*minuti* [,*secondi* [, *millisec*]])

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCMonth(*numMese* [, *giorno*]))

come l'analogo metodo, ma riferito al tempo universale (UTC).

setUTCSeconds(*secondi* [, *millisec*])

come l'analogo metodo, ma riferito al tempo universale (UTC).

setYear(*anno*)

imposta l'anno in base ad *anno*. Metodo obsoleto da rimpiazzare con **setFullyear**

toGMTString()

ritorna una stringa contenente la rappresentazione in convenzione GMT di un oggetto **date** (che internamente è rappresentato in numero di millisecondi dal 1° genn. 1970). Il formato della stringa è del tipo "09 Jun 1999 00:00:00 GMT"

toLocaleString()

ritorna una stringa contenente la rappresentazione di un oggetto **date** (che internamente è rappresentato in numero di millisecondi dal 1° genn. 1970) secondo le convenzioni derivanti dall'impostazione internazionale del computer locale. Il formato di ritorno è diverso in dipendenza dell'impostazione internazionale del computer

toUTCString()

ritorna una stringa contenente la rappresentazione in convenzione UTC di un oggetto **date** (che internamente è rappresentato in numero di millisecondi dal 1° genn. 1970). Il formato della stringa dipende dalla piattaforma

date.UTC(*anno, mese, giorno* [, *ora* [, *min* [, *sec* [, *millisec.*]]]])

ritorna il numero di millisecondi tra mezzanotte del primo gennaio 1970 e la data espressa dai parametri. I parametri sono numeri interi. In particolare *mese* è un numero da 0 ad 11 (Gennaio- Dicembre).
E' un metodo statico, che non richiede la preventiva creazione di un oggetto **date** per essere usato, ma va chiamato come **date.UTC(.....)**.

In buona sostanza è simile al costruttore ma che usa il tempo UTC (a differenza del costruttore **new date** che usa il tempo locale) e ritorna un numero anziché un oggetto.

FUNCTION

Le funzioni in javascript, nella loro essenza considerate istruzioni definite dall'utente, sono strutturate come oggetti.
La creazione di tali oggetti normalmente viene effettuata scrivendo le funzioni con **function** *nomefunzione* **(***parametri***) {** ... *istruzioni* .....**}**.
Ma può avvenire anche con il costruttore
var *nomefunzione* new function ([*arg1* **[,** *argN***],]** *corpoFunzione***)**
dove *corpoFunzione* è una stringa contenete il corpo della funzione ed i parametri *arg1* ... *argN* sono i nomi dei parametri passati alla funzione, così come la funzione li individua nel suo corpo.
E', comunque, meglio dichiarare le funzioni piuttosto che creare oggetti funzione, in quanto questi vengono valutati ogni volta che la funzione sottostante è chiamata, con ovvio rallentamento dell'esecuzione, mentre le funzioni dichiarate vengono compilate una sola volta, all'inizio, qualunque sia il numero di volte che vengono utilizzate.

FUNCTION:PROPRIETÀ

arguments

è un array i cui elementi contengono i parametri passati alla funzione. Gli elementi sono accessibili con l'indice che identifica la posizione dell'elemento nell'array secondo l'ordine in cui i parametri appaiono tra le parentesi.
All'array **arguments** è possibile accedere soltanto nel corpo della funzione, senza bisogno di specificare la referenza con il *nomeFunzione*. Basta, quindi, ad es., **arguments [2]** (che torna il valore del terzo parametro).
Con la proprietà **lenght**, propria di tutti gli array, è possibile determinare in concreto quanti parametri siano stati passati.

MATH

è un oggetto che fornisce costanti e funzioni matematiche e può essere solo usato. Non è possibile creare nuovi oggetti. Va chiamato direttamente, senza referenza ad oggetti. Ad es. :**Math.abs(**100**)**.

MATH:PROPRIETÀ

le proprietà esprimono essenzialmente costanti numeriche.

Math.E	costante di Eulero, base dei logaritmi naturali; circa 2,718
Math.LN2	logaritmo naturale di 2; circa 0,693
Math.LN10	logaritmo naturale di 10; circa 2,302
Math.LOG2E	logaritmo di E base 2; circa 1,442
Math.LOG10E	logaritmo di E base 10; circa 0,434
Math.PI	pi greco; circa 3.141592653589793
Math.SQRT1_2	radice quadrata di 0,5; circa 0,707
Math.SQRT2	radice quadrata di 2; circa 1,414

MATH:METODI

Math.abs(*num*)

ritorna il valore assoluto dell'espressione numerica *num.*

Math.acos(*num*)

ritorna l'arcocoseno in radianti dell'espressione numerica *num.*

Math.asin(*num*)

ritorna l'arcoseno in radianti dell'espressione numerica *num.*

Math.atan(*num*)

ritorna l'arcotangente in radianti dell'espressione numerica *num.*

Math.atan2(y, *x*)

Calcola in coordinate polari l' angolo(theta) di una coppia di coordinate *x* e *y*. **Attenzione**: il primo parametro è la coordinata *y*, mentre *x* è il secondo.

Math.ceil(*num*)

ritorna il numero intero uguale o maggiore dell'espressione numerica *num.*

Math.cos(*num*)

ritorna il coseno dell'espressione numerica *num.*

Math.exp(*num*)

ritorna E (base logaritmi naturali) elevato a *num.*

Math.floor(*num*)

ritorna il numero intero uguale o minore dell'espressione numerica *num.*

Math.log(*num*)

ritorna logaritmo naturale dell'espressione numerica *num.*

Math.max(*num1, num2*)

ritorna il numero maggiore tra *num1* e *num2.*

Math.min(*num1, num2*)

ritorna il numero minore tra *num1* e *num2.*

Math.pow(*base , esponente*)

ritorna l'elevazione a potenza del numero *base* elevato ad *esponente.*

Math.random()

ritorna un numero pseudocasuale compreso tra 0 e 1 incluso.

Math.round(*num*)

ritorna *num* arrotondato per eccesso se la parte frazionaria è uguale o superiore a 0,5, o per difetto se la parte frazionaria è inferiore.

Math.sin(*num*)

ritorna il seno dell'espressione numerica *num.*

Math.sqrt(*num*)

ritorna la radice quadrata dell'espressione numerica *num.*

Math.tan(*num*)

ritorna la tangente dell'espressione numerica *num.*

NUMBER

E' un oggetto involucro per i tipi di dato numerici. Anche se teoricamente sia possibile costruire un proprio oggetto **number**, è difficile che serva far ciò.
Le proprietà sono proprietà della classe e non dell'oggetto in sé.
Il costruttore è
mionumero **= new Number (*valoreNumerico*)**

Con la funzione di top-level **number(*oggetto*)** è possibile trasformare qualunque oggetto in un numero.

NUMBER: PROPRIETÀ

Le proprietà vanno riferite all'oggetto **Number**.
Ad es: *maxNumero* **= Number.MAX_VALUE**

MAX_VALUE

il più grande numero rappresentabile in javascript; approssimativamente 1.79E+308. Il valore immediatamente più grande assume il valore di **Infinity**

MIN_VALUE

il più piccolo numero positivo (superiore a zero) rappresentabile in javascript;. Il valore immediatamente più piccolo assume il valore zero.

NaN

Speciale valore che ha il significato "non è un numero". E' rappresentato come valore letterale **NaN** senza virgolette di delimitazione. Non è comparabile positivamente con alcun valore, neppure con se stesso.
Per verificare se un valore è equivalente a **NaN** usare la funzione di top-level **isNaN(*valore*)**.

NEGATIVE_INFINITY

Speciale valore che rappresenta l'infinito negativo . E' rappresentato come valore letterale **-Infinity** senza virgolette di delimitazione. Tale valore è restituito quando un valore numerico negativo è superiore a
-Number.MAX_VALUE

POSITIVE_INFINITY

Speciale valore che rappresenta l'infinito positivo. E' rappresentato come valore letterale **Infinity** senza virgolette di delimitazione. Tale valore è restituito quando un valore numerico negativo è superiore a **Number.MAX_VALUE**

OBJECT

E' il primitivo tipo di oggetto javascript da cui tutti gli altri discendono. **Tutti gli oggetti javascript ereditano metodi e proprietà di questo oggetto**.

OBJECT:PROPRIETÀ

constructor

contiene il riferimento alla funzione che ha creato l'istanza di quel particolare oggetto (non è una stringa con il valore letterale del nome della funzione!). Ad esempio, se *mioOggetto* è un'istanza dell'oggetto definito dall'utente di tipo g*iudizio*, creata con la funzione

function giudizio(tribunale, sezione, istruttore, stato)

e si vuol sapere se *mioOggetto* è di tipo giudizio, si ricorrerà alla verifica
if (*mioOggetto***.constructor ==** giudizio)
{ .....},
ovvero, per sapere di che tipo è *mioOggetto* si può ricorrere all'istruzione **switch**.

prototype

contiene il riferimento al prototipo per una classe di oggetti. Si utilizza per aggiungere proprietà e/o metodi ad un oggetto già definito o predefinito.
Vedi : Aggiungere proprietà e metodi ad un oggetto definito

OBJECT:METODI

toString( )

ritorna una stringa rappresentante lo specifico oggetto referenziato.
Ogni oggetto possiede tale metodo che viene chiamato automaticamente quando deve essere rappresentato come valore testo ovvero quando è richiamato in una concatenazione di stringhe.
Per default il metodo ritorna "object *nomeDelTipoDiOggetto*". Ma il metodo può essere sovrascritto per oggetti creati dall'utente.
Per sovrascrivere il metodo, creare la funzione che sostituirà **toString()** per il tipo di oggetto (ad es. 'giudizio'), e sostituirla con

giudizio**.prototype.toString =** *miaFunzioneSostitutiva*

Gli oggetti predefiniti e intrinseci di javascript hanno funzioni che sovrascrivono il metodo **toString**.

Array il metodo restituisce una stringa unica che è la risultante di tutti gli elementi dell'array convertiti in stringa, separati da virgola.
Boolean se il valore dell'oggetto è 'vero' ritorna "**true**", altrimenti ritorna "**false**".
Number ritorna la rappresentazione in testo del numero
String ritorna il contenuto della stringa

valueOf()

ritorna il valore primitivo dell'oggetto referenziato. Il metodo può essere sovrascritto per oggetti creati dall'utente, come specificato in **toString**.

REGEXP

E' un tipo di oggetto introdotto in javascript 1.2. Si tratta di modelli di ricerca che consentono di cercare combinazioni di caratteri entro una stringa.
In realtà vi sono due tipi di oggetto, ancorchè vengono considerati come unico, l'oggetto **Regular Expression**, che è quello creato dall'utente e che fornisce il modello di ricerca, e l'oggetto **RegExp** predefinito, unico per ciascuna finestra, che mantiene i risultati dell'ultima ricerca. Ogni finestra, infatti, ha un oggetto **RegExp** predefinito; per cui ogni processo di esecuzione di Javascript ha il suo oggetto, in modo che non vi siano interferenze di esecuzione tra gli script.
Per altri aspetti **Vedi** sopra **Modelli di ricerca (Regular Expression)**.

