Javascript
Nozioni di Base

Benvenuto nel wikibook:

JavaScript

Autore: Ramac

```
12 function impostaCookie (nome, valore, percorso, scadenza) {
13    valore=escape(valore);
14
15    if (scadenza == "") {
16       var oggi = new Date();
17       oggi.setMonth(oggi.getMonth()+6);
18       scadenza=oggi.toGMTString();
19    }
20    if (percorso!="")
21       percorso= ";Path=" + percorso;
22
23    document.cookie = nome + "=" + valore + ";expires=" + scadenza + percorso;
24 }
25
```

Vai ai contenuti >>

Fase di sviluppo:JavaScript (Sviluppo)

JavaScript è un linguaggio di scripting: nel suo utilizzo più frequente, quello della programmazione per il web, consiste in un linguaggio formale che fornisce al browser determinate istruzioni da compiere. Una pagina creata in HTML è infatti *statica*, in quanto una volta che la pagina è stata interpretata dal browser la disposizione degli elementi rimane immutata, così come il loro contenuto.
Tramite il JavaScript, invece, è possibile conferire dinamicità alle pagine web permettendo, ad esempio, di creare rollover sulle immagini, modificare i contenuti in base a input dell'utente o creare applicazioni per il Web.

1 Breve storia di JavaScript

Il linguaggio JavaScript fu sviluppato inizialmente nel 1995 dalla Netscape Communications con il nome di **LiveScript** e incluso nel browser Netscape Navigator; il nome fu poi cambiato in JavaScript anche per l'assonanza con il linguaggio Java, che rappresentava una delle tecnologie più avanzate per l'epoca, ma con cui JavaScript non ha niente in comune, se non una sintassi similare.
Dopo il decollo ed il successo di JavaScript, Microsoft decise di aggiungere al proprio browser Internet Explorer un proprio linguaggio di scriptng, **JScript**, che aveva però notevoli differenze con la versione sviluppata dalla Netscape. Nacque così la necessità di standardizzare il JavaScript e venne sviluppato lo standard **ECMAScript**.

2 Strumenti necessari

Gli unici strumenti necessari per la programmazione JavaScript per il Web sono un semplice editor di testi e un browser per vedere il proprio lavoro in azione.

Esistono comunque programmi che aiutano lo sviluppatore JavaScript fornendo un'evidenziazione della sintassi JavaScript o finestre di dialogo per velocizzare il lavoro.

3 Limitazione nell'uso di JavaScript

Una delle principali limitazioni di JavaScript è la possibilità che essi vengano facilmente disabilitati dall'utente tramite le impostazioni del browser. Questo è possibile poiché il JavaScript è un linguaggio lato client, interpretato cioè dal computer dell'utente, che ha quindi la possibilità di disabilitarne alcune funzionalità.
Anche per questo è meglio non demandare funzioni importanti come la gestione di dati sensibili a JavaScript bensì a linguaggi lato server come PHP o Perl.

3.1 Compatibilità tra browser

Un'altra grande limitazione all'uso dei JavaScript è la **compatibilità**: più che per la programmazione HTML o CSS, un programmatore JavaScript deve essere molto attento che il suo **lavoro** sia **compatibile con differenti browser** e **versioni più o meno recenti**.
Ad esempio, le due versioni parallele di JavaScript sviluppate dalla Microsoft per Internet Explorer e dalla Netscape (ora ereditata dalla Mozilla) hanno ancora oggi molte differenze: nonostante la sintassi fondamentale non cambi, molte funzionalità non sono disponibili o sono differenti a seconda del browser in uso. In questo wikibook si cercherà il più possibile di implementare soluzioni compatibili con Mozilla Firefox, Chrome e Windows Internet Explorer; nei casi in cui ciò non sarà possibile, verranno presi eventuali accorgimenti, segnalando comunque le differenze.

4 Inserire un JavaScript in una pagina HTML

Per inserire un codice JavaScript in una pagina HTML, è necessario semplicemente usare l'etichetta **<script>**.

L'attributo più importante di questo tag è *type*, che specifica il tipo di script usato: nel nostro caso il valore da inserire è application/javascript ma è anche usato, nonostante non sia standard, text/javascript; per la maggior parte dei browser è tuttavia possibile omettere l'attributo type in quanto JavaScript è il linguaggio di scripting web predefinito. L'attributo type è opzionale in HTML5.
Esiste anche, ma è sconsigliato rispetto all'uso di type, un attributo language per specificare sempre il linguaggio usato (es language="JavaScript").

All'interno del tag script una volta, per problemi di compatibilità con i browser che non supportavano i JavaScript, il testo veniva delimitato da i delimitatori di commento, per evitare che il codice venisse mostrato in forma “grezza”. Ad esempio:

<script> <!-- ...codice... --> </script>

Questo accorgimento è tuttavia oggi sconsigliato, oltre ad essere praticamente superfluo in quanto la maggior parte dell'utenza utilizza browser JavaScript-compatibili.

È possibile inoltre usare l'attributo src per inserire un codice JavaScript esterno (solitamente un file con estensione *.js*). In questo caso viene lasciato vuoto lo spazio all'interno del tag. Ad esempio:

<script src="script.js"></script>

Il codice JavaScript può essere inserito sia nella sezione head che nella sezione body della pagina HTML, ma le istruzioni vengono comunque eseguite in ordine (a parte nei casi di funzioni che però vedremo più avanti nel corso del libro); la differenza principale è sostanzialmente che il codice dentro head viene eseguito durante il caricamento della pagina, mentre dentro body viene azionato quando richiamato.

Un'altra coppia di attributi, l'uno in alternativa all'altro, che si può usare è async e defer. L'attributo async indica al browser di eseguire lo script in modo asincrono, defer dopo che è stato eseguito il parsing della pagina.
E' consigliabile quindi inserire il tag script nella sezione head, e non nel body, e usare questi attributi.
Questi attributi non dovrebbero essere usati per script inline, cioè che non hanno l'attributo src, in quanto di norma non hanno effetto su questo tipo di script. L'attributo async è nuovo in HTML5.

5 Ciao, Mondo!

Eccoci finalmente al nostro primo programma JavaScript: queste poche righe di codice identificano una pagina HTML su cui viene stampato *Hello, world!*. Create il seguente file con un qualsiasi editor e apritelo con il vostro browser:

<html> <head> <title>La mia prima applicazione JavaScript</title> </head> <body> <script> document.write(“Hello, world!"); </script> </body> </html>

Analizziamo l'unica riga di codice JavaScript presente nella pagina:

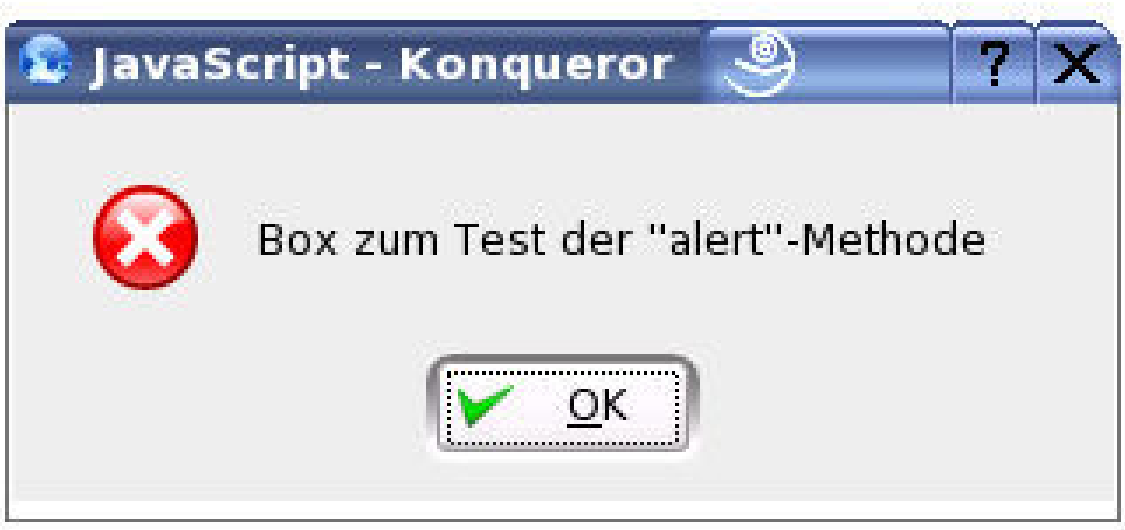
document.write(“Hello, world!");

Quando il browser analizza la pagina, nel momento in cui incontra le istruzioni, se non indicato diversamente (come vedremo più avanti), le esegue secondo l'ordine nel quale sono indicate, ovvero in **sequenza** (questa operazione è chiamata *parsing*).
Le istruzioni sono sempre separate da un punto e virgola (;), ma JavaScript separa ugualmente le istruzioni se intervallate da ritorni a capo.

In particolare, la riga di codice analizzata è un'istruzione che stampa sul documento il testo *Hello, world* tramite il metodo write dell'oggetto document. Per ora ci basti sapere che un *metodo* è un sottoprogramma associato esclusivamente ad un oggetto, che può richiedere uno o più parametri per funzionare al meglio. Il metodo write dell'oggetto document stampa nella pagina il testo indicato tra virgolette all'interno delle parentesi.

Cerchiamo di capire meglio come avviene il *parsing* di uno script JavaScript. Essendo infatti il browser piuttosto veloce a interpretare il codice, non è possibile ai nostri occhi notare l'esecuzione dello script, e sulla pagina compare subito il testo *Hello, world!*. Modifichiamo quindi il nostro codice come segue:

... <script> alert(“Questo è un messaggio”); document.write(“Hello, world!"); alert(“Questo è un altro messaggio”); </script> ...
Il metodo alert mostra una finestrella contenente il testo indicato tra parentesi e un pulsante “OK”. Quando viene



Un alertbox in tedesco visualizzata dal browser Konqueror

eseguito, il *parsing* della pagina si ferma fino a quando l'utente non clicca sul bottone OK; possiamo quindi capire meglio come lavora il browser: l'esecuzione dello script si fermerà infatti due volte in corrispondenza delle due istruzioni alert. Quando viene mostrato il primo messaggio la pagina in secondo piano apparirà vuota, perché non vi è ancora stato impostato il contenuto; quando viene mostrato il secondo messaggio comparirà anche il testo *Hello, world!* in quanto sarà stata già eseguita il metodo write.