REGEXP: PROPRIETÀ

poiché vi sono molte differenze tra il motore di Netscape (da 4 in poi) e quello di Explorer (da 4.1 in poi), si riportano solo le proprietà comuni e con medesimo funzionamento.

$1 ...$9

sono proprietà dell'oggetto statico **RegExp** e contengono le nove ultime ricorrenze più recenti trovate nella ricerca e che corrispondono alla parte di modello tra le parentesi. Vanno usate referenziando l'oggetto **RegExp**.
Esempio di utilizzo, in cui vengono usate le proprietà $1 e $2 per invertire due parole:

```
<SCRIPT LANGUAGE =”Javascript1.2”>
<!-- //
var pattern =new RegExp (“(\w+) \s (\w+)”) ;
var stringa = “blu notte”;
var mioArray =  pattern.exec(stringa); // esegue la ricerca
stringa = RegExp.$2 +” “+RegExp.$1;  // stringa ora contiene “notte blu”
// -->
</SCRIPT>
```

index

proprietà dell'oggetto statico **RegExp**, contiene la posizione del carattere (partendo da zero) nella stringa target dove inizia la prima corrispondenza trovata. Va usata referenziando l'oggetto **RegExp**.

Attenzione: tale proprietà nel Reference di Explorer è riferita all'oggetto **RegExp**. Nel Reference di Netscape non c'è traccia di questa proprietà per tale oggetto, ma tale proprietà è riferita all'oggetto **Array** creato con il metodo **exec**.

input

proprietà dell'oggetto statico **RegExp**, contiene la stringa a cui è stata applicata l'ultima ricerca. Va usata referenziando l'oggetto **RegExp**.

lastIndex

proprietà dell'oggetto **Regular Expression** che viene referenziato.
La proprietà, oltre che letta, può essere impostata da programma.
Prima dell'esecuzione di una ricerca il numero ivi contenuto è usato per indicare alla ricerca da quale carattere della stringa target (partendo da zero) inizia la ricerca.
Se il valore di **lastIndex** è maggiore della lunghezza della stringa, i metodi **exec** e **test** falliscono e **lastIndex** viene riportato a zero; se è uguale alla lunghezza della stringa il modello potrà funzionare solo se rappresenta una stringa vuota.
Dopo l'esecuzione della ricerca contiene la posizione (partendo da zero) nella stringa target del primo carattere successivo all'ultima ricorrenza della ricerca trovata, o zero se non è stata trovata alcuna corrispondenza
La proprietà è settata solo se la ricerca è di tipo global (cioè è stato usato il *modificatore* **g** nel definire la regular expression).

source

proprietà di sola lettura dell'oggetto **Regular Expression** che viene referenziato, contenente il modello di ricerca, senza gli slash (/) di delimitazione e senza gli eventuali modificatori (i, g, gi).

REGEXP: METODI

i metodi appartengono agli oggetti creati con le **Regular expressione** non all'oggetto predefinito **RegExp** che non ha metodi.

compile(*modello* [,*modificatori*])

ricompila durante l'esecuzione dello script un oggetto già creato con il costruttore **new** .
Può servire sia per forzare la compilazione subito dopo la creazione in formato interno (più veloce) che si manterrà per tutti gli utilizzi dell'oggetto, sia per cambiare durante l'esecuzione (ad esempio entro un loop) il *modello* di ricerca, per effettuare ricerche di tipo diverso.
modello e *modificatori* sono espressioni stringa. **Vedi** sopra **Modelli di ricerca (Regular Expression)** per maggiori informazioni.

exec(*stringa*)

esegue una ricerca con l'oggetto **Regular Expression** referenziato nella *stringa*. Se la ricerca ha successo ritorna un array e aggiorna le proprietà dell'oggetto **Regular Expression** e dell'oggetto statico predefinito **RegExp**. Se la ricerca non ha successo torna **null**.

test(*stringa*)

applica la ricerca del modello definito nella **Regular Expression** alla *stringa* oggetto della ricerca, tornando *true* o *false* a seconda se la ricerca ha successo o meno. L'oggetto statico predefinito **RegExp** non subisce modifiche.

STRING

Fornisce una serie di metodi per trattare stringhe di testo.
Sintassi:
var *miastringa* **= new String(***valoreStringa***)**
L'oggetto **String** è concettualmente diverso dalla stringa creata direttamente con un valore letterale, come ad esempio **var** miastringa **=”**Ma che bella giornata**”**.
Tuttavia i metodi e la proprietà dell'oggetto **String** possono essere applicati ad una stringa creata con notazione letterale, in quanto questa viene trasformata in un oggetto temporaneo di tipo **String** che poi viene scartato una volta applicato il metodo.
Per cui difficilmente è necessario creare specificamente un oggetto di tipo **String**; verosimilmente ciò potrebbe esser necessario se si vuole aggiungere una particolare proprietà o metodo a tutti gli oggetti **String**, usando la relativa proprietà **prototype**.
Di contro, se si cuole usare il metodo statico **eval**, l'argomento di questo dev'essere una stringa creata con notazione letterale; se si utilizza un oggetto **String** il risultato di ritorno sarà la stringa stessa.

STRING: PROPRIETÀ

lenght

è l'unica proprietà dell'oggetto, e contiene il numero dei caratteri contenuti nella stringa aumentato di 1. Poiché il conteggio parte da zero (primo carattere) l'intero espresso da **lenght** corrisponde alla quantità di caratteri secondo la notazione numerica corrente.

STRING: METODI

String: metodi di formattazione HTML

Una serie di metodi applicati ad una variabile stringa o ad una stringa letterale restituiscono una stringa formattata secondo le direttive HTML, contente gli appropriati TAG.
As es.: **var** *stringaFormattata* **=** “Sommario”**.anchor(**“SOM”**)**
la variabile stringaFormattata conterrà “<A NAME=”SOM”>Sommario</A>”

anchor(*nomeAncora*)

A NAME =”*nomeAncora*”

big()

BIG applicato al testo della stringa referenziata (carattere grande).

bold()

B applicato al testo della stringa referenziata (grassetto).

fixed()

TT applicato al testo della stringa referenziata (font fisso)

fontColor(*colore*)

FONT COLOR="*colore*" applicato al testo della stringa referenziata. c*olore* è una stringa contenente la descrizione del colore secondo la sintassi HTML

fontsize(*nIntero*)

FONT SIZE="*nIntero*" applicato al testo della stringa referenziata.
nIntero è un intero positivo o negativo tra 1 e 7

italics()

I applicato al testo della stringa referenziata (corsivo).

link(*nomeLink*)

A HREF ="*nomeLink*"

small()

SMALL applicato al testo della stringa referenziata (carattere piccolo).

strike()

STRIKE applicato al testo della stringa referenziata (carattere barrato).

sub()

SUB applicato al testo della stringa referenziata (carattere pedice o deponente).

sup()

SUP applicato al testo della stringa referenziata (carattere apice o esponente).

String: altri metodi

charAt(*indice*)

restituisce il carattere che si trova all'interno della stringa referenziata alla posizione *indice*. I caratteri si trovano dalla posizione zero alla posizione **lenght** -1.

charCodeAt(*indice*)

(Js 1.2) restituisce il valore numerico in formato Unicode del carattere che si trova all'interno della stringa referenziata alla posizione *indice*. I caratteri si trovano dalla posizione zero alla posizione **lenght** -1.

concat(*stringa2*)

(Js 1.2) ritorna una stringa che è la risultante della stringa referenziata concatenata alla *stringa2*. Equivale a *stringareferenziata* **+** *stringa2*

fromCharCode(*n1* [, ...*nX*])

(Js 1.2) trasforma i codici *n1* ..... *nX* (codici carattere Unicode) in caratteri ritornandoli in formato stringa.
E' un metodo statico dell'oggetto **String** e non va referenziato con una particolare stringa. Esempio di uso:

var nuovastringa =**String.fromCharCode(**65, 66, 67**);**

indexOf(*sottostringa*, [,*inizio*])

ritorna la posizione (partendo da zero) all'interno della stringa referenziata del carattere da cui inizia la prima occorrenza di *sottostringa*, se trovata, altrimenti torna -1. Se fornito il parametro *inizio*, la ricerca inizia da tale carattere; altrimenti la ricerca parte dal primo carattere (posizione zero).
La ricerca va da sinistra a destra.

lastIndexOf(*sottostringa*, [,*inizioAritroso*])

ritorna la posizione (partendo da zero) all'interno della stringa referenziata del carattere da cui inizia l'ultima occorrenza di *sottostringa*, se trovata, altrimenti torna -1. Se fornito il parametro *inizioAritroso*, la ricerca inizia da tale carattere; altrimenti la ricerca parte dall'ultimo carattere (posizione *stringa*.**lenght** -1).
La ricerca va da destra a sinistra.
Il valore ritornato riflette comunque la posizione a partire dal primo carattere da sinistra.

match(*regexpr*)

(Js 1.2) applica alla stringa referenziata la **regular expression** *regexpr*, tornando un array con i risultati della ricerca, se questa ha avuto successo. L'elemento zero dell'array contiene l'ultima sequenza di caratteri corrispondente al *modello*. Gli elementi da 1 a *array*.**lenght** -1, contengono le ricorrenze delle sottostringhe corrispondenti alla parte di *modello identificata* tra parentesi tonde. L'oggetto statico **RegExp** viene modificato.
Se si vuole una ricerca globale o non case-sensitive occorre formulare la **regular expression** *regexpr* fornendole i modificatori **g** e/o **i**..

replace(*regexpr* , *sottostringa*)

(Js 1.2) ritorna una nuova stringa che è la risultante della stringa referenziata con le occorrenze trovate tramite *regexpr* sostituite dalla *sottostringa*. La stringa referenziata non subisce modifiche.
sottostringa può includere le proprietà **$1** ....**$9** dell'oggetto statico **RegExp**; tale oggetto, inoltre, viene modificato.

search(*regexpr*)