Da notare la differenza di sintassi tra il metodo alert ed il metodo write. Il primo è preceduto dalla parola document, mentre il secondo no. In realtà si potrebbe scrivere per esteso anche window.alert, che non è obbligatorio. Il significato di questa sintassi lo si vedrà nel seguito di questo corso.

6 Inserire dei commenti

In JavaScript è possibile inserire dei commenti, ovvero porzioni di testo che verranno ignorate dal parser, di una o più righe delimitandoli da /* e */. È possibile inoltre prevedere commenti di una sola riga utilizzando //. Ad esempio:

questo codice verrà interpretato /* questo verrà ignorato*/ questo codice verrà interpretato /* questo verrà ignorato anche questo */ questo verrà interpretato //e invece questo no //e questo neppure! mentre questo sì

Un elemento necessario per la programmazione è la possibilità di salvare delle informazioni introdotte dall'utente o elaborate a partire da altre dal programma. In informatica, queste vengono chiamate **dati** i quali possono essere salvati in celle di memoria identificate da una **variabile**.

7 Tipi di dati

I dati in JavaScript possono essere di varie tipologie e a seconda della tipologia si potrà lavorare in modo diverso con essi. Il JavaScript è un linguaggio chiamato a **tipizzazione debole**, in quanto ogni volta che si fa riferimento ad un dato non è necessario specificare il tipo, che viene attribuito automaticamente dal parser in base al contesto. In linguaggi di programmazione come il C, ad esempio, è necessario specificare se una variabile conterrà una stringa di testo o un numero o altri tipi di dati, mentre ciò non è necessario in JavaScript.

7.1 Numeri

I dati **numerici** possono essere positivi e negativi e si distinguono in **integer** (numeri interi) e **float** (numeri a virgola mobile).
Per convertire un valore qualsiasi in un valore numerico, JavaScript mette a disposizione due funzioni: parseInt e parseFloat. Per ora ci basti sapere che una *funzione* è un sottoprogramma (come un metodo) che non è associato ad alcun oggetto ma restituisce comunque un valore in base ad eventuali parametri accettati in ingresso. Per convertire i dati, ad esempio una stringa, queste funzioni analizzano carattere per carattere la stringa fornita come input prendendo in considerazione solo i numeri e, nel caso di parseFloat, anche il separatore decimale .. Ad esempio:

parseFloat(“34acb”) //restituisce 34. parseInt(“3eac34”) //restituisce 3.

7.1.1 Not a Number

Può tuttavia succedere che il valore passato alle funzioni di parsing dei numeri non possa essere elaborato come numero. In questo caso la funzione restituisce un valore particolare, **NaN**, che è l'acronimo per **N**ot **a** **N**umber, non un numero. È possibile testare se un valore è NaN usando la funzione isNaN:

isNaN(“123”) //restituisce false o 0. isNaN(“abc”) //restituisce true o 1.

7.2 Stringhe

In informatica una **stringa** è una sequenza di uno o più caratteri alfanumerici. In JavaScript le stringhe si indicano inserendole all'interno di virgolette doppie (") o apici (') Bisogna però fare attenzione a chiudere una stringa con lo stesso apice con la quale è stata aperta; sono ad esempio stringhe valide:

“Hello, world ! 1234” “Peter O'Toole”

ma non lo è ad esempio

'Peter O'Toole'

in quanto il parser analizzerebbe la stringa e, arrivato a O' penserebbe che la stringa si chiuda, senza sapere cosa sia Toole'.

È possibile anche indicare che caratteri come " e ' non indicano la fine del testo ma un carattere qualsiasi facendole precedere dal **carattere di commutazione** o *backslash* \. Ad esempio sono stringhe valide:

'Peter O\'Toole' “Questo libro è depositato su \"it.wikibooks\""

In realtà ogni carattere è commutabile in una sequenza esadecimale usando la notazione \xNN dove NN è un numero esadecimale che identifica un carattere nel set Latin-1.

Per forzare la conversione da un numero ad una stringa basta usare la sintassi numero.toString().

7.3 Dati booleani

Il tipo di dato **booleano** può assumere i soli valori true (vero) e false (falso). Il tipo di dato booleano è fondamentale per la selezione binaria e per il *decision-making*.
Quando è atteso un tipo di dato booleano il parser si comporta in maniere differenti:

- se viene fornito un numero, questo viene convertito in false se è 0 oppure -0, in true se è 1.
- se viene fornito una stringa, questa viene convertito in false se è una delle seguenti:
  - null
  - ""
  - false
  - undefined
  - NaN

in true negli altri casi.

8 Variabili

Una **variabile** in JavaScript identifica una porzione di memoria nella quale vengono salvati i dati durante l'esecuzione dello script.

Quando si lavora con le variabili, è necessario per prima cosa indicare al *parser* il suo nome utilizzando l'istruzione var, nonostante sia equivalente la dichiarazione di una variabile anche senza quest'istruzione:

var nome_variabile;

dichiara una variabile nome_variabile. Essendo JavaScript un linguaggio a tipizzazione debole, non è necessario indicare il tipo di dato della variabile, a meno che non si stia lavorando con oggetti (si vedrà più avanti). In questo caso la variabile appena dichiarata non avrà valore, non è stata cioè ancora **inizializzata**; è possibile però inizializzare la variabile direttamente insieme alla dichiarazione:

var nome_variabile = espressione;

dichiara una variabile con il valore di espressione. Per **espressione** si intende una sequenza di operatori, variabili e/o dati che restituisca a sua volta un valore.

Quando si scelgono dei nomi per le variabili, si faccia attenzione ad alcune regole:

- JavaScript è case-sensitive (sensibile alle maiuscole): var1 e Var1 sono due variabili differenti
- in JavaScript esistono, come in ogni linguaggio formale, delle parole riservate, che identificano cioè particolari comandi per il parser. Non bisogna quindi usare parole riservate come nomi di variabili. Per l'elenco delle parole riservate in JavaScript, si veda l'appendice.
- una variabile non può contenere caratteri particolari (? : ; , . ecc...), tranne l'*underscore*, e non può iniziare con un numero

L'operazione fondamentale da usare con le variabili è l'**assegnazione**, che consiste nell'assegnare, nell'attribuire ad una variabile un valore. La sintassi è intuitiva e semplice:

nome_variabile = valore;

dove valore è un'espressione. Ad esempio:

var1 = “ciao!"; var2 = 3; var3 = false; var4 = var3; //attribuisce a var4 una copia del valore di var3

E' possibile assegnare a una variabile un array:

lettere = ["a”, “b”, “c"];

JavaScript non supporta gli array associativi. Al loro posto si usano gli oggetti. Quindi usare i metodi degli array su array associativi produrrà risultati non corretti. Per rappresentare i valori con coppie chiave-valore si usa questa sintassi (che inizializza un oggetto):

persona = { nome: “Mario”, cognome: “Rossi”, indirizzo: “Roma” };

Per accedere al valore di una proprietà della variabile (e quindi dell'oggetto):

x = persona.nome;

In questo modulo affronteremo l'uso degli **operatori** e della **conversione** (***casting***) per lavorare con i dati in JavaScript. Di ogni operatore verrà indicato il tipo di dati con i quali lavora e il tipo di dato restituito.

9 Operatori matematici

Questi operatori lavorano con valori interi o float e restituiscono sempre un valore numerico; sono piuttosto intuitivi in quanto corrispondono ai normali operatori algebrici. Sono:

- + e - (addizione e sottrazione algebrica)
- * e / (moltiplicazione e divisione algebrica)
- % restituisce il resto della divisione intera tra i due operanti (modulo).

Quando si lavora con gli operatori numerici è bene fare attenzione alla precedenza algebrica delle operazioni. Vengono infatti valutate prima moltiplicazioni e divisioni e poi addizioni e sottrazioni.
È tuttavia possibile utilizzare delle parentesi per comporre più operatori e più espressioni; in JavaScript non esistono distinzioni tra parentesi tonde, quadre o graffe, in quanto si usano solo quelle tonde.

Vediamo, ad esempio, come convertire un valore di temperatura espresso in gradi Fahrenheit al suo corrispondente in gradi Celsius. Inseriamo nella pagina HTML questo codice:

<script type="text/javascript"> var gradiF = prompt(“Introdurre la temperatura in gradi Fahrenheit”,100); var gradiC = 5/9 * (gradiF - 32); alert(gradiC); </script>

Analizziamo riga per riga il listato JavaScript:

- la prima riga dichiara la variabile gradiF e le assegna il valore restituito dalla funzione prompt.

  Il metodo prompt visualizza una finestra di input che mostra il testo indicato come primo argomento e un valore di default indicato come secondo argomento (separato tramite la virgola). Nel nostro caso, la funzione mostrerà una finestra chiedendo di introdurre il valore della temperatura in gradi Fahrenheit proponendo come valore 100. L'utente è libero di modificare o meno questo valore e di premere uno dei due pulsanti OK o Annulla. Premendo OK, la funzione restituisce il valore immesso dall'utente, altrimenti restituisce 0.

Un esempio di finestra prompt

- la seconda riga calcola il valore della temperatura in gradi Celsius usando l'equazione apposita
- la terza riga mostra una finestra alert (già vista nei precedenti moduli) contenente il valore della temperatura in gradi Celsius.

Si noti che il metodo prompt restituisce sempre un valore stringa ma, nel momento in cui la variabile stringa gradiF viene usata insieme ad operatori numerici, il suo valore viene automaticamente convertito in un valore numerico. Possiamo vedere inoltre che, poiché il metodo alert richiede un valore stringa mentre noi abbiamo passato un valore numerico, JavaScript converte automaticamente il numero in una stringa contenente esattamente il valore richiesto.

Nel comporre le espressioni è necessario tenere conto della precedenza delle operazioni (prima vengono valutate moltiplicazioni e divisioni, poi addizioni e sottrazioni). Nel comporre le espressioni possono risultare utili le parentesi; in questo caso è possibile usare solo parentesi tonde (JavaScript non supporta l'uso di parentesi quadre e graffe nelle espressioni). Ad esempio:

3 + 4 / 2 //restituisce 5 (3 + 4) / 2 //restituisce 3.5 (3 + (2 + 6) * 2) /2 //restituisce 9.5

Esistono poi due operatori chiamati **unari** in quanto lavorano su un solo dato numerico. Sono:

- ++ incrementa di uno il valore della variabile
- -- decrementa di uno il valore della variabile

Questo operatore può essere usato da solo oppure all'interno di un'espressione:

var1++ //corrisponde a var1 = var1 + 1 var1-- //corrisponde a var1 = var1 - 1 var2 = var1++ - 15 // corrisponde a var2 = var1 −15; var1 = var1 +1

È possibile inoltre nel caso di istruzioni come

var1 = var1 + n; var2 = var2 / n

Usare la notazione:

var1 += n; var2 /= n;

10 Operatori stringa

L'operazione più utilizzata quando si lavora con le stringhe è la **concatenazione**, che consiste nell'unire due stringhe accostandole una di seguito all'altra. L'operatore di concatenazioni in JavaScript è +. Vediamo alcuni esempi:

alert(“Io sono " + “un utente di wikibooks!"); var nome = “Luigi"; alert(“Io mi chiamo " + nome);

L'output di questo breve listato sono due finestre alert contenti il testo *Io sono un utente di wikibooks!* e *Io mi chiamo Luigi*. Si noti lo spazio prima delle virgolette. Possiamo quindi ampliare l'esempio precedente modificando l'ultima riga con:

alert(gradiF + " gradi Farhenheit corrispondono a " + gradiC + " gradi Celsisus”);

In questo caso la variabile gradiC viene convertita automaticamente in una stringa.

11 Operatori booleani

Sono chiamati operatori **booleani** o **di confronto** quelli che restituiscono un valore booleano (*vero* o *falso*) e confrontano due valori dello stesso tipo di dato. Sono:

- < (minore)
- <= (minore o uguale)
- == (uguale)
- >= (maggiore o uguale)
- > (maggiore)

Tranne che per ==, il loro uso è strettamente legato ai dati numerici ed è intuitivo. Ad esempio:

4 > 3 //restituisce true 4 == 5 //restituisce false

È possibile inoltre comparare le stringhe. Ad esempio:

“wikibooks” == “wikibooks” //true “wikibooks” == “it.wikibooks” //false “wikibooks” == “Wikibooks” //false (distingue maiuscole e minuscole)

È possibile inoltre comporre le espressioni booleane usando gli operatori della logica, che lavorano con valori booleani e restituiscono a loro volta un valore bollano. Sono:

- && (corrisponde alla congiunzione logica, *et*)
- || (corrisponde alla disgiunzione logica, *vel*)
- ! (corrisponde alla negazione logica, *non*)

Anche in questo caso è possibile comporre le espressioni con le parentesi

(a > 0) && (a < 100) // Restituisce true se il valore di a è maggiore di 0 e minore di 100 (a < 0) || (b > 20) // Restituisce true se almeno una delle due espressioni è vera !(a < 0) // Restituisce true se il valore di a NON è minore di 0

Se JavaScript si aspetta un dato booleano e riceve un altro tipo di dato, converte in false le stringhe vuote e il numero 0, in true il resto.

12 Comporre le espressioni

Dopo aver analizzato gli operatori, è possibile comporre le **espressioni**. Un'istruzione è un insieme di uno o più valori legati da operatori che restituisce un valore; per comporre un'espressione possiamo usare quindi qualsiasi istruzioni restituisca un valore, ad esempio:

alert(“Ciao, " + prompt(“Come ti chiami?"));

è una espressione valida, in quanto la funzione prompt restituisce un valore; il risultato è una casella di prompt seguita da un alertbox che dà il benvenuto all'utente

La **selezione** (o *controllo*) è, insieme alla sequenza (già vista nei moduli precedenti) e al ciclo (che vedremo nel prossimo), una delle tre strutture fondamentali della programmazione e al suo livello più semplice consiste in una scelta tra due blocchi di istruzioni da seguire in base ad una condizione valutata inizialmente, che può essere vera o falsa.

13 Selezione semplice

È la forma più semplice di selezione. Vediamo la sua sintassi:

if (condizione) { istruzioni1 } else { istruzioni2 }

Quando il parser raggiunge questo listato, valuta il valore dell'espressione booleana condizione. Nel caso condizione restituisca true verranno eseguite le istruzione comprese tra la prima coppia di parentesi graffe (nell'esempio istruzioni1) altrimenti vengono eseguite le istruzioni comprese tra la seconda coppia di parentesi (istruzioni2). Vediamo un esempio:

var a = prompt(“In che anno Cristoforo Colombo scoprì le Americhe?",2000); if (a == 1492) { alert(“Risposta esatta!"); } else { alert(“Hai sbagliato clamorosamente...!"); }

Il metodo già visto prompt chiede all'utente di inserire la data della scoperta delle Americhe.
Nella riga successiva compare la nostra selezione: se a (ossia il valore inserito dall'utente) è uguale a 1492 allora viene mostrato un messaggio di complimenti; in caso contrario, viene mostrato un altro messaggio che informa l'utente dell'errore commesso. Si noti l'uso dell'operatore booleano == (uguale) che può restituire true o false. Notare anche la differenza tra l'*operatore di assegnazione*(=) e quello di *uguaglianza*(==), che se confusi erroneamente restituiscono risultati diversi da quelli previsti.

È possibile, nel caso non sia necessario, omettere il blocco else; se inoltre il primo blocco di istruzioni è costituito da una sola riga e non c'è il blocco else, è possibile tralasciare le parentesi graffe. Esempio:

var a = prompt(“In che anno Cristoforo Colombo scoprì le Americhe?",2000); if (a == 1492) alert(“Risposta esatta!"); else alert(“Hai sbagliato clamorosamente...!");

è una soluzione valida.

14 Blocchi if annidati e condizioni multiple

All'interno di un blocco if è ovviamente possibile inserire a loro volta altri blocchi di selezione, che sono chiamati **annidati** (uno dentro l'altro). Si faccia però attenzione al seguente esempio:

if (cond1) { if (cond2) { istruzioni } }

Questo codice è inutilmente lungo, in quanto è possibile riassumerlo con un unico blocco usando gli operatori logici già visti in precedenza:

if (cond1 && cond2) { istruzioni }

15 else if

Esiste una forma particolare di else che permette di eseguire all'interno del suo blocco un'altra selezione. Vediamo questo codice:

if (cond1) { istruzioni; } else { if (cond2) { istruzioni } else { istruzioni; } }

Se all'interno del blocco else c'è una sola selezione è possibile usare questa sintassi più abbreviata

if (cond1) { istruzioni1; } else if (cond2) { istruzioni2 } else { istruzioni3; }

In questo caso il parser controllerà cond1; se essa è vera vengono eseguite le istruzioni1, se invece è falsa e cond2 è vera, verranno eseguite le istruzioni2; se entrambe le condizioni sono false verranno eseguite le istruzioni3.

16 switch

L'istruzione switch è un tipo particolare di selezione che permette di verificare i possibili valori dell'espressione presa in considerazione. La sintassi è:

switch (espressione) { case val1: istruzioni1; case val2: istruzioni2; ... case val_n: istruzioni_n; default: istruzioni }

Quando il browser incontra questa istruzione, scorre tutti i valori val1, val2, ...val_n fino a che non incontra un valore uguale all'espressione indicata tra parentesi oppure un blocco default. In questo caso inizia ad eseguire il codice che segue i due punti. Se non c'è alcun blocco default e non ci sono valori che soddisfino l'uguaglianza, il parser prosegue eseguendo le istruzioni dopo la chiusura delle parentesi graffe.
Si faccia però attenzione a questo spezzone:

var a = 1; switch (a + 1) { case 1: alert(“a = zero”); case 2: alert(“a = uno”); case 3: alert(“a = due”); case 4: alert(“a = tre”); default: alert(“a maggiore di 3”); }

Si potrebbe supporre che l'output di questo codice sia solo una alertbox contenente il messaggio a = 1 in quanto il blocco 2 soddisfa l'uguaglianza (a + 1 = 2). Non è tuttavia così: il parser dopo essersi fermato al blocco 2 proseguirà leggendo anche i blocchi successivi e il blocco default. Per evitare ciò dobbiamo inserire un'istruzione break. L'istruzione break indica al parser di interrompere la lettura della struttura switch e di proseguire oltre le parentesi graffe. L'esempio corretto è:

switch (a + 1) { case 1: alert(“a = zero”); break; case 2: alert(“a = uno”); break; case 3: alert(“a = due”); break; case 4: alert(“a = tre”); break; default: alert(“a maggiore di 3”); }

Ovviamente il break non è necessario per l'ultimo blocco.
Vediamo poi un altro listato:

switch (a + 1) { default: alert(“a maggiore di 3”); break; case 1: alert(“a = zero”); break; case 2: alert(“a = uno”); break; case 3: alert(“a = due”); break; case 4: alert(“a = tre”); }

Bisogna fare attenzione a non porre il blocco default in cima, in quanto il parser si fermerebbe subito lì e, incontrata l'istruzione break, salterebbe alla fine del blocco switch senza valutare le altre espressioni.

Notare che la selezione switch è sostituibile con una selezione di if ed else if.

17 Operatore ternario

Esiste oltre agli operatori già menzionati nel capitolo precedente anche un cosiddetto **operatore ternario**, che lavora con tre valori. La sua sintassi è:

condizione ? esp1 : esp2

Quando il parser incontra questa notazione, valuta il valore booleano di condizione. Se è vero, restituisce il valore di esp1 altrimenti quello di esp2. Questo permette di creare semplici selezioni; ad esempio:

var anni = prompt('Quanti anni hai?', 20); var msg = “Ciao, vedo che sei " + (anni >= 18 ? “maggiorenne” : “minorenne”) + "!"; alert(msg);

In questo caso l'operatore ternario restituisce “maggiorenne” se anni è maggiore o uguale a 18, altrimenti restituisce “minorenne”.

18 Esercizi

- Scrivere un programma che converta un voto numerico chiesto all'utente (da 0 a 10) in un giudizio (insufficiente, sufficiente, buono, ecc...).

Soluzione
var voto = prompt(“Introduci il voto dell'alunno”,6); var msg; switch (voto) { case “0": msg="Il compito non è stato consegnato (non classificabile)"; break; case “1": case “2": msg="Insufficienza grave"; break; case “3": case “4": msg="Insufficienza"; break; case “5": msg="Sufficienza scarsa"; break; case “6": msg="Sufficienza"; break; case “7": msg="Sufficienza piena"; break; case “8": msg="Buono"; break; case “9": msg="Discreto"; break; case “10": msg="Ottimo con lode"; break; default: msg="Dati non validi"; } alert(msg);
Analizzando il codice, possiamo vedere che nei casi voto sia uguale ad esempio a 1 e 2 o a 3 e 4 viene eseguita la stessa operazione in quanto non c'è un break per ogni blocco case. Se nessuno dei valori soddisfa l'uguaglianza con a viene restituito un messaggio di errore

- È possibile risolvere il programma precedente usando solo con istruzioni if?