(Js 1.2) applica alla stringa referenziata la **regular expression** *regexpr*, tornando la posizione nella stringa referenziata (partendo da zero) se ha avuto successo, oppure -1 (*false*)

slice(*inizio* [, *fine*])

ritorna una nuova stringa che è la parte della stringa referenziata che inizia dalla posizione *inizio* (partendo da zero) e finisce alla posizione *fine* -1.
Se *fine* è omesso ritorna la sottostringa da *inizio* alla fine della stringa referenziata.
Se *fine* è un numero negativo, la posizione finale è *stringa*.**lenght** -*fine* -1

split(*separatore*)

ritorna un array i cui elementi sono le sottostringhe che nella stringa referenziata sono separate l'una dall'altra dal *separatore*, che può essere un carattere o stringa (o carattere) o una **regular expression** (ovviamente da Js 1.2). Il *separatore* non viene riportato nelle sottostringhe.

substr(*inizio* [,*lunghezza*])

ritorna la sottostringa della stringa referenziata che inizia alla posizione (partendo da zero) *inizio* di dimensione in caratteri *lunghezza*.
Se *lunghezza* è zero o un numero negativo, il risultato sarà una stringa vuota; se *lunghezza* è omesso, viene tornata la sottostringa da *inizio* alla fine della stringa referenziata.

substring(*posizioneA* [, *posizioneB*])

E' un metodo da evitare, ma molto ricorrente negli script che ho incontrato.
In versioni anteriori a javascript 1.2 (o se non viene specificato come LANGUAGE "Javascript1.2") ritorna una nuova stringa che è la parte della stringa referenziata che inizia (partendo da zero) dalla posizione minore tra *posizioneA* e *posizioneB* e finisce alla posizione di valore più alto meno uno.
Se *posizioneB* è omesso viene estratta la sottostringa sino alla fine della stringa referenziata.
Se viene specificato LANGUAGE = "Javascript1.2" e *posizioneA* è superiore a *posizioneB* viene generato un errore di run-time.

toLowerCase()

ritorna una stringa eguale alla stringa referenziata con tutti i caratteri alfabetici in minuscolo. I caratteri non alfabetici restano immutati. La stringa referenziata non viene modificata.

toUpperCase()

ritorna una stringa eguale alla stringa referenziata con tutti i caratteri alfabetici in maiuscolo. I caratteri non alfabetici restano immutati. La stringa referenziata non viene modificata.

FUNZIONI E PROPRIETÀ PREDEFINITE (TOP-LEVEL)

Javascript ha alcune proprietà e funzioni predefinite, che cioè funzionano indipendentemente da un oggetto specifico.
Le cd. proprietà predefinite , in realtà sono dei valori costanti predefiniti

PROPRIETÀ PREDEFINITE DI TOP LEVEL

Infinity

valore numerico che rappresenta infinito

NaN

valore che rappresenta Non-a-Number (non è un numero)

undefined

valore indefinito

FUNZIONI PREDEFINITE DI TOP LEVEL

escape (*stringa*)

ritorna la codifica esadecimale di una stringa in caratteri ISO-latino-1 (ad es. %20 per spazio)

eval *(stringa*)

(Js 1.2) esegue le istruzioni javascript contenute nella *stringa*, che è una stringa creata con notazione letterale (non un oggetto di tipo **String**). *stringa* può contenere espressioni, o anche una serie di istruzioni.. Il valore di ritorno è il risultato della valutazione dell'espressione.
Es. : *miaVar* = **eval(** "2 * 8" **)** A *miaVar* verrà assegnato 16 (il risultato della valutazione dell' espressione in stringa).
Se l'argomento non è una stringa letterale, ritorna l'argomento non modificato.
Nelle versioni precedenti a javascript 1.2, **eval** è un metodo di tutti gli oggetti.

IsNaN(*arg*)

valuta se *arg* è un non-numero. Cioè ritorna *true* se l'argomento fornito non è un numero; se è un numero ritorna *false*.

Number(*oggetto*)

Converte *oggetto* ad un numero, se possibile. Altrimenti ritorna **NaN**.
Se *oggetto* è una data, ritorna un valore dei millisecondi trascorsi dal 1° gennaio 1970. Se *oggetto* è una stringa contenente un numero ritorna il numero rappresentato, altrimenti ritorna **NaN.** *Oggetto* può essere una stringa contenente un valore letterale che esprime il numero (in inglese), come "two", "five", etc.

parseFloat(*stringa*)

esamina *stringa* e ritorna il numero reale che la *stringa* può rappresentare.
Se il primo carattere non può essere convertito ad un elemento di numero reale ritorna **NaN**.
La funzione ferma l'analisi al primo carattere diverso da numero, segno + o -, punto decimale (virgola) o esponente, eliminando tutti gli spazi iniziali e finali presenti

parseInt(*stringa* [,*radice*])

Come **parseFloat**, ma ritorna un numero intero. Se *radice* è specificato, restituisce il numero nel formato richiesto.
radice è un numero che rappresenta la radice (8=ottale; 10=decimale; 16=esadecimale)
Se *radice* non è specificato, il valore di ritorno sarà in ottale se *stringa* comincia con 0 (zero), esadecimale se comincia con 0x, altrimenti decimale

String(*oggetto*)

Converte *oggetto* in una stringa . Se *oggetto* è di tipo date ritorna una stringa completa leggibile anziché la rappresentazione in stringa dei millisecondi trascorsi.

taint (.....)

Da approfondire. Pare che serva a mascherare dati

unescape(*stringa*)

il contrario di **escape**. Ritorna la stringa ASCII del valore esadecimale rappresentato in *stringa*

untaint(...)

il contrario di **taint**

GERARCHIA DEGLI OGGETTI DEL BROWSER

In javascript esistono due oggetti principali: **Window** e **Navigator**.
Tali oggetti contenitore, a loro volta, contengono altri oggetti dipendenti, come nella figura di seguito

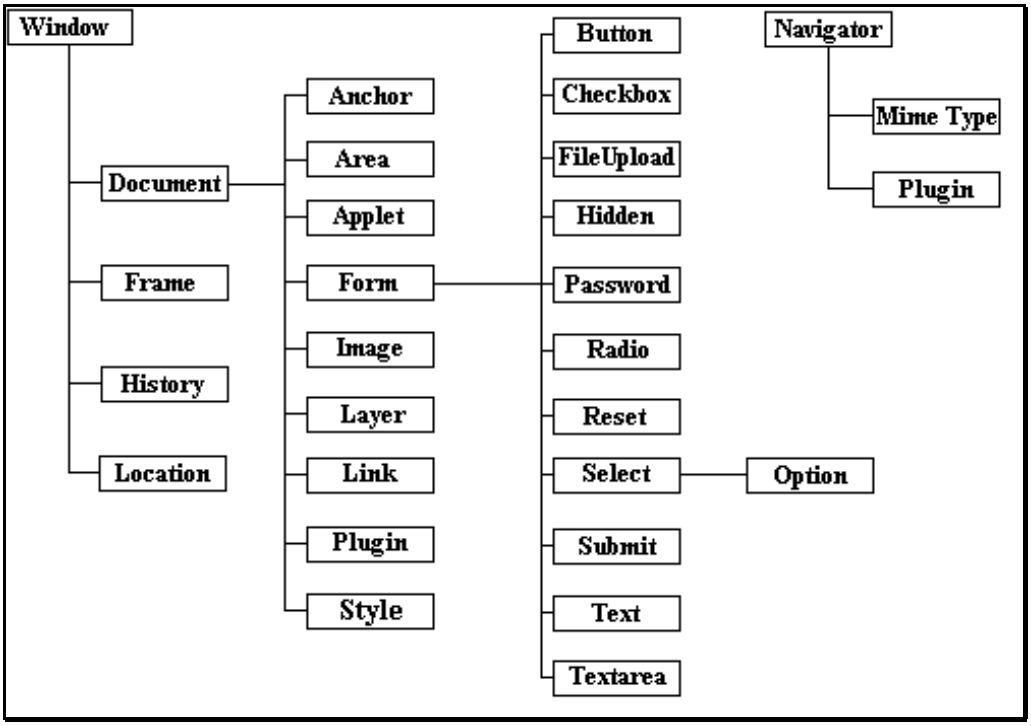

Figura 1 - Gerarchia degli oggetti

Per cui spesso alcuni oggetti si referenziano come proprietà di un oggetto dal quale discendono o a cui appartengono.

OGGETTO WINDOW

E' una finestra del browser o un frame. Il motore javascript crea un oggetto window per ogni entità racchiusa tra i tags <BODY></BODY> e <FRAMESET> </FRAMESET>.
In realtà un frame è un oggetto window dipendente dall'oggetto window principale che lo ospita, ed a cui si riferisce come genitore.

WINDOW: PROPRIETÀ PRINCIPALI

closed

valore booleano. Indica se una finestra è stata chiusa (*true*) o è ancora aperta (*false*)

defaultstatus

la stringa (messaggio) che appare nella barra di stato per default. Da non confondere con la proprietà **status**

document

è l'oggetto **document** corrente.

frames

è un array che contiene tutte gli oggetti **frame** della finestra

history

l'oggetto che contiene le URL visitate da quella finestra

innerHeight

(Js 1.2) specifica la dimensione verticale dell'area visibile della finestra (in pixel)

innerWidth

(Js 1.2) specifica la dimensione orizzontale dell'area visibile della finestra (in pixel)

lenght

restituisce il numero delle frames contenute nella finestra

location

è l'oggetto **location** associato con la finestra

name

una stringa contenente il nome dell'oggetto **window**

opener

punta all'oggetto window che ha aperto la finestra. Una finestra, infatti, può essere aperta da un'altra con il metodo **open**. Il documento chiamante, pertanto, può essere referenziato come
nomewindow**.opener.**proprietà o metodo del chiamante

outerHeight

(Js 1.2) specifica la dimensione verticale dell'area della finestra relativa agli elementi esterni, quali le scrollbars, la riga di stato, i bordi, etc.(in pixel)

outherWidth

(Js 1.2) specifica la dimensione orizzontale dell'area della finestra relativa agli elementi esterni, quali le scrollbars, la riga di stato, i bordi, etc.(in pixel)

parent

è l'oggetto window o frame cui appartiene il corrente oggetto

self

è un riferimento dell'oggetto **window** a se stesso.

status

specifica un messaggio transitorio per la barra di stato

top

è un riferimento all'oggetto window di più alta gerarchia, quello che contiene tutte le frames ed i frameset nidificati

window

è un riferimento a se stesso. Meglio usare **self**

Altre proprietà:

locationbar, menubar, personalbar, scrollbars, statusbar sono proprietà che si riferiscono alle varie parti della finestra esterne all'area dei contenuti. Tutte tali proprietà hanno una loro proprietà **visible** il cui stato (*true* o *false*) determina se la relativa zona è o meno visualizzata. (Js 1.2)

WINDOW: METODI PRINCIPALI

alert(*stringa*)

visualizza un box di dialogo con il messaggio contenuto in *stringa*

back()

(Js 1.2) Torna indietro di una URL dell'history della finestra primaria (top level). Equivale a premere il tasto INDIETRO del browser. Diversamente, il metodo **back** dell'oggetto **history** torna indietro con riferimento alla finestra corrente.

blur()

toglie il focus dall'oggetto specificato

clearInterval(*identificativo*)

(Js 1.2) cancella un l'evento timeout ripetitivo settato con **setInterval** come *identificativo*

clearTimeout(*identificativo*)

cancella un evento timeout non ripetitivo settato con **setTimeout** come *identificativo*

close()

chiude la finestra che chiama il metodo
es: *quellafinestra***.close()**.
Se il metodo è chiamato senza specificare la finestra di riferimento viene chiusa la finestra corrente.
Nella gestione di eventi, per chiudere la finestra occorre usare **window.close()** anziché solo **close()**.

confirm(*stringa*)

visualizza un box di conferma con il messaggio contenuto in *stringa*. Ritorna *true* se l'utente preme OK, altrimenti torna *false*.
Per avere un titolo personalizzato per il box di conferma, occorre costruirsi una finestra di conferma personalizzata, con il metodo **open()**

find(*stringa [casesens, indietro]*)

(Js 1.2) Cerca il testo in *stringa*. Se viene fornita solo *stringa* come parametro effettua una ricerca in avanti, senza badare al formato maiuscole-minuscole.
Se si vuole effettuare una ricerca che tenga conto del formato maiuscole-minuscole ovvero che cerchi indietro, occorre fornire gli altri due parametri. Questi sono valori booleani.

focus()

assegna il focus all'oggetto specificato. E' il contrario di **blur()**.

forward()

(Js 1.2) equivale a **history.forward()**, o ad **history.go(1)** o a premere il tasto indietro del browser. V. anche **back()**

home()

torna all'home page settata nelle preferenze del browser

moveBy(*orizz., vertic.*)

(Js 1.2) sposta la finestra avanti o indietro (num. negativo) del numero di pixel specificato dai parametri

moveTo(*x,y*)

(Js 1.2) sposta l'angolo in alto a sinistra della finestra alle coordinate assolute *x* (da sinistra) e *y* (dall'alto)

open(*url, nomefinestra [,listaopzioni]*)

apre una nuova finestra del browser, identificandola come *nomefinestra*. Il parametro *url* è la url da aprire nella nuova finestra. Se *url* è una stringa vuota, viene creata una finestra vuota.
nomefinestra è una stringa contenente il nome del nuovo oggetto, che servirà ad identificarlo nell'attributo *TARGET*
listaopzioni, è una stringa complessa, contenente l'identificativo dei vari attributi con l'assegnazione (con il segno =) del valore consono che si vuole dare. Gli attributi vanno separati con virgole.
Se non è specificato alcun attributo (e quindi è omessa *listaopzioni*) per default gli attributi booleani sono attivati e le dimensioni e le posizioni sono standard. Se invece viene specificato qualche attributo in *listaopzioni* gli attributi booleani sono settati a disattivato (*no*).
Gli attributi più rilevanti sono:

dependent (Js1.2.) booleano. Se *yes* la finestra è una discendente di quella che l'ha aperta e viene chiusa con quella
height l'altezza in pixel della finestra - es. height=300
innerHeight l'altezza in pixel del contenuto visibile della finestra (alternativo ad **height**)
innerWidth la larghezza in pixel del contenuto visibile della finestra (alternativo ad **width**)
location , menubar, resizable, scrollbars, status, titlebar, toolbar specificano se la finestra ha la zona indirizzi, i menu, se la finestra è ridimensionabile, se ha le scrollbars, la barra di stato, la barra dei titoli, la barra degli strumenti. I valori sono booleani ed attivano (*yes* o *1*) o disattivano (*no* o *0*) l'attributo.
width la larghezza in pixel della finestra

print()

(Js 1.2) stampa il contenuto della finestra

prompt(*messaggio [,contenutoiniziale]*)

apre una finestra di dialogo con il *messaggio* specificato nel parametro stringa, con una casella di input, vuota o con il valore di *contenutoiniziale*, e due pulsanti: OK e Annulla. Ritorna *true* se viene premuto OK, altrimenti *false*.

resizeBy(*orizz., vertic.*)

(Js 1.2) ridimensiona la finestra del numero di pixel indicati nei parametri (aumenta se positivi, restringe se negativi) partendo dall'angolo inferiore destro. E' un metodo che agisce sulle proprietà **outerHeight** ed **outerWidth**

resizeTo(*x,y*)

(Js 1.2) ridimensiona la finestra al valore assoluto *x* ed *y* dei pixel indicati nei parametri partendo dall'angolo superiore sinistro. E' un metodo che agisce sulle proprietà **outerHeight** ed **outerWidth**

scroll(*x,y*)

effettua lo scroll della finestra in modo che l'angolo in alto a sinistra della zona visibile corrisponda alle coordinate *x-y* .Js 1.2 sostituisce questo metodo con **scrollTo**, pur mantenendolo per compatibilità

scrollBy(*orizz., vert.*)

effettua lo scroll della zona visibile della finestra del numero di pixels specificati dai parametri (che vengono aggiunti o sottratti)

scrollTo(*x,y*)

(Js 1.2) effettua lo scroll della finestra in modo che l'angolo in alto a sinistra della zona visibile corrisponda alle coordinate *x-y*

setInterval(*espressione, msec*)

(Js 1.2) valuta ciclicamente un'*espressione* che è una stringa che contiene il nome dell'espressione da valutare. In *espressione* può essere anche passata la chiamata ad una funzione, con i relativi argomenti tra parentesi.
Con la sintassi **setInterval(***nomefunzione, msec* [*,param1 ...,paramN*]**)** (Js 1.2) esegue la funzione *nomefunzione* (non identificata tra virgolette) ogni tanti *msec*, passando alla funzione i parametri *param1 .. paramN* (gli argomenti che la funzione deve ricevere) se necessari.
La differenza tra il passare la funzione nella prima o nella seconda forma è che nell'ultimo caso la funzione viene valutata immediatamente, mentre nel primo allo scadere del tempo fissato.
setInterval genera un evento ripetitivo, il cui identificativo va assegnato ad una variabile che verrà, poi, usata come identificativo per il metodo **clearInterval**.
Es.: vTemporizzatore=**setInterval(“**fSuona()”,5000**)**
clearInterval (vTemporizzatore)

setResizable(*booleano*)

stabilisce se l'utente può ridimensionare (*true*) o no (*false*) la finestra.

setTimeout(*espressione, msec*) e

setTimeout(*nomefunzione, msec [,param1 ...,paramN]*)

(Js 1.2) come **setInterval**, con la differenza che **setTimeout** non stabilisce un evento ciclico, ma programma un singolo evento futuro.

stop()

(Js 1.2) ferma il caricamento in corso. Equivale a premere il pulsante di stop caricamento del browser

ALTRI METODI DELL'OGGETTO WINDOW

trattamento di eventi

Js 1.2 (forse solo Netscape) fornisce agli oggetti window e document dei metodi per l'intercettazione di eventi di altri oggetti contenuti, onde poterli trattare con funzioni scritte dall'utente.
captureEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*]**)**
routeEvent(**Event.***tipoevento***)**, **handleEvent**(**Event.***tipoevento***)** e
releaseEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*]**)**

OGGETTI DI WINDOW

OGGETTO FRAME

L'oggetto frame viene trattato da javascript come un oggetto **window** dipendente dalla finestra che lo contiene. Pertanto ha tutte le proprietà e i metodi dell'oggetto **window**.

OGGETTO DOCUMENT

Contiene informazioni sul documento, cioè su quella parte della finestra o del frame che contiene il documento HTML.

DOCUMENT: PROPRIETÀ PRINCIPALI

alinkColor

è una stringa che contiene il colore del link attivo. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB"

anchors

un array di tutti gli oggetti **anchor** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento.

Es. : *miodocumento*.**anchors[***inizio***]** oppure
miodocumento.**anchors[0]**

applets

un array di tutti gli oggetti **applet** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento

bgColor

è una stringa che contiene il colore dello sfondo. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB"

cookie

stringa contenente le informazioni dei cookies associati al documento

domain

contiene il nome del dominio da cui è caricato il documento

embeds

array contenente un elemento per ciascun oggetto incluso nel documento. Tali oggetti in genere sono collegati a plug-in

fgColor

è una stringa che contiene il colore del testo. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB".

forms

un array di tutti gli oggetti **form** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento.
Inoltre l'oggetto **document** possiede delle **proprietà aggiuntive** che sono riferimenti agli oggetti **form** contenuti, i cui identificativi sono i nomi attribuiti alle form. Ad esempio, se *miodocumento* ha tre forms, la seconda delle quali è stata chiamata *ricerca*, il riferimento a tale form potrà essere:

miodocumento**.forms[**ricerca**]**
miodocumento**.forms[1]**
miodocumento**. Ricerca**

height

(Js 1.2) contiene il valore, in pixel, dell'altezza del documento

images

un array di tutti gli oggetti **image** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento.

lastmodified

stringa contenente la data dell'ultima modifica del documento, ottenuta dalle informazioni del web server (in genere la data dell'FTP sul server). Alcuni server non forniscono tale informazione che resta a 0, per cui la proprietà dà 1° gennaio 1970.

layers

(Js 1.2) un array di tutti gli oggetti **layer** esistenti del documento, nell'ordine decrescente di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa (down-top) nel documento, ma non per essere aggiunti o modificati, anche se le relative proprietà sono accessibili e modificabili.

linkColor

è una stringa che contiene il colore dei links. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB".

links

un array di tutti gli oggetti **link** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento.

plugins

un array di tutti gli oggetti **plugins** del documento, nell'ordine di apparizione. Gli oggetti possono essere referenziati nell'indice dell'array o con il nome del segnalibro o con il numero ordinale corrispondente alla sua comparsa sequenziale (top-down) nel documento.

referrer

contiene la URL del documento attraverso i cui link si è arrivati al documento corrente. In altri termini se al documento corrente si è pervenuti cliccando il link trovato su un altro documento, il documento che si è aperto conterrà nella proprietà **referrer** la URL del documento chiamante.

title

una stringa con il titolo del documento

URL

una stringa con l'URL completa del documento

vlinkColor

è una stringa che contiene il colore dei links già visitati. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB".

width

(Js 1.2) contiene il valore, in pixel, della larghezza del documento

DOCUMENT: METODI PRINCIPALI

close()

chiude il canale di scrittura aperto con il metodo **open()** e determina la visualizzazione dei dati in attesa.

getSelection()

ritorna il testo contenuto nella corrente selezione

open([*tipoMime*,[“replace”]])

Apre un canale di scrittura per ricevere l’output dei metodi **write()** e **writeln()**.
tipoMime è una stringa i cui valori possono essere:

“text/html” il documento contiene testo ASCII formattato con l’HTML
“text/plain” il documento contiene testo ASCII con carattere di fine riga
“image/gif” il documento contiene un’immagine GIF
“image/ipg” il documento contiene un’immagine JPG
“image/x-bitmpap” il documento contiene un’immagine bitmap
“*nomePlugin*” carica il PlugIn specificato e lo usa come destinazione dei metodi **write**

“replace” stringa testuale “replace” da usare per *tipoMime* “text/html”. Se il parametro è usato, determina che l’history del documento precedentemente contenuto nella finestra viene riutilizzata dal documento corrente.