Soluzione

Sì, prevedendo una serie di if, meglio se utilizzando una struttura else if:

//... if (voto == 0) { msg = “Il compito non è stato consegnato (non classificabile)"; } else if (voto == 1 || voto == 2) { msg = “Insufficienza grave"; //così avanti tutti gli altri casi, fino all'ultimo //che corrisponde al default } else { msg = “Dati non validi"; }

L'**iterazione** è una struttura di controllo (come la selezione) che permette l'esecuzione di una sequenza di una o più istruzioni fino al verificarsi di una data condizione.

19 Cicli con condizione

La forma più semplice di ciclo è composta da una condizione valutata inizialmente e ad un blocco di istruzioni eseguito sino a quando la condizione iniziale non risulti false. In JavaScript usiamo in questo caso un ciclo while la cui sintassi è:

while (condizione) { istruzioni; }

Quando JavaScript incontra questa struttura:

1. valuta il valore booleano di condizione. Possiamo avere quindi due casi:
   (a) se è vero, esegue il blocco di istruzioni delimitato dalle parentesi graffe e quindi ricomincia dal punto 1
   (b) se è falso, salta al punto 2
2. prosegue la lettura delle istruzioni successive

Si noti che:

- è possibile che il blocco di istruzioni non venga mai eseguito. Questo è possibile se il valore della condizione iniziale è falso già dalla sua prima valutazione

- se il blocco di istruzioni non modifica in qualche modo le variabili che intervengono nella condizione iniziale, può accadere che questo venga ripetuto all'infinito

Esiste poi la possibilità di porre la condizione alla fine del blocco di istruzioni; in questo caso si è sicuri che questo verrà eseguito almeno una volta. La sintassi è:

do { istruzioni; } while (condizione)

Vediamo alcuni esempi; miglioramo ad esempio il piccolo programma che valutava se l'utente era maggiorenne o minorenne:

var anni; do { anni = prompt(“Quanti anni hai?",20); } while (isNaN(anni)) var msg = “Ciao, vedo che sei " + (anni >= 18 ? “maggiorenne” : “minorenne”) + "!"; alert(msg);

In questo modo abbiamo inserito un controllo per verificare che il valore immesso dall'utente sia un numero: la richiesta dell'età viene infatti ripetuta se il valore introdotto **non è** un numero.

20 Cicli con contatore

JavaScript, come molti altri linguaggi programmazione, offre la funzionalità di cicli con un contatore che viene incrementato o decrementato ad ogni ripetizione. La sintassi è:

for (condizione_iniziale; condizione_finale; incremento_contatore) { istruzioni; }

Ad esempio:

for (var i = 1; i <= 10; i++) { alert(“i = " + i); }

Eseguendo il codice appena visto otterremo come output dieci messaggi che indicano il valore di i ad ogni iterazione.

È possibile inoltre fare in modo che il contatore venga incrementato di più di un'unità al secondo:

for (var i = 1; i <= 20; i+=2) { alert(“i = " + i); }

Questo codice mostra sempre i primi 10 numeri pari.

Se l'operazione da eseguire è molto semplice, come nel primo esempio che abbiamo visto, è possibile anche usare una sintassi abbreviata:

for (var i = 1; i<= 10; alert(“i = " + i++));

L'istruzione di alert contiene infatti un operatore unario "++" applicato alla variabile contatore i; ad ogni ripetizione del ciclo, JavaScript mostra prima il messaggio di *alert* e poi incrementa la variabile.

21 Iterare sugli elementi di un array

Per iterare in modo automatico gli elementi di un array (funzionalità trattata più avanti nel corso di questo libro), esiste un ciclo particolare, la cui sintassi è:

for (indice in nomeArray) { //... }

L'array assegna man mano ad indice i valori dell'array indicato.

22 break e continue

Le istruzioni break e continue servono per alterare il normale flusso dell'esecuzione di un ciclo:

- **break** serve per uscire completamente dal ciclo in cui ci si trova riprendendo dalla prima riga dopo la fine del ciclo. Questo è molto utile se ad esempio si sta effettuando una ricerca tra un insieme di valori: una volta trovato il valore che ci interessa, è inutile andare avanti nel ciclo e quindi si usa break.
- **continue** è leggermene diverso da break. Quando viene inserito in un ciclo, interrompe la ripetizione corrente e l'esecuzione continua con il valore successivo del ciclo. Ad esempio si vogliono stampare tutti i numeri da n a m tranne quelli divisibili per 7:

for (var i = n; i <= m; i++) { if (i % 7 == 0) continue; // se divisibile per 7 interrompe e ricomincia document.write(i); }

Come molti linguaggi di programmazione, anche JavaScript da la possibilità di creare le proprie funzioni personalizzate.

Come è stato già accenato in precedenza, una funzione è un sottoprogramma identificato da una sequenza di caratteri che può accettare o meno nessuno o più parametri e può restituire un valore.

23 Creare le proprie funzioni: una panoramica

La sintassi per la creazione di una nuova funzione è la seguente:

function nome_funzione (arg1, arg2, argN...) { "codice" }

Ad esempio:

function benvenuto (nome) { alert(“Benvenuto, " + nome); }

Se vogliamo eseguire il codice contenuto nella funzione dobbiamo **richiamarla** (o **invocarla**). Ad esempio:

var n = prompt(“Come ti chiami?"); if (n != "") benvenuto (n);

Con questo breve spezzone chiediamo all'utente di inserire il nome e, se non risponde con una stringa vuota e non ha premuto Annulla, gli porge il benvenuto (si noti che il metodo prompt restituisce una stringa vuota se l'utente fa clic su “Annulla”).

24 Usare i parametri

Come abbiamo visto, è possibile prevedere che l'utente possa passare alcuni parametri alla funzione. Dando una sguardo alla funzione di benvenuto creata precedentemente, vediamo che il parametro nome della funzione diventa poi automaticamente una variabile; se quando viene chiamata la funzione viene omesso, il parametro assumerà un valore nullo (che diventa 0 per i numeri, una stringa vuota per i valori alfanumerici, falso per i valori booleani).

25 Impostare un valore di ritorno

Impostare un valore di ritorno di una funzione è molto semplice, basta seguire la sintassi:

return valore;

Quando incontra l'istruzione return, JavaScript interrompe l'esecuzione della funzione e restituisce il valore indicato. Ad esempio:

function somma (a, b) { //una semplice funzione return a+b; } var c = somma(3,5); //c assumerà il valore 8 somma(4,12); //in questo caso non succede nulla

Una funzione può anche essere utilizzata prima che ne venga dichiarato il contenuto. Il codice precedente e quello riportato qui sotto sono infatti entrambi validi

var c = somma(3,5); //la funzione somma non esiste ancora! function somma (a, b) { //una semplice funzione return a+b; }

Il concetto di **oggetto** è molto importante nella programmazione in JavaScript. In questo modulo verranno spiegate le caratteristiche comuni degli oggetti; nei moduli seguenti verranno invece trattati nel dettaglio gli oggetti intrinseci di JavaScript.

L'uso degli oggetti e delle loro funzionalità entra a far parte del paradigma della **programmazione orientata agli oggetti** (abbreviata **OOP**, **O**bject **O**riented **P**rogramming)

26 Cosa sono gli oggetti

Per avvicinarci al concetto di oggetto in informatica, possiamo pensare al concetto di oggetto nel mondo reale.
Per creare un nuovo oggetto è necessario partire da un modello (in JavaScript un oggetto chiamato **prototipo**) che indichi come creare altri oggetti dello stesso tipo (ogni oggetto è un'**istanza** del prototipo).

Gli oggetti possono inoltre possedere delle caratteristiche (**proprietà**): nel caso ad esempio di macchina (per fare un paragone con la realtà), saranno la cilindrata, le dimensioni, il costo, ecc....

Ciascuna istanza espone inoltre la possibilità di effettuare delle operazione su di essi (**metodi**): per la nostra macchina, metterla in moto o guidare. Queste operazioni modificheranno delle caratteristiche come il livello del suo carburante o la velocità.

Una volta introdotto il concetto di oggetto, dobbiamo avere però la capacità di astrarre: gli oggetti dell'informatica non corrispondono a quelli della realtà; hanno però numerosi vantaggi, tra i quali la possibilità di trattare dati più complessi di numeri e stringhe.

27 Gli oggetti in JavaScript

JavaScript permette di creare i propri oggetti personalizzati; tuttavia per imparare è meglio iniziare a lavorare sugli oggetti predefiniti del linguaggio.

Per creare una nuova istanza di un oggetto si utilizza una funzione chiamata **costruttore**. Di fatto, in realtà, per definire un oggetto prototipo è quindi sufficiente creare un nuovo costruttore.

La sintassi quindi:

var variabile = new nome_oggetto ();

In questo modo la variabile variabile sarà l'unico modo per fare riferimento all'istanza di nome_oggetto appena creata.

I costruttori solitamente prevedono anche una serie di parametri per impostare automaticamente alcune proprietà dell'oggetto creato:

var variabile = new nome_oggetto (parametri_del_costrutture);

Supponiamo di avere a che fare con un oggetto macchina implementato in JavaScript. Per inizializzare la variabile “la_mia_macchina” dobbiamo dunque richiamare la funzione costruttore del prototipo macchina, che per esempio potrebbe prevedere un parametro “modello” e “colore vernice":

var la_mia_macchina = new macchina (“Fiat Bravo”, “Rosso”);

E' possibile creare un oggetto anche assegnando a una variabile delle coppie chiave - valore:

var oggetto = { proprieta1: valore1, proprieta2: valore2 };

27.1 Proprietà

Possiamo pensare ad una proprietà come ad una variabile associata al singolo oggetto; il suo valore viene attribuito inizialmente dal costruttore e successivamente viene modificato agendo sull'oggetto (operando sui metodi, ecc...). Per fare riferimento alla proprietà (per il recupero o per l'assegnazione) si usa la sintassi:

nome_oggetto.nome_proprietà

Alcune proprietà possono essere di sola lettura, cioè il loro valore può essere letto ma non modificato: esse sono infatti legate a caratteristiche intrinseche dell'oggetto, oppure sono determinate dal prototipo in base a dati forniti in precedenza.

Se ad esempio riverniciamo la nostra macchina, ne cambiamo la proprietà “colore_vernice":

la_mia_macchina.colore_vernice = “Giallo”

Inoltre, supponiamo che il prototipo preveda la ricerca automatica della lunghezza della macchina in base al modello e che tale valore sia stato memorizzato nella proprietà di sola lettura “lunghezza” (infatti non è possibile intervenire sulla lunghezza della macchina, una volta scelto il modello):

alert('La mia macchina è lunga ' + la_mia_macchina.lunghezza + ' m!');

27.2 Metodi

Un metodo è una funzione associata al singolo oggetto e definito nel costruttore; se nel prototipo è prevista una funzione metodo_esempio sarà possibile eseguire la funzione tramite la sintassi:

nome_oggetto.metodo_esempio () //ricordarsi le parentesi anche se non passiamo parametri!

Dal momento che le funzioni possono prevedere un valore di ritorno, sarà possibile inserire la notazione vista precedentemente all'interno di un'espressione.

Ad esempio, con la nostra macchina:

la_mia_macchina.rifornisci (20) //20 euro di benzina

NOTA: l'esempio riportato non ha nulla a che vedere con le intenzioni espresse, poiché aggiunge semplicemente un parametro di ingresso al metodo, senza alcun “valore di ritorno” previsto e/o salvato. Correggere.

28 La struttura with

Quando si lavora con gli oggetti, può risultare comodo il costrutto with, che permette di accedere più volte ad un oggetto senza dover ogni volta specificare il suo nome. Ad esempio:

with (la_mia_macchina) { rifornisci (20) //20 euro di benzina alert(lunghezza); altra_macchina.rifornisci(50); //per riferirmi ad altri oggetti devo indicare il loro nome } //qui si conclude il blocco with

29 Creare oggetti personalizzati

Per creare oggetti personalizzati, consultare il modulo Costruttori e prototipi.

L'oggetto **String** permette di effettuare numerose operazioni sulle stringhe quali ricerca, isolamento di un carattere e altro ancora.

30 Costruttore

Per creare un nuovo oggetto della classe String usiamo la sintassi:

var nome_variabile = new String(stringa);

Il costruttore prende come parametro la stringa che sarà manipolata nel corso dello script.

L'oggetto String è un po' particolare, in quanto, come vi sarete già accorti, corrisponde ad un “doppione” del tipo di dato primitivo stringa, analizzato precedentemente nel corso del libro.
Dal momento che JavaScript effettua automaticamente la conversioni dei dati quando necessario, la differenza è dal punto di vista pratico, in quanto:

- se creiamo una stringa con la normale sintassi

var variabile = “testo";

  e successivamente volessimo trattarla come un oggetto String usando metodi o proprietà della classe String, JavaScript converte automaticamente la stringa in un oggetto String

- se abbiamo un oggetto String e volessimo recuperare la sua stringa, JavaScript converte automaticamente l'oggetto String in una stringa contente la stringa indicata nel costruttore.

31 Proprietà

L'oggetto String dispone di una proprietà rilevante, la proprietà length che restituisce il numero di caratteri contenuti in una stringa:

var esempio = “Sono una stringa primitiva” alert (esempio.length); //restituisce 26

Nell'esempio appena visto, nella seconda riga la variabile esempio viene convertita in un oggetto String per accedere alla proprietà length.
Sempre per quanto appena detto, potremmo semplicemente scrivere:

alert (“Sono una stringa primitiva”.length); //restituisce 26

32 Metodi

I metodi della classe String permettono di eseguire molteplici operazioni sulle stringhe; si noti che questi metodi lavorano sulla stringa contenuta nell'oggetto ma restituiscono il valore desiderato senza modificare il valore dell'oggetto stringa.

32.1 toLowerCase() e toUpperCase()

Questi due metodi restituiscono la stringa forzando la capitalizzazione o tutta minuscola o tutta minuscola. Attenzione:

var t1 = new String (“TeStO”); var t2 = t1.toLowerCase() //restituisce “testo” var t3 = t1.toUpperCase() //Restituisce “TESTO” // attenzione: alla fine di questo codice la variabile t1 contiene ancora “TeStO"!

32.2 charAt() e charCodeAt()

Il metodo charAt() restituisce il carattere della stringa che si trova alla posizione specificata; il primo carattere è identificato dalla posizione 0. Ad esempio:

var testo = prompt(“Inserisci un testo”, “Hello, world!"); var primoCarattere = testo.charAt(0); var quartoCarattere = testo.charAt(3); var ultimoCarattere = testo.charAt(testo.length - 1);

Questo semplice codice estrapola dalla stringa fornita in input dall'utente l'ultimo carattere. Per fare ciò recupera il carattere alla posizione testo.length - 1: infatti, dal momento che il conteggio dei caratteri parte da 0, nel caso di “Hello, world!" avremo queste posizioni:

Quindi, essendo la stringa sia composta da 13 caratteri, l'ultimo si trova alla posizione 12, ovvero 13-1.

Il metodo charCodeAt() funziona come charAt() ma invece di restituire il carattere alla posizione specificata, restituisce il suo codice ASCII. Ad esempio:

var testo = “Hello, world!"; alert(testo.charCodeAt(1)); //mostra il codice ASCII del carattere “e”, ovvero 101

32.3 fromCharCode()

Il metodo fromCharCode() è l'opposto di charCodeAt(): infatti prende come argomento un numero qualsiasi di numeri che vengono interpretati come codici Ascii e trasformati in una stringa.
Questo metodo è però un po' particolare, in quanto non necessita di lavorare su un oggetto String; per questo viene detto **statico**. Per utilizzare il metodo sarà sufficiente usare la sintassi:

String.fromCharCode(parametri);

facendo cioè riferimento alla classe String.
Vediamo un esempio:

var stringa; for (codice = “0”.charCodeAt(0); codice <= “9”.charCodeAt(0); codice ++) { stringa = stringa + String.fromCharCode(codice); }

Questo semplice *snippet* scorre dal codice del carattere “0” a quello del carattere “9” creando così la stringa:

0123456789

32.4 indexOf() e lastIndexOf()

Uno dei metodi dell'oggetto String che useremo più di frequente sarà indexOf() la cui sintassi è:

oggetto_string.indexOf(search, start);

In pratica, il metodo cerca nella stringa la prima occorrenza della sottostringa search e ne restituisce la posizione (a partire da 0); se la stringa non viene trovata, restituisce -1. Il parametro opzionale start specifica la posizione dalla quale iniziare la ricerca (di default è 0). Ad esempio;

var stringa = “Ma la volpe col suo balzo ha raggiunto il quieto fido” //ricordiamoci che JS converte automaticamente le stringhe var posiz = stringa.indexOf(“v”); //contiene il valore 6) var posiz2 = stringa.indexOf(“k”); // restituisce -1

Il metodo lastIndexOf() funziona analogamente ad indexOf() ma inizia la ricerca dalla fine della stringa e non dall'inizio

32.5 substr()

Il metodo substr() presenta la sintassi:

nome_oggetto_string.substr(start, length)

Il metodo estrae dalla stringa indicata una sottostringa a partire dalla posizione indicata dal parametro start un numero length di caratteri. Se quest'ultimo parametro non è specificato, il metodo include anche tutti i caratteri a partire da quello iniziale fino alla fine della stringa. Ad esempio:

var stringa = “Questo è un esempio"; var str2 = stringa.substr(0, 6); //restituisce “Questo” var str3 = stringa.substr(7); //restituisce "è un esempio”

32.6 replace()

Il metodo replace() restituisce la stringa iniziale sostituendo alla prima occorrenza della stringa indicata come primo parametro quella fornita come secondo. Ad esempio:

var s = “L'utente Pinco ha modificato questa pagina” var s2 = s.replace(“Pinco”, “Pallino”); //s2 ora contiene “L'utente Pallino ha modificato questa pagina” //s contiene ancora “L'utente Pinco ha modificato questa pagina”

Se si desidera sostituire tutte le occorrenze si può usare questa funzione:

function replace (str, first, last) { while (str.indexOf(first) != −1) { str = str.replace(first, last); } return str; }

con la sintassi replace('torta', 't', 'f') (che restituisce "*forfa*"). Oppure si può usare un'espressione regolare:

var s = “L'utente Pinco ha modificato questa Pinco pagina” var s2 = s.replace(/Pinco/g, “Pallino”); //s2 ora contiene “L'utente Pallino ha modificato questa Pallino pagina” //s contiene ancora “L'utente Pinco ha modificato questa Pinco pagina”

32.7 split()

Il metodo split() viene presentato ora in quanto fa parte della classe String, ma il suo uso richiede la conoscenza degli array, che verranno esaminati più avanti nel corso del libro, basti sapere che un array è una sorta di contenitore di variabili.
La sintassi è:

stringa.split(separatore);

Il metodo restituisce un array contenente le diverse sottostringhe in cui la stringa è divisa dal separatore. Ad esempio:

“a,e,i,o,u”.split(","); //restituisce “a”, “e”, “i”, “o”, “u” “a,e,i,o,u,”.split(","); //attenzione: restituisce “a”, “e”, “i”, “o”, “u”, ""

33 Esercizi

- Scrivere una funzione che conti le occorrenze di una sottostringa in una stringa (ad esempio se vengono passati come parametri “raffaele” e “a” deve restituire 2, ma se si inseriscono “raffaele” e “ff” deve restituire 1.

Soluzione
function subStrCount (string, subString) { var stringa = string, count = 0; while (stringa.indexOf(subString) != −1) { stringa = stringa.replace(subString, ""); count++; } return count; }
La funzione elimina tutte le occorrenze della sottostringa nella stringa tramite il metodo replace(); poi sottrae la lunghezza della stringa principale a quella della stringa appena ottenuta; in questo modo si ottiene il numero di caratteri occupati dalla sottostringa che, divisi per la sua lunghezza, ne danno il numero di occorrenze.

- Creare una funzione che fornisca la sottostringa di una stringa data compresa tra due altre sottostringhe, a partire da sinistra. Ad esempio, dati come parametri una@bella&stringa, "@" e "&", restituisca “bella”.