Se nella finestra esiste già un documento, questo viene cancellato, tranne che *tipoMime* sia un plugin.
Il flusso di scrittura, dopo l’invio dei dati con i metodi **write** va poi chiuso con **close()**.

write(*espress1* [,.....,*espressN*])

Scrive una o più espressioni javascript, letterali o numeriche o che danno un risultato letterale o numerico, nel documento corrente aperto per la scrittura con **open()**. A differenza del metodo **writeln()**, non aggiunge un fine riga alla fine dell’output.
Il metodo può essere usato entro i TAG SCRIPT o in un gestore di eventi.
Il gestore di eventi, se non è esplicitamente chiamato il metodo **open()**, apre automaticamente un nuovo documento di tipo “text/html”.
Attenzione a codificare i caratteri speciali per scrivere codice HTML!

writeln(*espress1* [,.....,*espressN*])

Come **write(),** ma aggiunge a fine riga un a capo. L’HTML in genere non tiene conto dei caratteri “a capo”, tranne che in particolari contesti, come tra i tags <PRE> </PRE>.

trattamento di eventi

Js 1.2 (forse solo Netscape) fornisce agli oggetti window e document dei metodi per l’intercettazione di eventi di altri oggetti contenuti, onde poterli trattare con funzioni scritte dall’utente.
captureEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*])
routeEvent(**Event.***tipoevento*), **handleEvent**(**Event.***tipoevento*) e
releaseEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*])

DOCUMENT: PROPRIETÀ E METODI PER GLI STILI

In javascript 1.2 sono disponibili gli stili (ed i fogli di stile) riferiti ai vari tags del documento HTML. Per cui è possibile stabilire, ad esempio, gli attributi (carattere, colore, sfondo, bordo, etc.) da attribuire al testo contrassegnato dai tag <H1> ..</H1>.
L'oggetto **document** ha i seguenti proprietà e metodi collegati con tali possibilità:

classes

rea un oggetto **Style** per connotare uno specifico TAG con gli attributi di una particolare classe di stile.

Sintassi: *miodocumento***.classes.*nomeclasse.nomeTag***

dove *nomeclasse* è il nome dell'attributo di classe e *nomeTag* è il nome del TAG HTML (come H2 o BLOCKQUOTE). Per usare la classe per tutti i tags, usare **all** come *nomeTag*.
Si applica sempre al documento corrente.

ids

crea un oggetto **Style** collegandolo ad un identificativo che potrà essere usato per una specifica ricorrenza di un TAG che lo richiama.

Sintassi: *documento***.ids.***nomeidentificativo*

L'utilizzo è utile quando per una classe si è definito uno stile, ma per una particolare ricorrenza di TAG si vuole usare uno stile diverso. Ad esempio, se si è definito il colore rosso per il TAG H2, si potrebbe volere, in certe circostanze, usare il verde. Pertanto in via generale si definisce la classe per tutti i TAG H2, e si definisce un identificativo che usa il verde, da richiamare all'occorrenza, con qualunque tipo di TAG venga usato. Il codice della definizione sarà:

```
<STYLE TYPE="text/javascript">
classes.Titolo.H2.fontsize="24pt"
classes.Titolo.H2.color="red"
ids.alternativo.color="green"
</STYLE>
```

Il seguente testo apparirà in rosso

```
<H2 CLASS=”Titolo”> Testo rosso</H”>
```

ma quest'altro apparirà in verde

```
<H2 CLASS=”Titolo” ID=”alternativo”> Testo verde</H”>
```

mantenendo gli altri attributi della classe (in questo caso la dimensione 24 punti)

tags

crea un oggetto **Style** per un TAG HTML.

Sintassi: *documento***.tags.***nomeTAG*

contextual(*contesto1, [....contestoN],stileInteressato*)

fornisce un livello di definizione particolare per un oggetto **Style** allorchè si trovi ad operare in un particolare contesto definito da *contesto1*, [*....contestoN*].
Ad esempio, se si vuole che il testo tra i tag <LI> </LI> appaia in verde allorchè sia compreso entro due o più tags <UL> </UL>, il metodo verrà così applicato (entro una definizione di stile con i tags <STYLE> o entro uno script <SCRIPT LANGUAGE= "Javascript 1.2">)

OGGETTO LOCATION

Contiene informazioni sulla corrente URL completa.
E' un oggetto predefinito che costituisce una proprietà dell'oggetto **window** cui si riferisce, ed è accessibile tramite tale proprietà della **window**. Se non è referenziata la **window** di appartenenza, **location** si riferisce alla corrente finestra. Tuttavia nei gestori di eventi occorre referenziarlo con la finestra relativa, altrimenti si riferirà alla proprietà **URL** dell'oggetto **document** corrente.

Principali tipi di protocolli URL

javascript:	codice javascript. Es: javascript:back() Il protocollo javascript valuta l'espressione dopo i due punti e, se ve ne è una carica la pagina definita dall'espressione. Se l'espressione è indefinita (ad esempio javascript:void(0)) non carica nulla. **ATTENZIONE** caricando una nuova pagina su quella contenente lo script si perdono tutte le variabili e funzioni.
http:	world wide web. Es: http://arcaonline.cjb.net
ftp:	file protocol transfert. Es. ftp://spazioweb.inwind.it/arcaonline/downloads
file:	file locali. Es: file:///c:/javascript/motore.html
mailto:	mailing. Es: maito:arcadp@neomedia.it
news:	news. Es: news://news.pippo.com/comp.lang.delphi

LOCATION: PROPRIETÀ

Le proprietà dell'oggetto **location** riflettono le parti della URL

La struttura della URL completa è:
protocol//host:port/pathname#hash?search
protocollo//dominio:porta/percorso#ancora?query

Le seguenti proprietà sono stringhe che riflettono le relative parti della URL.

hash

la stringa comincia con un cancelletto (#)

host

hostname

è la combinazione del ***dominio*** e della ***porta*** **(host + port)**

href

contiene l'intera URL

pathname

port

protocol

search

la stringa comincia con un punto interrogativo (?).

LOCATION: METODI

reload([*forza*])

Ricarica la pagina corrente. Se viene usato il parametro booleano *forza* passando il valore *true* , viene forzato il caricamento della pagina dal server anziché dalla cache (normalmente di default non ricarica la pagina dal server).

replace(*URL*)

Carica la URL sovrascrivendo la corrente voce dell'history. Per cui l'utente non può ricaricare la URL precedente con il tasto back.

OGGETTO HISTORY

E' sostanzialmente un array i cui elementi contengono le URL visitate in quella finestra del browser. Gli elementi sono creati nell'ordine di accesso, per cui history[0] contiene la prima URL visitata e history[history.lenght] contiene l'ultima.

HISTORY: PROPRIETÀ

current

stringa contenente la URL corrente

lenght

il numero degli elementi dell'array history (partendo da zero).

next

stringa contenente la URL successiva a quella corrente, se questa non è l'ultima.

previous

stringa contenente la URL precedente a quella corrente, se questa non è la prima

HISTORY: METODI

back()

carica la URL precedente. Equivale ad history.go(-1).

forward()

carica la URL successiva. Equivale ad history.go(1).

go(*numero | stringa)*

se il parametro fornito è un *numero* (intero, positivo o negativo) carica la URL che è posta nell'elemento precedente (negativo) o seguente quello corrente del *numero* di posti specificato.
Se il parametro e *stringa*, tale stringa deve contenere la URL nella lista o parte di essa. Carica la URL posizionata nell'elemento più vicino a quello corrente che contiene quanto specificato nella *stringa*.

OGGETTI DI DOCUMENT

ANCHOR

Contrassegna una posizione nel documento cui punta un link.
L'ancora è creata con il TAG html A NAME, oppure con un metodo dell'oggetto **string** (*oggettostring*.**anchor(***nomeancora***)**).

ANCHOR:PROPRIETÀ

name

stringa contenente il nome dell'ancora.

text

stringa contenente il testo dell'ancora. E' la parte di testo che nel codice HTML sta tra i TAG <A NAME="*nomeancora*"> ed </A>

x , y

esprimono la posizione in pixel dell'ancora, rispettivamente orizzontale e verticale e dai margini sinistro e alto

APPLET

è un oggetto che include una applet Java nel documento. E' creato con il TAG html APPLET.
L'insieme degli oggetti applet del documento è inserito nella proprietà **applets** dell'oggetto **document**.

AREA

E' un tipo di oggetto **link.** Definisce un'area dell'immagine come mappa per links.
Vedi link.

FORM

Consente all'utente di immettere testo, effettuare scelte tramite checkbox, bottoni radio e listbox ed inviarli per l'elaborazione ad uno javascript ovvero ad un server remoto.
L'oggetto è creato dal TAG html FORM.

Gli oggetti **form** sono accessibili dall'oggetto **document** o tramite la proprietà **forms** dell'oggetto document o tramite le proprietà dell'oggetto document che hanno lo stesso nome delle forms stesse.
Ogni oggetto **form** ha i propri elementi, creati con il codice html. A tali elementi si può accedere o con il relativo nome, o usando la proprietà **elements** di **form** (un array in cui gli elementi prendono posto secondo l'ordine di dichiarazione).

FORM: PROPRIETÀ

action

stringa contenente la URL di destinazione cui vengono inviati i dati della form.
La URL può anche essere il protocollo **javascript:**
La proprietà riflette l'attributo ACTION contenuto nel TAG FORM

elements

un array degli oggetti contenuti nella form (come listbox, bottoni radio, etc.) nell'ordine di dichiarazione.
Ogni elemento dell'array contiene le istruzioni HTML complete per l'elemento stesso.

encoding

stringa contenente la codifica MIME della FORM.
Corrisponde all'attributo ENCTYPE dichiarato nel TAG FORM. Tuttavia tale attributo può essere sovrascritto assegnando altro valore alla proprietà **encoding**

lenght

contiene il numero degli elementi dell'oggetto **form**. Equivale a form.elements.lenght

method

stringa contenente le informazioni di come i dati della form saranno inviati al server. Può assumere i valori "**get**" o "**post**".
Corrisponde all'attributo METHOD dichiarato nel TAG FORM.

name

stringa contenente il nome iniziale della FORM. Può essere sovrascritta.

target

stringa che contiene il nome della finestra cui va l'azione di risposta dopo che i dati della FORM sono stati inviati.
Corrisponde all'attributo TARGET dichiarato con i tags A, AREA e FORM, ma può essere sovrascritta.
Alla proprietà **target** non può essere assegnato un valore corrispondente ad un'espressione javascript o ad una variabile.