Soluzione
function substrChr (string, chr1, chr2) { chr1Pos = string.indexOf(chr1); chr2Pos = string.indexOf(chr2); return string.substr(chr1Pos +1, chr2Pos - chr1Pos - 1); }

L'oggetto **Math** mette a disposizione dello sviluppatore JavaScript numerose funzioni e costanti matematiche; è un oggetto particolare, in quanto non richiede di lavorare su istanze di oggetti ma permette di accedere ai suoi metodi usando la notazione:

Math.nome_metodo(); Math.nome_proprietà

34 Proprietà

Le proprietà dell'oggetto Math consistono in diverse costanti matematiche:

- PI: restituisce il valore approssimato di Pi greco
- E: restituisce il valore della costante matematica E
- LN2 e LN10: il valore del logaritmo naturale di 2 e di 10

Ovviamente sono tutte proprietà di sola lettura

35 Metodi

Ecco una lista dei metodi corrispondenti alle principali funzioni matematiche e trigonometriche:

- abs(): restituisce il valore assoluto (o modulo) del numero fornito come argomento
- min() e max(): restituiscono il minimo e il massimo tra i due numeri passati come parametri
- sqrt() e pow(b, *n*): restituiscono la radice quadrata del numero passato o l'nesima potenza del numero b
- sin(), cos() e tan(): restituiscono il seno, il coseno e la tangente trigonometrica dell'angolo passato in radianti
- log(): restituisce il logaritmo naturale in base e del numero passato come parametro

35.1 Arrotondare i numeri

Per l'arrotondamento l'oggetto Math mette a disposizione tre metodi:

- ceil(): arrotonda il numero al numero intero maggiore più vicino (per eccesso). Ad esempio:

Math.ceil(10.01) //restituisce 11 Math.ceil(-6.3) //restituisce -6

- floor(): arrotonda l'oggetto al numero intero minore più vicino (per difetto). Ad esempio:

Math.floor(10.01) //restituisce 10 Math.floor(-6.3) //restituisce -7

- round(): arrotonda l'oggetto per eccesso se la parte decimale è maggiore o uguale a 5, altrimenti arrotonda per difetto:

Math.round(10.01) //restituisce 10 Math.round(−6.3) //restituisce −6 Math.round(2.5) //restituisce 3

Tutte queste tre funzioni differiscono da parseInt() che non arrotonda i numeri, bensì li tronca. Ad esempio:

Math.parseInt(10.01) //restituisce 10 Math.parseInt(−6.3) //restituisce −6 Math.parseInt(2.5) //restituisce 2

35.2 Generare numeri casuali

L'uso di Math permette anche di generare numeri (pseudo) casuali tramite il metodo:

Math.random()

che restituisce un numero casuale decimale compreso tra 0 (incluso) e 1 (escluso). Nella pratica non è molto utile se non viene trasformato in un numero intero. Ad esempio questo codice simula il lancio di un dado:

var numLanci = prompt (“Quanti lanci vuoi effettuare?", 10); var i; var lancio; for (i = 0; i < numLanci; i++) { lancio = Math.floor(Math.random() * 6 + 1); alert(“Lancio " + (i + 1) + ": " + lancio); }

La riga che ci interessa è quella dell'assegnazione della variabile lancio. Per capire l'algoritmo usato, ragioniamo sul risultato che vogliamo ottenere.
Il numero che vogliamo ottenere (che chiameremo *d*) deve essere un numero intero compreso tra 1 e 6 inclusi. Dal momento che il generatore restituisce numeri tra 0 incluso e 1 escluso, dovremo:

- moltiplicare per sei il numero casuale ottenuto (che chiameremo *n*): in questo modo possiamo ottenere un numero che va da 0 (se $n = 0$, allora $d = 0$) fino a sei escluso. Infatti *d* potrebbe essere uguale a 6 solo se *n* potesse essere uguale a 1, cosa che non è possibile
- sommare uno al valore ottenuto: in questo modo *d* sarà compreso tra 1 (incluso) e 7 (escluso)
- arrotondare il valore per difetto: in questo modo *d* sarà un numero intero compreso tra 1 e 6; infatti, dato che il 7 è escluso, arrotondando per difetto potremo ottenere al massimo 6.

36 Un esempio pratico

Di seguito viene mostrata un'applicazione pratica dei metodi dell'oggetto Math per la creazione di una funzione che arrotondi le cifre alla *n*esima cifra decimale indicata.

function arrotonda (num, dec, mod) { var div = Math.pow(10, dec); switch (mod) { case 'e': case 'E': //arrotonda per eccesso return Math.ceil(num * div) / div; break; case 'd': case 'D': //arrotonda per difetto return Math.floor(num * div) / div; break; case 't': case 'T': //troncamento return parseInt(num * div) / div; break; case 'a': case 'A': default: //arrotonda return Math.round(num * div) / div; } } //per testarla usiamo queste righe di codice... alert(arrotonda(3.43, 1, 't')); alert(arrotonda(3.42, 1, 'e')); alert(arrotonda(3.469, 2, 'd')); alert(arrotonda(3.427, 2, 'e')); alert(arrotonda(3.55, 1, 'a')); alert(arrotonda(3.46, 1, 't'));

La funzione appena mostrata arrotonda il numero fornito come primo parametro alla cifra decimale fornita come secondo parametro usando diversi metodi di arrotondamento. Il tutto sta nel fatto che le funzioni di Math agiscono solo sui numeri interi e non su quelli decimali.
Per effettuare gli arrotondamenti, quindi, spostiamo la virgola di *n* posizioni decimali fornite come parametro ed eseguiamo l'arrotondamento desiderato; successivamente, spostiamo di nuovo la virgola all'indietro. Per spostare la virgola è sufficiente moltiplicare e dividere per una potenza di dieci: per fare ciò ci serviamo del metodo Math.pow()

L'oggetto **Date** permette di gestire in modo semplice e veloce le date; nella pratica, un'istanza della classe Date identifica una data e le funzionalità della classe permettono di aggiungere o sottrarre anni, mesi o giorni o recuperarne il valore.

37 Costruttore

Il metodo più semplice per creare una data è:

var data = new Date()

Nel caso non si passino valori al costruttore, verrà creata un oggetto Date contenente la data e l'ora corrente. Altrimenti possiamo creare date precise con altri metodi:

- fornendo il numero di millisecondi passati dalla mezzanotte del primo gennaio 1970 (questo metodo è macchinoso ma è quello che più si avvicina al metodo con cui il computer ragiona con le date). Questo valore è chiamato *timestamp*.
- fornendo una stringa che indichi la data, ad esempio:

var data = new Date(“10 July 2007”); var data2 = new Date(“10-7-2007”);

> Usando il metodo descritto nel secondo esempio, bisogna fare attenzione in quanto, essendo JavaScript eseguito con le impostazioni del client, per un utente USA invece che il 10 luglio 2007 la data sarà 7 ottobre 2007

- l'ultimo metodo è il migliore in quanto affidabilità e praticità messe assieme. La sintassi è:

var data = new Date(anno, mese, giorno, ora, minuti, secondi, millisecondi)

> Crea un oggetto data in base ai parametri passati; normalmente è possibile omettere l'ultimo parametro. Attenzione, gennaio è il mese 0 e non 1 e di conseguenza anche gli altri cambiano.

38 Metodi

38.1 Recuperare i valori della data

- getDay(): restituisce un numero da 0 a 6 corrispondente al giorno della settimana della data (Domenica è 0, Lunedì 1, ... Sabato 6)
- getDate(): restituisce il giorno del mese
- getMonth(): restituisce il numero del mese (Gennaio è 0, Dicembre è 11)
- getFullYear(): restituisce il numero dell'anno con quattro cifre.

Visti questi metodi, possiamo già creare una funzione che restituisca una stringa contente la data formattata. La funzione prende come parametro un oggetto Date e formatta la data in esso contenuta:

function formattaData(data) { var giornoS = data.getDay(); var giornoM = data.getDate(); var mese = data.getMonth(); var anno = data.getFullYear(); switch (giornoS) { //converte il numero in nome del giorno case 0: //domenica giornoS = “domenica"; break; case 1: giornoS = “lunedì"; break; case 2: giornoS = “martedì"; break; case 3: giornoS = “mercoledì"; break; case 4: giornoS = “giovedì"; break; case 5: giornoS = “venerdì"; break; case 6: //sabato giornoS = “sabato"; break; } switch (mese) { //converte il numero in nome del mese case 0: mese = “gennaio"; break; case 1: mese = “febbraio"; break; case 2: mese = “marzo"; break; case 3: mese = “aprile"; break; case 4: mese = “maggio"; break; case 5: mese = “giugno"; break; case 6: mese = “luglio"; break; case 7: mese = “agosto"; break; case 8: mese = “settembre"; break; case 9: mese = “ottobre"; break; case 10: mese = “novembre"; break; case 11: mese = “dicembre"; break; } //es. martedì 9 luglio 2007 return giornoS + " " + giornoM + " " + mese + " " + anno; }

Possiamo usare la funzione in questi modi:

var d1 = new Date () //oggi var d2 = new Date (2007, 6, 10); document.write(formattaData(d1)); document.write(formattaData(d2));

La funzione potrebbe sembrare inutile data l'esistenza del metodo dell'oggetto Date toLocaleString() che restituisce una stringa contenente la data formattata secondo le impostazioni locali dell'utente; tuttavia, per questo motivo, ad esempio, un utente inglese invece che martedì 10 luglio 2007 otterrebbe Tuesday, July 10, 2007 00:00:00.

38.2 Impostare i valori della data

L'oggetto Date permette inoltre di modificare la data memorizzata nell'oggetto Date tramite i metodi setDate(), setMonth e setFullYear. Questi metodi sono molto utili, perché hanno la caratteristica di ragionare come è giusto fare per le date. Ad esempio:

var d = new Date(2007, 3, 4); //4 Aprile 2007 d.setDate(31);

In questo esempio impostiamo come giorno del mese di aprile il valore di 31, nonostante aprile conti solo 30 giorni. JavaScript lo sa e per questo porta la data al 1 di maggio. Questa funzionalità è spesso usata insieme ai metodi per ottenere le date:

d.setMonth(d.getMonth() - 6); //sei mesi prima

38.3 Lavorare con l'ora

L'oggetto Date mette a disposizione funzioni analoghe anche per gestire le ore:

- getHours(), getMinutes(), getSeconds() e getMilliseconds restituiscono rispettivamente l'ora, i minuti, i secondi e i millisecondi dell'oggetto Date. Ad esempio:

var d = new Date(2007, 3, 4, 23, 45, 6, 12); //4 aprile 2007, ore 23:45:6 (11 pm) d.getSeconds(); //restituisce 6

Il metodo getMilliseconds permette anche di cronometrare l'esecuzione di una porzione di codice o il caricamento della pagina HTML. Inseriamo in testa alle istruzioni da cronometrare il codice:

var dataPrima = new Date(); //adesso //memorizza secondi e millisecondi var sPrima = dataPrima.getSeconds(); var msPrima = dataPrima.getMilliseconds();

In questo modo salviamo i secondi e i millisecondi del momento in cui inizia il caricamento; poi inseriamo in fondo alla pagina:

var dataDopo = new Date(); //adesso //memorizza secondi e millisecondi var sDopo = dataDopo.getSeconds(); var msDopo = dataDopo.getMilliseconds(); document.write('Pagina caricata in ' + (sDopo - sPrima) + ' secondi e ' + Math.abs(msDopo - msPrima) + ' millisecondi');

38.4 Gestire i fusi orari

Tutte le funzioni che abbiamo visto lavorano sull'ora dell'utente o su quella impostata dalla pagina web. Non abbiamo però considerato che Internet è una rete che coinvolge centinaia di paesi e quasi tutti i fusi orari del nostro pianeta; per questo motivo JavaScript mette a disposizione delle funzioni basate sull'**UTC** (**Universal Time Coordination**), una convenzione precentemente nota come **GMT** (**Greenwich Mean Time**) che rappresenta la data basandosi su quella del meridiano 0, passante per la famosa località di Greenwich. Ovviamente anche queste funzioni sono basate su una corretta configurazione del computer del client.

I metodi UTC ricalcano i metodi per il locale; per impostare l'ora UTC è necessario ad esempio usare il metodo set**UTC**Hour; esiste inoltre un metodo toUTCString che restituisce l'ora formattata secondo le convenzioni UTC e un metodo getTimeZoneOffset che restituisce in minuti la differenza di ora tra il fuso di Greenwich e quello dell'utente (può essere anche negativo).

Last but not least esaminiamo gli array, una funzionalità presente nella maggior parte dei linguaggi di alto livello, che in JavaScript è disponibile tramite l'uso della classe Array.

39 Cos'è un array

In informatica, un array (o vettore) è un tipo di dato strutturato (non semplice, che dispone quindi di altri parametri) che permette di organizzare i dati secondo un indice; in pratica, lo stesso oggetto contiene a sua volta numerosi valori, ciascuno dei quali contrassegnato da una chiave numerica. Questo concetto può essere rappresentato da un tabella a singola entrata:

40 Gli array in JavaScript

Come è stato già detto, per creare un vettore in JavaScript facciamo riferimento alla classe Array:

var vettore = new Array (); //crea un array vuoto

Il costruttore della classe può essere tuttavia usato in maniere differenti:

var vettore = new Array (5); //crea un array contenente 5 elementi var vocali = new Array (“A”, “E”, “I”, “O”, “U”); //crea un array contente le vocali var lettere_straniere = ["J”, “K”, “W”, “X”, “Y"]; //metodo breve

Per accedere ad un elemento dell'array, per leggerlo o modificarlo, usiamo la notazione:

vettore[indice]

Ad esempio, facendo riferimento alla variabile vocali, avremo questa tabella:

e potremo lavorare in questo modo:

alert(vocali[0]); //mostra “A” vocali[1] = “ciao” alert(vocali[1]); //mostra “ciao”

Da notare che la chiave dell'array è numerica e parte da 0 e che il valore di ciascun elemento può essere di qualsiasi tipo di dato, ciascuno diverso dall'altro.
Ad esempio, potremmo stabilire un metodo per memorizzare le informazioni su dei prodotti decidendo che all'elemento 0 corrisponde il nome, all'elemento 1 il modello, ecc... creando diversi array:

var descCampi = new Array (“ID”, “Modello”, “Prezzo”) var prod1 = new Array (1, “3000 plus”, 35); var prod2 = new Array (5, “4000 minus”, 12);

40.1 Array associativi?

Javascript **non** supporta i vettori con indici associativi. Al suo posto si usano gli oggetti. Se si tenta di creare un array associativo, il parser lo interpreta come un oggetto con un indice denominato. Quindi i metodi degli array non funzioneranno in modo corretto. Ad esempio:

var vet = new Array(); vet["pippo"] = “valore1"; alert(vet.length); //output: 0 /* length è una proprietà che si applica solo agli array con indice numerico, quindi dà un valore errato */

In JavaScript gli array hanno solo indici numerici.

40.2 Array multi-dimensionali

Per inizializzare un vettore multi-dimensionale (vettore di vettori) possiamo fare cosi:

var multivettore=new Array(new Array(“A1”,"A2”),new Array(“B1”,"B2")) alert(multivettore[0][0]); //output: A1 alert(multivettore[0][1]); //output: A2 alert(multivettore[1][0]); //output: B1 alert(multivettore[1][1]); //output: B2

41 Proprietà

L'oggetto Array dispone di una sola proprietà interessante, length, che restituisce il numero di elementi in esso contenuti (contando anche quelli vuoti); da notare che, poiché gli indici partono da 0, un array di 5 elementi avrà gli elementi con le chiavi 0, 1, 2, 3 e 4.

42 Metodi

L'uso dell'oggetto Array diventa utile in relazione all'uso dei metodi.

42.1 concat()

Il metodo concat() restituisce un altro array contente tutti gli elementi dell'array a cui è applicato seguiti da tutti gli elementi dell'array passato come parametro. Ad esempio:

var a = new Array (1, 2, 3); var b = new Array (4, 5, 6); var c = a.concat(b); //1, 2, 3, 4, 5, 6

Da notare che a contiene ancora solo gli elementi 1, 2 e 3.

42.2 push()

Il metodo push() inserisce l'elemento indicato come parametro come ultimo elemento dell'array e restituisce la nuova lunghezza dell'array. Ad esempio:

var a = new Array (1, 2, 3); var b = a.push(7); alert(b); //4

Da notare che l'array a contiene ora gli elementi 1, 2, 3 e 7

42.3 pop() e shift()

I metodi pop() e shift() eliminano rispettivamente l'ultimo e il primo elemento dell'array e restituiscono il valore dell'elemento eliminato:

var a = new Array (1, 2, 3); var b = a.pop(); //ora a contiene 1 e 2 alert(b); //3

42.4 sort()

Il metodo sort() ordina l'array secondo l'ordine alfabetico:

var a = new Array (“e”, “a”, “u”, “i”, “o”); a.sort() //ora le vocali sono ordinate nell'array a //attenzione: lavora direttamente sull'array!

Attenzione: in JavaScript le minuscole seguono alle maiuscole, quindi ad esempio

var a = new Array (“E”, “a”, “U”, “i”, “o”); a.sort() //ora le vocali sono ordinate in questo modo: //E U a i o!

Per ovviare a questo problema possiamo lavorare su stringhe solo minuscole, usando il metodo toLowerCase in questo modo:

var a = new Array (“E”, “a”, “U”, “i”, “o”); //il nostro array for (var i = 0; i < a.length; i++) { //itera sui singoli elementi... a[i] = a[i].toLowerCase(); //...rendendoli minuscoli }

42.5 reverse()

Questo metodo agisce sull'array invertendo l'ordine degli elementi in esso contenuti. Ad esempio:

var vocali = new Array (“A”, “E”, “I”, “O”, “U”); vocali.reverse(); //ora contiene U O I E A

42.6 slice()

Il metodo slice(), infine, serve per copiare porzioni di array. La sua sintassi è:

arr.slice(a, b)

Il metodo restituisce un array contente gli elementi di arr compresi tra a (incluso) e b (escluso). Se b non è indicato, vengono copiati tutti gli elementi a partire da a fino alla fine. Ad esempio:

var a = new Array (“A”, “E”, “I”, “O”, “U”); var b = a.slice(1,4); //b contiene gli elementi E I O

43 Iterare sugli elementi di un array

Come abbiamo visto precedentemente, è possibile iterare su tutti gli elementi di un array con un ciclo particolare; ad esempio, volendo azzerare il valore di tutti gli elementi di un array, possiamo usare il codice:

var a = new Array (“A”, “E”, “I”, “O”, “U”); for (indice in a) { a[indice] = ""; //annulla il valore } //ora l'array contiene 5 elementi tutti vuoti

43.1 Cercare un elemento in un array

Di seguito è mostrata a titolo di esempio una funzione arrayIndexOf() che restituisce la posizione di un elemento dato in un array. Nel caso l'elemento non sia presente, restituisce -1.

function arrayIndexOf(array, search) { var indice; //contatore for (indice in array) { if (array[indice] == search) return indice; } return -1; }

Il ciclo scorre su tutti gli elementi dell'array: se l'elemento corrente corrisponde a quello cercato (array[indice] == search) la funzione esce restituendo come valore l'indice corrente (return indice;). L'ultima istruzione (return -1;) verrà quindi eseguita se non verrà trovata nessuna corrispondenza nell'array (che è quindi il risultato che si voleva ottenere). La funzione presentate è una ricerca lineare, nel caso di array ordinati si può utilizzare anche l'algoritmo di ricerca binaria.

Una funzionalità molto interessante messa a disposizione da JavaScript è la possibilità di **gestire i cookie**.
In informatica, i cookie sono piccoli file di testo memorizzati sul computer dell'utente che contengono informazioni salvate dai siti web. La comodità dei cookie sta quindi nella possibilità di memorizzare in modo **permanente** delle informazioni univoche rispetto ad un utente; hanno tuttavia lo svantaggio di poter essere eliminati e disabilitati dall'utente.
Ciascun cookie è composto da alcuni parametri, tra i quali:

- **nome**: un nome identificativo per il cookie
- **valore**: il valore da memorizzare

- **scadenza** (*expiration date*): è opzionale, stabilisce la data di scadenza del cookie, cioè la data dopo la quale questi vengono eliminati dal disco rigido dell'utente.