FORM: METODI

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

reset()

riporta gli elementi della form al loro valore di default, azzerando tutte le scelte operate dall'utente.
Non è necessario che nella form sia definito un pulsante di reset.

submit()

Invia i dati della form. Equivale al click su un pulsante di tipo submit.
L'invio di dati ad URLs mailto: o news: con il metodo **submit** non avviene, ma l'utente non viene informato della non riuscita. Per sottomettere dati a tali URLs occorre che l'invio avvenga tramite un pulsante di tipo submit.

IMAGE

E' un'immagine sul documento HTML. L'oggetto è creato con il TAG html IMG.
Ma è possibile creare un oggetto **Image** con il costruttore

miaFigura **= new Image([***larghezza***] [,** *altezza***])**

dove *larghezza* e *altezza* sono opzionali valori in pixel.
La creazione di un oggetto con javascript è utile per caricare e decodificare un'immagine prima che debba essere visualizzata. Quindi occorre settare la proprietà **src** dell'immagine posizionata con la corrispondente dell'oggetto creato (*miodocumento***.images[***indice***].src** = *miaFigura***.src**). In tale modo l'immagine viene sì ricaricata, ma dalla cache del browser.
E' anche possibile creare gestori di eventi per ciascun evento dell'oggetto, assegnando alla proprietà correlata all'evento una funzione di gestione degli eventi (**abort, error, keydown, keypress, keyup, onload**).
Esempio: *miaFigura***.onabort =***funzioneGestore*
Gli eventi **click** e **MouseOut** e **MouseOver** non sono gestiti dall'oggetto **Image**. Ma è possibile identificare l'immagine come link (con i TAG html A HREF) o definire un oggetto **area** per l'immagine.

Posizione e dimensione dell'immagine sono determinate dai TAG html e non possono essere modificate con javascript (le proprietà **height** e **width** sono di sola lettura).

IMAGE: PROPRIETÀ

Le seguenti proprietà sono stringhe che corrispondono agli attributi di egual nome del codice html relativo al TAG IMG

border

height

hspace

lowsrc

name

src

vspace

width

Inoltre

complete

è un valore booleano che indica se il browser ha completato il caricamento dell'immagine

IMAGE: METODI

Ha un solo metodo:

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

LAYER

E' un oggetto introdotto in Javascript 1.2 **Netscape**, creato dai TAGs LAYER o ILAYER, o dalla sintassi CSS.
Corrisponde ad uno strato della pagina HTML.

LAYER: PROPRIETÀ

above

l'oggetto layer precedente a questo, in ordine inverso, o la finestra se questo è il primo

background

l'oggetto **Image** usato come sfondo per quello strato. Il valore è **null** se lo strato non ha sfondo

below

l'oggetto layer successivo a questo, in ordine inverso, o **null** se questo è l'ultimo

bgColor

è una stringa che contiene il colore dello sfondo. Può essere specificato con il nome del colore, ovvero in formato RGB, composto da tre coppie di caratteri che identificano il valore esadecimale dei colori nella sequenza "RRGGBB"

clip

è un oggetto che definisce il rettangolo visibile del layer. Le parti eccedenti il rettangolo non sono visibili. Di **Clip** sono accessibili le relative proprietà (in pixel):

bottom	margine inferiore
height	altezza
left	margine sinistro
right	margine destro
top	margine superiore
width	larghezza

document

l'oggetto **document** associato al layer.
Ogni oggetto **layer** contiene il proprio oggetto **document**.. tale oggetto può essere usato per accedere alle immagini, links, ancore e layers contenuti nel corrente layer. Inoltre possono essere usati i metodi dell'oggetto **document** per cambiare i contenuti del corrente layer.

left

la posizione orizzontale in pixel del layer dal margine sinistro del layer genitore (o della finestra se è il primo). Equivale a **layer.x**

name

stringa contenente il nome assegnato al layer dall'attributo ID del tag LAYER.

pageX

posizione orizzontale relativa alla pagina, in pixel.

pageY

posizione verticale relativa alla pagina, in pixel.

parentLayer

l'oggetto **layer** genitore del corrente, o la finestra se il corrente layer non è nidificato.

siblingAbove

l'oggetto layer "fratello" (e cioè che condivide lo stesso layer genitore) precedente a questo, in ordine inverso, o **null** se non vi sono "fratelli" precedenti.

siblingBelow

l'oggetto layer "fratello" (e cioè che condivide lo stesso layer genitore) successivo a questo, in ordine inverso, o **null** se non vi sono "fratelli" successivi.

src

stringa contenente la URL da cui ha origine il contenuto del layer. Corrisponde all'attributo SRC del TAG html LAYER.

top

la posizione verticale in pixel del layer dal margine superiore del layer genitore (o della finestra se è il primo). Equivale a **layer.x**

visibility

determina se il layer è o meno visibile. I valori che può assumere sono:

show	per mostrare il layer
hide	per nascondere il layer
inherit	eredità la visibilità del layer genitore

window

l'oggetto **window** o **frame** che contiene il layer, indipendentemente dal fatto che il layer stesso sia nidificato in altro layer.

x

la posizione orizzontale in pixel del layer dal margine sinistro del layer genitore (o della finestra se è il primo). Equivale a **layer.left**

y

la posizione verticale in pixel del layer dal margine superiore del layer genitore (o della finestra se è il primo). Equivale a **layer.top**

zIndex

La posizione relativa nell'ordine decrescente del corrente layer rispetto ai propri "fratelli".
I "fratelli" con un numero zIndex più basso sono posizionati sotto il corrente layer.

LAYER: METODI

load(*nomeFile, larghezza*)

Cambia l'origine del contenuto del layer, caricandone un altro da un file esterno e ridimensiona la larghezza

nomeFile	è una stringa contenente il nome del file
larghezza	è il valore in pixel cella larghezza da assegnare al contenuto dell'HTML

moveAbove(*layer*)

Immagazzina il corrente layer prima di quello specificato in argomento, senza cambiare le relative posizioni verticali e orizzontali. Dopo tale reimmagazzinamento entrambi i layers condividono lo stesso layer genitore.

moveBelow(*layer*)

Immagazzina il corrente layer dopo quello specificato in argomento, senza cambiare le relative posizioni verticali e orizzontali. Dopo tale reimmagazzinamento entrambi i layers condividono lo stesso layer genitore.

moveBy(*orizz., vertic.*)

Muove il layer dall'attuale posizione di tanti pixel quanto indicano i parametri.

moveTo(*x, y*)

Sposta l'angolo sinistro in alto del layer alle coordinate di schermo espresse da *x* ed *y*, in pixel, all'interno del layer che lo contiene.

moveToAbsolute(*x, y*)

Sposta l'angolo sinistro in alto del layer alle coordinate assolute della pagina espresse da *x* ed *y*, in pixel.

resizeBy(*larghezza, altezza*)

Ridimensiona il layer aggiungendo ai correnti i valori (positivi o negativi) specificati dai parametri (in pixel).
Il metodo non determina la riscrittura dell'HTML contenuto, ma dimensiona soltanto il rettangolo visibile. Per cui parte del testo visualizzato potrebbe sparire se il rettangolo viene ristretto.

resizeTo(*larghezza, altezza*)

Ridimensiona il layer ai valori assoluti specificati dai parametri (in pixel).
Il metodo non determina la riscrittura dell'HTML contenuto, ma dimensiona soltanto il rettangolo visibile. Per cui parte del testo visualizzato potrebbe sparire se il rettangolo viene ristretto.

trattamento di eventi

Js 1.2 fornisce agli oggetti window e document dei metodi per l'intercettazione di eventi di altri oggetti contenuti, onde poterli trattare con funzioni scritte dall'utente.
captureEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*]**)**
handleEvent(**Event.***tipoevento***)**
routeEvent(**Event.***tipoevento***)**, e
releaseEvents(Event.*tipoevento1* [...,**Event.***tipoeventoN*]**)**

LINK

Parte di testo, immagine o area identificati come link di ipertesto.
In html è creato con i TAG A HREF o AREA. In javascript può essere creato con il metodo **link** dell'oggetto **string** *miastringa***.link(***attributoHREF***)**.
Ogni oggetto **link** è un oggetto **location**. Dal momento che il contenuto è un riferimento ad una URL. Per cui il link, oltre che ad una URL del WEB può puntare ad un'azione javascript, se l'attributo HREF ha per destinazione un'entità javascript.
Se si desidera creare un link che non punta a nulla dare come attributo HREF una funzione javascript vuota: <A HREF="javascript:void(0)">.

LINK: PROPRIETÀ

Le seguenti proprietà sono stringhe che riflettono le relative parti della URL.

hash

la stringa comincia con un cancelletto (#)

host

hostname

è la combinazione del ***dominio*** e della ***porta*** **(host + port)**

href

contiene l'intera URL

pathname

port

protocol

search

la stringa comincia con un punto interrogativo (?).

target

può essere riassegnata in ogni momento con un'altra stringa. Non può riferirsi ad espressioni o variabili javascript.

text

(Js 1.2) stringa contenente il testo intercluso tra i tag <A> ed </A>.

x

(Js 1.2) posizione orizzontale del margine sinistro del link rispetto al margine sinistro del documento, in pixel

y

(Js 1.2) posizione verticale del margine superiore del link rispetto al margine superiore del documento, in pixel

LINK: METODI

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

PLUGIN

Un oggetto **plugin** corrisponde ad un plug-in installato. Un plug-in è un modulo software che il browser chiama per trattare particolari dati inclusi nel documento.
Ogni oggetto **plugin** è un array che in ciascun elemento contiene un oggetto **mimeType** supportato da quel plug-in.
Ciascun oggetto **plugin** è un elemento dell'array costituente la proprietà **plugins** dell'oggetto **document.**

PLUGIN:PROPRIETÀ

description

stringa contenente la descrizione del plug-in

filename

il nome del file su disco del plug-in

lenght

il numero di elementi dell'array di oggetti **mimeType**

name

stringa con il nome del plug-in

OGGETTI DI FORM

Gli oggetti di form sono elementi accessibili tramite la proprietà **elements** dell'oggetto **form**. Tale proprietà è un array ciascun elemento del quale corrisponde ad uno degli oggetti contenuti dall'oggetto **form** e contiene il codice HTML completo per quell'elemento. L'indice dell'array **elements** è o il numero assegnato all'elemento, nell'ordine di dichiarazione, ovvero il nome assegnato all'elemento nella dichiarazione html.

button

Corrisponde ad un pulsante nella FORM.
E' creato con il codice html, con il tag <INPUT TYPE="button" ....>.

BUTTON:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("button"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo visualizzato sul bottone,;modificabile.

BUTTON:METODI

blur()

leva il focus dall'elemento

click()

simula il click del mouse sul pulsante, ma non invoca il gestore di eventi onClick

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

checkbox

Corrisponde ad una casella di spunta della FORM, con la quale l'utente può scegliere se attivare o meno un'opzione.
E' creato con il codice html, con il tag <INPUT TYPE="checkbox" ....>.

CHECKBOX: PROPRIETÀ

checked

valore booleano, settabile da programma, che riflette lo stato della casella, *true* se spuntata, altrimenti *false.*

defaultChecked

valore booleano che setta lo stato iniziale della casella, *true* se spuntata, altrimenti *false.*
Riflette l'attributo CHECKED della dichiarazione html della checkbox

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("checkbox"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo visualizzato sul bottone; modificabile.

CHECKBOX:METODI

blur()

leva il focus dall'elemento

click()

simula il click del mouse sul pulsante, ma non invoca il gestore di eventi onClick

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

fileUpload

Corrisponde ad una casella di testo della FORM, nella quale l'utente indica un file locale da inviare come attachement della FORM, provvista di un pulsante "Sfoglia" che consente di navigare tra le directories del computer per individuare il file.
E' creato con il codice html, con il tag <INPUT TYPE="file" ....>.

FILEUPLOAD:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("file"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo visualizzato nella casella di input.

FILEUPLOAD:METODI

blur()

leva il focus dall'elemento

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

select()

seleziona l'area di input della casella, evidenziando il testo contenuto, in modo che l'immissione di nuovo testo da parte dell'utente azzeri il precedente contenuto.

hidden

E' un oggetto di tipo TEXT nascosto, che non viene visualizzato nella FORM e non è modificabile dall'utente, ma il cui valore (proprietà: **value**) può essere settato da programma per passare dati.
E' creato con il codice html, con il tag <INPUT TYPE="hidden" ....>.

HIDDEN:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("hidden"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo associato, modificabile da programma.

password

E' una casella di testo il cui contenuto viene visualizzato nella FORM mascherato da asterischi.
E' creato con il codice html, con il tag <INPUT TYPE="password" ....>.
Nelle versioni di Javascript dalla 1.2 in poi, il contenuto della proprietà **value** è in chiaro e non è associato a protezioni.

PASSWORD:PROPRIETÀ

defaultValue

stringa contenente il valore di default dell'elemento

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("password"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo immesso nella casella di input, ancorchè venga visualizzata con asterischi..

PASSWORD:METODI

blur()

leva il focus dall'elemento

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

select()

seleziona l'area di input della casella, evidenziando il testo contenuto, in modo che l'immissione di nuovo testo da parte dell'utente azzeri il precedente contenuto.

radio

Un oggetto **radio** di una FORM è un insieme di bottoni, uno solo dei quali può risultare selezionato; la selezione di uno di essi deseleziona tutti gli altri.
E' creato con il codice html, con il tag <INPUT TYPE="radio" NAME="stessonome"....>.
Infatti tutti i bottoni radio dello stesso insieme devono avere l'attributo NAME uguale per tutti.
Ciascun bottone radio è posizionato in un array di oggetti sottostante all'oggetto radio identificato con lo stesso nome.
Il riferimento al singolo bottone va fatto usando o l'indice numerico corrispondente al l'ordine di dichiarazione del singolo bottone, o usando come indice il valore (stringa) attribuito all'attributo VALUE del singolo bottone (se tale attributo è stato dato).

RADIO: PROPRIETÀ

checked

(proprietà del singolo bottone) valore booleano, settabile da programma, che riflette lo stato del singolo bottone, *true* se selezionato, altrimenti *false*. Selezionato un bottone, l'analoga proprietà degli altri dell'insieme viene settata a *false*

defaultChecked

(proprietà del singolo bottone) valore booleano che setta lo stato iniziale singolo bottone, *true* se selezionato, altrimenti *false*.
Riflette l'attributo CHECKED della dichiarazione html.

form

(proprietà dell'insieme di bottoni) stringa contenente il nome della **form** cui appartiene

name

(proprietà dell'insieme di bottoni) il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

(proprietà dell'insieme di bottoni) il tipo dell'elemento ("radio"). Corrisponde all'attributo TYPE del TAG html

value

la proprietà **value** riflette l'attributo VALUE della dichiarazione html, ma ha significato e contenuto diverso a seconda se riguarda l'insieme dei bottoni radio o il singolo bottone dell'array dell'insieme.
Se, infatti, nel dichiarare i singoli bottoni (l'insieme non viene dichiarato ma viene costruito dal motore javascript accorpando tutti i bottoni con lo stesso NAME) viene individuato per ciascuno l'attributo VALUE="*nomebottone*", la

proprietà **value** dell'insieme dovrebbe ritornare il *nomebottone* che risulta selezionato dall'utente e, inizialmente, quello dell'ultimo dichiarato.
Invece la proprietà **value** del singolo bottone

*miodocumento.miaform.**nomeOggettoRadio**[**indice**]*.**value**

ritorna la stringa usata con l'attributo VALUE nella dichiarazione html, se l'attributo è stato settato, altrimenti torna "**on**".
La proprietà **value** del singolo

RADIO:METODI

blur()

leva il focus dall'elemento

click()

simula il click del mouse sul singolo bottone, ma non invoca il gestore di eventi onClick

focus()

posiziona il focus sul bottone radio

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

reset

Corrisponde ad un pulsante nella FORM la cui pressione determina l'azzeramento delle scelte utente sugli elementi della form che vengono riportati ai valori di default.
E' creato con il codice html, con il tag <INPUT TYPE="reset" ....>.
Poiché non è possibile evitare che una volta premuto il pulsante si determini l'azzeramento, ove si voglia intercettare l'azzeramento (per consentirlo, ad es. a certe condizioni), occorre usare un pulsante di tipo BUTTON, ed assegnare all'evento **onClick** una funzione appropriata.

RESET:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("reset"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo visualizzato sul bottone di reset,;modificabile.

RESET:METODI

blur()

leva il focus dall'elemento

click()

simula il click del mouse sul pulsante, ma non invoca il gestore di eventi onClick

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

select

Corrisponde ad una lista di selezione nella FORM da cui è possibile selezionare una delle voci della lista.
E' creato con il codice html, con il tag <INPUT TYPE="select" ....>.

SELECT:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

length

il numero degli elementi OPTIONS contenuti nella lista

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

options

è un array ciascun elemento del quale contiene una voce della lista, nell'ordine della relativa dichiarazione (con l'attributo OPTION del TAG html).
E' possibile aggiungere o cancellare elementi di tale array

Oggetti option

Corrispondono alle voci di selezione e possono essere creati o con la dichiarazione html entro l'oggetto **select**, ovvero con il costruttore

miaVoce **= new Option ([** *testo* **[,** *valore***[,** *default* **[,** *stato***]]]])**

dove *testo* è la voce della lista, *valore* è il valore di ritorno in caso di scelta della voce, *default* è lo stato di default iniziale di selezione (*true* o *false*) e *stato* è lo stato corrente di selezione (*true* o *false*).

option: proprietà

defaultSelected lo stato di default iniziale di selezione (*true* o *false*). Inizialmente riflette l'impostazione effettuata con l'attributo SELECTED della dichiarazione html

index la posizione nell'array **select.options** (partendo da 0)

lenght il numero di elementi dell'array **select.options**

selected lo stato corrente di selezione (*true* o *false*). Se l'oggetto **select** è stato creato con la clausola MULTIPLE, per individuare le voci selezionate occorre far riferimento a questa proprietà di tutti gli elementi dell'array **select.options**, per determinare gli elementi selezionati.

text il testo che compone la voce della lista

value il valore che viene ritornato dall'elemento quando lo stesso è stato selezionato.

selectedIndex

è un numero intero corrispondente all'elemento dell'array di **options** selezionato (il primo elemento è 0). Se non è stata selezionata alcuna voce della lista, il valore è -1.
La proprietà è settabile da programma, e in tal caso la visualizzazione della voce corrisponderà al settaggio operato.
Se la dichiarazione html è stata effettuata con la clausola MULTIPLE, occorrerà interrogare la proprietà **select** di ciascun elemento **option** dell'array **options** per verificare quali elementi della selezione multipla siano stati scelti.

type

per gli oggetti **select** creati con la clausola MULTIPLE contiene la stringa "select-multiple"; per quelli creati senza tale clausola la stringa è "select-one".
L'uso della clausola consente di operare la selezione di diverse voci; senza la clausola MULTIPLE, invece, è possibile selezionare una sola voce.

submit

Corrisponde ad un pulsante di tipo SUBMIT nella FORM, cioè un pulsante alla cui pressione viene determinato l'invio dei risultati della form alla URL specificata nella proprietà **form.action**.
E' creato con il codice html, con il tag <INPUT TYPE="submit" ....>.
Poiché non è possibile evitare che una volta premuto il pulsante si determini l'invio dei dati, ove si voglia intercettare l'invio (per consentirlo, ad es. a certe condizioni), occorre usare un pulsante di tipo BUTTON, ed assegnare all'evento **onClick** una funzione appropriata.

SUBMIT:PROPRIETÀ

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("submit"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo visualizzato sul bottone; modificabile.

SUBMIT:METODI

blur()

leva il focus dall'elemento

click()

simula il click del mouse sul pulsante, ma non invoca il gestore di eventi onClick

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

text

E' una casella che consente l'immissione di testo (stringhe e numeri) nella FORM.
E' creato con il codice html, con il tag <INPUT TYPE="text" ....>.

TEXT:PROPRIETÀ

defaultValue

stringa contenente il valore di default dell'elemento, e cioè il contenuto preimpostato. Riflette l'impostazione effettuata con l'attributo VALUE della dichiarazione html

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("text"). Corrisponde all'attributo TYPE del TAG html

value

stringa che contiene il testo della casella di input. Tale testo inizialmente è quello contenuto nella proprietà **defaultValue** (se preimpostato con l'attributo VALUE nella dichiarazione html); poi quello immesso dall'utente. Può anche

essere impostato da programma, in qualunque momento; in tal caso il testo impostato viene visualizzato subito nella casella.

TEXT:METODI

blur()

leva il focus dall'elemento

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

select()

seleziona l'area di input della casella, evidenziando il testo contenuto, in modo che l'immissione di nuovo testo da parte dell'utente azzeri il precedente contenuto.

textarea

E' una casella che consente l'immissione di testo (stringhe e numeri) su più righe nella FORM.
E' creato con il codice html, con il tag <TEXTAREA ....>.
Per iniziare una nuova riga, occorre un carattere di fine riga. In windows tale carattere è la sequenza \r e in UNIX e McIntosh \n. Pertanto, per inserire da programma un fine riga occorre testare la piattaforma in cui lo script opera (proprietà **navigator.appversion**)

TEXTAREA:PROPRIETÀ

defaultValue

stringa contenente il valore di default dell'elemento, e cioè il contenuto preimpostato. Riflette l'impostazione effettuata tra i tag di apertura e chiusura TEXTAREA della dichiarazione html

form

stringa contenente il nome della **form** cui appartiene

name

il nome dell'elemento. Corrisponde all'attributo NAME del TAG html

type

il tipo dell'elemento ("textarea").

value

stringa che contiene il testo della casella. Tale testo inizialmente è quello contenuto nella proprietà **defaultValue** (se preimpostato tra i tag di apertura e chiusura TEXTAREA della dichiarazione html); poi quello immesso dall'utente. Può anche essere impostato da programma, in qualunque momento; in tal caso il testo impostato viene visualizzato subito nella casella.

TEXTAREA:METODI

blur()

leva il focus dall'elemento

focus()

posiziona il focus sul pulsante

handleEvent(*nomeevento*)

Invoca il gestore di eventi per l'evento passato come parametro.

select()

seleziona l'area di input della casella, evidenziando il testo contenuto, in modo che l'immissione di nuovo testo da parte dell'utente azzeri il precedente contenuto.

OGGETTO NAVIGATOR

E' un oggetto creato dal motore javascript che dà informazioni sulla versione del browser in uso.
Le proprietà di **navigator** sono di sola lettura e consentono di conoscere la versione del browser, i tipi MIME che possono essere trattati, i plug-in installati.

NAVIGATOR:PROPRIETÀ

appCodeName

stringa specificante il nome in codice del browser. Per Netscape dà "Mozilla"

appName

stringa che specifica il nome del browser

appVersion

stringa che specifica la versione del browser

mimeTypes

array degli oggetti **mimeType** installati

Oggetto mimeType

E' un oggetto creato dal motore javascript per ciascun tipo di MIME installato.

mimeType:proprietà

- **description** stringa contenente la descrizione del tipo dei dati trattati dal tipo Mime
- **enabledPlugin** contiene il riferimento all'oggetto plug-in configurato per quel tipo di MIME.
- **suffixes** stringa contenente le possibili estensioni (del nome file) per il tipo MIME, separate da virgole
- **type** stringa specificante il nome del tipo MIME, univoco rispetto agli altri tipi MIME

plugins

array degli oggetti **plugin** installati
Per l'oggetto **plugin** vedi tra gli oggetti di **document**.
La proprietà **plugins**, oltre la proprietà **lenght**, in realtà propria di tutti gli array, ha un metodo: **refresh(***valBooleano***)**, che consente di rendere disponibile immediatamente un plug-in appena installato.
Chiamando il metodo **refresh** con un parametro *true*, viene ricostruito l'array **plugins** e vengono ricaricati i documenti aperti; usando *false* l'array viene ricostruito ma i documenti non vengono ricaricati.

NAVIGATOR:METODI

javaEnabled()

ritorna *true* se Java è abilitato, altrimenti *false*

preference(*nomeOpzione*)

è un metodo che ritorna il settaggio della opzione di impostazione del browser identificata da *nomeOpzione*.
Il *nomeOpzione* per sapere se javascript sia (*true*) o meno (*false*) abilitato è javascript.enabled.

EVENTI

Gli eventi permettono un controllo del programma al verificarsi di un'azione da parte dell'utente.
Ad esempio, cliccare su un link, premere un pulsante di una form, passare con il mouse su un link o su un'area genera il corrispondente evento che è intercettato e a cui può seguire un'azione programmata.

GESTIONE DEGLI EVENTI

Normalmente i vari eventi hanno una propria gestione standard. Ad esempio, cliccando su di un link ipertestuale, viene effettuato il tentativo di caricamento della pagina puntata, o raggiunto il documento all'ancora ove punta il link. Ovvero, se viene premuto un pulsante RESET di una FORM, viene azzerata la FORM.
L'evento è intercettabile per essere gestito tramite codice javascript; in altri termini è possibile associare una o più funzioni javascript al verificarsi di un evento, realizzando così un "gestore di evento", che determina un'azione diversa da quella di default.
La gestione di un evento va prevista nella dichiarazione HTML dell'elemento che può determinare l'evento, associando al gestore dell'evento il codice javascript, in tale maniera:

<TAG .... on*Evento *="*codice Javascript*">

Es: Home page

TAG è una qualunque dichiarazione TAG html (ad es. A HREF) di un elemento che può determinare l'*evento*

Evento è l'evento che si vuole gestire. Il gestore di eventi è identificato facendo precedere da **on** il nome dell'evento (ad es. **on**Blur).

codice Javascript va racchiuso tra virgolette. Può essere una singola istruzione; una chiamata di funzione; una serie di istruzioni e/o chiamate di funzioni separate da punti e virgola. Se le istruzioni o parametri di funzione devono essere racchiuse tra virgolette, usare le virgolette (") per delimitare *codice javascript* e gli apici (') al posto delle virgolette nel corpo del codice.

Nota: In javascript gli elementi associati ad un evento sono trattati come oggetti che, in aggiunta alle altre proprietà, possiedono delle proprietà i cui nomi corrispondono ai vari gestori degli eventi associati. Ad esempio, un oggetto di tipo **button** oltre alle altre proprietà ha anche le proprietà **onBlur**, **onClick, onFocus, onMouseDown** e **onMouseUp**. Tali proprietà contengono il **riferimento** al codice gestore; per cui, ad esempio, se si volesse settare diversamente il gestore dell'evento **onFocus** (già associato, nella dichiarazione, alla funzione *alfa()*), occorrerà attribuire al valore dalla proprietà il riferimento alla funzione *alfa2* e non la chiamata alla funzione *alfa2()*.

EVENTI JAVASCRIPT

ABORT

Si verifica quando si interrompe il caricamento di una immagine (Js 1.2)
Associato a: **immagini**

BLUR

Si verifica quando un oggetto sulla pagina perde il *focus*, ovvero quando viene deselezionato
Associato a: **finestre, frames** e tutti gli elementi di una **form**

CHANGE

Si verifica quando un campo di testo di un FORM viene modificato
Associato a: elementi **text, textarea** e **select** di una **form**

CLICK

Si verifica quando si fa clic su un elemento o un link.
Per gli elementi di una form, se il valore di ritorno del gestore dell'evento ritorna *false* l'azione associata ordinariamente con l'evento **Click** non viene effettuata.
Associato a: **link**, e pulsanti, checkbox, radio di una **form**

DBLCLICK

Si verifica quando si clicca due volte su un elemento di una form o su un link (Js 1.2)
Associato a: **link**, e elementi di una **form**

DRAGDROP

Si verifica quando si rilascia sulla finestra del browser un oggetto trascinato con il mouse, realizzando il normale funzionamento drag&drop del sistema operativo (Js 1.2).
Se il gestore dell'evento ritorna *false* il drag&drop non viene eseguito.

ERROR

Si verifica quando un documento o un' immagine non possono essere caricati correttamente (Js 1.2).
L'errore riguarda Javascript, non un errore di browser.
Associato a: **document**, **image**.

FOCUS

Si verifica quando un oggetto sulla pagina acquisisce il *focus*, ovvero viene selezionato.
Associato a: tutti gli elementi di un **form**, **frame** e **window**.
N.B. se in un gestore di **onFocus** si utilizzano metodi di avvertimento utente (**alert, confirm, prompt**), all'uscita del metodo di avvertimento il focus ritorna

all'elemento che rigenera l'evento **focus**, continuando all'infinito il ciclo di chiamata.

KeyDown

Si verifica quando si preme un tasto (Js 1.2). L'evento si verifica prima dell'evento **KeyPress**, e se il gestore di **KeyDown** ritorna *false* non viene generato l'evento **KeyPress**.
Associato a: **document, image, link, textarea**.

KeyPress

Si verifica quando si preme o si tiene premuto un tasto; in quest'ultimo caso l'evento si verifica ripetutamente (Js 1.2). L'evento si verifica dopo l'evento **KeyDown**, e solo se l'eventuale gestione di **KeyDown** non abbia riportato *false*.
Associato a: **document, image, link, textarea**.

KeyUp

Si verifica quando si rilascia un tasto (Js 1.2).
Associato a: **document, image, link, textarea**.

Load

Si verifica quando il browser finisce il caricamento di una pagina nella finestra, di tutte le pagine nei frames specificati in una struttura FRAMESET, o di un' immagine
La chiamata del gestore **onLoad** va effettuata nelle dichiarazioni html BODY, FRAMESET e IMAGE.
N.B. se viene caricata una GIF animata, e per l'immagine è stato previsto un gestore **onLoad**, il gestore verrà chiamato ogni volta che cambia l'immagine.
Associato a: **window, image**.

MouseDown

Si verifica quando si preme un bottone del mouse (Js 1.2)
Se il gestore ritorna *false* l'azione tipica è annullata.
Associato a: **document, link** e pulsanti della **form**.

MouseMove

Si verifica quando si sposta il cursore con il mouse (Js 1.2)
L'evento normalmente non avviene e non è gestito. Se dovesse servire gestirlo, va gestito con una particolare procedura di cattura degli eventi. Vedere la guida Javascript di Netscape per approfondimenti.

MouseOver

Si verifica quando il mouse si posiziona sopra un link o una mappa
Associato a: **link**

MouseOut

Si verifica quando il puntatore del mouse esce dall'area del link.o della mappa
Associato a: **link**

MouseUp

Si verifica quando si rilascia un bottone del mouse (Js 1.2)
Se il gestore ritorna *false* l'azione tipica è annullata.
Associato a: **document, link** e pulsanti della **form**.

Move

Si verifica quando l'utente o lo script sposta una finestra o un frame (Js 1.2)
Associato a: **window**, **frame**.

Reset

Si verifica quando si clicca sul bottone Reset di una **form**, azzerandola.
Associato a: **form**.

Resize

Si verifica quando l'utente o lo script ridimensiona una finestra o un frame (Js 1.2)
Associato a: **window**, **frame**.

Select

Si verifica quando si seleziona un testo all' internodi un area di testo odi un campo della **form**.
Associato a:**text, textarea**.

Submit

Si verifica alla pressione del tasto submit di un **form**.
Associato a: **form**.

Unload

Si verifica quando si scarica un documento dalla finestra o si esce dal browser.
La chiamata del gestore **onUnload** va effettuata nelle dichiarazioni html BODY o FRAMESET.
Associato a: **window**.

Sommario