44 Impostare i cookie

```
12 function impostaCookie (nome, valore, percorso, scadenza) {
13    valore=escape(valore);
14
15    if (scadenza == "") {
16       var oggi = new Date();
17       oggi.setMonth(oggi.getMonth()+6);
18       scadenza=oggi.toGMTString();
19    }
20    if (percorso!="")
21       percorso= ";Path=" + percorso;
22
23    document.cookie = nome + "=" + valore + ";expires=" + scadenza + percorso;
24 }
25
```

la funzione ImpostaCookie visualizzata dall'editor Bluefish

In JavaScript per impostare i cookie usiamo la proprietà cookie dell'oggetto document passandole come valore una stringa contenente i vari parametri separati da punto e virgola. Ecco ad esempio una semplice stringa di cookie:

NomeUtente=Ramac;expires=Tue, 28 August 2007 00:00:00

Questa stringa imposta un cookie NomeUtente al valore Ramac con scandenza 28 agosto 2007. Potremo salvare questo cookie con l'istruzione

document.cookie="NomeUtente=Ramac;expires=Tue, 28 August 2007 00:00:00”

La sintassi è quindi:

NomeCookie=Valore;parametri

Nel caso si volesse inserire nella stringa del valore un punto e virgola, è necessario usare la funzione escape() che converte tutti i caratteri particolari nel corrispondente esadecimale nel set di caratteri Latin-1. Questo vale anche per gli spazi, gli apostrofi, le virgole, ecc....

Per comodità, è possibile creare una funzione che crei i cookie partendo da tre parametri:

function impostaCookie (nome, valore, scadenza) { if (scadenza == "") { var oggi = new Date(); oggi.setMonth(oggi.getMonth() + 3); //restituisce la data nel formato necessario scadenza = oggi.toGMTString(); } valore = escape(valore); document.cookie=nome + "=" + valore + ";expires=" + scadenza; }

45 Ottenere i cookie

Per ottenere i valori dei cookie relativi al proprio documento è necessario accedere alla proprietà document.cookie che restituisce una stringa contenente coppie di nome-valore di ciascun dato separate da un punto e virgola e uno spazio. Ad esempio:

cookie1=value1; cookie2=value2; cookie3=value3;

Possiamo notare due punti importanti:

- mentre è possibile impostare valori come la data di scadenza del cookie, non è possibile ottenerli una volta modificati
- i dati per una comoda lettura necessitano di una manipolazione stringa

Per la lettura dei cookie, può risultare utile questa funzione che restituisce il valore di un cookie a partire dal suo nome:

function valoreCookie (nome) { var valore=document.cookie; //ottiene la stringa di cookie var inizioCookie=valore.indexOf(" " + nome + "="); //trova il cookie desiderato //se non esiste, magari è all'inizio della stringa if (inizioCookie == −1) { inizioCookie = valore.indexOf(nome + "="); } if (inizioCookie == −1) { //il cookie non esiste proprio valore = null; } if (inizioCookie >= 0) { //il cookie esiste //qui inizia la stringa del valore inizioCookie = valore.indexOf("=", inizioCookie) + 1; var fineCookie = valore.indexOf(";", inizioCookie); //qui finisce if (fineCookie == −1) //se non viene trovato, allora è l'ultimo cookie fineCookie = valore.length; //elimina i caratteri commutati valore = unescape(valore.substring(inizioCookie, fineCookie)); } return valore; }

46 Verificare se i cookie sono attivi

Come è già stato accennato in precedenza, uno degli svantaggi dei cookie risiede nel fatto che **possono essere facilmente disabilitati** dall'utente; è pertanto necessario sviluppare soluzioni alternative ai cookie, soprattutto se sono parte essenziale della propria applicazione web.

Il modo più semplice per verificare se i cookie sono attivati è quello di crearne uno fittizio e verificare se è possibile ottenerne il valore. La seguente è una funzione che restituisce true se i cookie sono abilitati:

function cookieAttivi () { ris = false; //imposta il risultato a falso impostaCookie(“testCookie”, “test”); //crea il cookie fittizio if (valoreCookie(“testCookie”) == “test”) { //se esiste ris = true; //allora i cookie sono abilitati } return ris; }

47 Eliminare o sostituire un cookie

Poiché ad ogni nome corrisponde un solo cookie, per modificare il contenuto di un cookie chiamato NomeUtente è sufficiente la riga

impostaCookie(NomeUtente, nuovo_contenuto, nuova_scadenza);

Per eliminare un cookie, invece, basta sostituire al testo del cookie una stringa vuota:

impostaCookie(NomeUtente, "");

48 Librerie JavaScript per i cookie

Puoi utilizzare una libreria JavaScript per facilitare la gestione dei cookie, come js-cookie, o altre.

Un'interessante funzionalità offerta da JavaScript nonché indispensabile per rendere una pagina dinamica in molte occasioni, è quella dei cosidetti ***timer***.
Grazie ad essi possiamo far effettuare un'istruzione ad intervalli regolari o dopo un numero prestabilito di secondi.

49 One-shot timer

Per **one-shot timer**, o **timeout**, si intendono quei timer che permettono di eseguire un'istruzione dopo un determinato numero di millisecondi. Il metodo setTimeout dell'oggetto window prende come parametri una stringa contenente l'istruzione da eseguire e il numero di secondi dopo i quali deve essere eseguita. Ad esempio:

var tim = setTimeout('alert(“ciao!")', 3000); //l'oggetto window può essere sottinteso

Inserendo questa istruzione al caricamento della pagina, dopo 3 secondi (3000 millisecondi) verrà visualizzato un messaggio di alert.

Attenzione: i timeout **non interrompono** il normale flusso degli script. Ad esempio:

var tim = setTimeout('alert(“ciao!")', 3000); alert(“secondo alert”);

In questo caso verrà mostrato prima il secondo alert in quanto l'esecuzione del primo è ritardata di tre secondi.
Nella stringa è possibile anche inserire una funzione, ovviamente.

Il metodo setTimeout restituisce un ID numerico che identifica univocamente ciascun timer impostato; questo è utile in relazione al metodo clearTimeout() che elimina il timeout impostato con il metodo visto precedentemente. Questo può servire per permettere all'utente di fermare un timer già iniziato. Ad esempio:

var tim = setTimeout('alert(“la bomba è esplosa”)', 3000); //la bomba esploderà tra 3 secondi var pwd = prompt(“se inserisci la password corretta potrai disinnescare la bomba prima che esploda....”); if (pwd == “disattiva”) { clearTimeout(tim); //disinnesca il timer (la “bomba”) }

50 Impostare intervalli regolari

Diversi dai timeout sono quei timer che impostano azioni da eseguire ad intervalli regolari. Ad esempio:

var tim = setInterval (“aggiornaPagina()", 1000);

Con questa istruzione facciamo sì che la funzione aggiornaPagina venga eseguita ogni secondo. Come per i timeout, possiamo eliminare i timer ricorrendo ad un'apposita funzione:

clearInterval(tim);

che interrompe l'esecuzione del timer precedentemente avviato.

Arriviamo finalmente a scoprire le funzionalità di JavaScript che lo rendono il linguaggio di scripting per le pagine web tra i più diffusi: la possibilità di interagire con il browser e la pagina HTML.

Tutto questo è possibile grazie all'**oggetto window** e, più generalmente, a quello che è il ***BOM*** (***Browser Object Model***), il *modello oggetto del browser*: per esso si intendono l'oggetto window tutti gli oggetti che da essi dipendono.

51 L'oggetto window

L'oggetto window rappresenta la finestra del browser che contiene la pagina stessa; nella pratica, questo vuol dire avere accesso, per esempio, alle dimensioni della finestra del browser, al testo della sua barra di stato, e molto altro.

L'oggetto window è un **oggetto globale**, il che significa che non è necessario specificarlo per utilizzare i suoi metodi: ne è un esempio l'uso che abbiamo sempre fatto del metodo window.alert(), che abbiamo sempre richiamato senza specificare l'oggetto al quale apparteneva.

51.1 Proprietà

51.1.1 defaultStatus e status

Questa proprietà permette di ottenere o di impostare il testo predefinito della barra di stato del browser. Per esempio, sulla maggior parte dei browser, mostra la scritta “Completato” dopo l'avvenuto caricamento di una pagina.
Per usarla vi accediamo semplicemente:

window.defaultStatus = “Testo personalizzato"; //oppure anche defaultStatus = “Testo personalizzato";

La seconda proprietà, status, indica invece il testo *corrente* nella barra di stato, ad esempio l'href di un link ipertestuale se vi sta passando sopra il mouse. Anche qui la sintassi è semplice:

window.status = “Testo personalizzato";

51.1.2 frames, length e parent

Queste tre proprietà sono legate all'uso di *frame* nelle pagine web.

Essendo l'uso dei frame una tecnica piuttosto obsoleta, sorpassata oramai da tecnologie lato server e deprecata dal W3C, il loro uso non sarà approfondito nel corso di questo libro.

51.1.3 opener

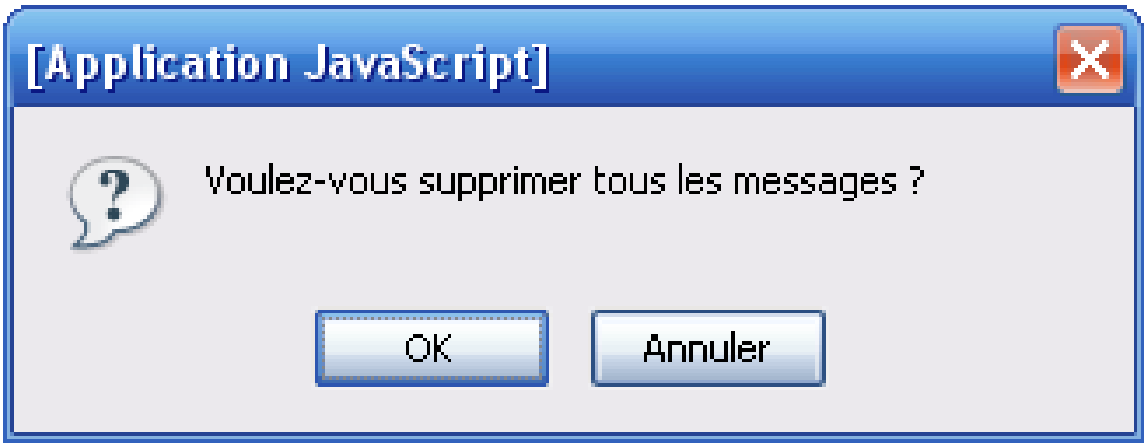
Questa proprietà restituisce un riferimento all'oggetto window che ha aperto la finestra corrente (si veda più avanti il metodo open()).

51.1.4 Oggetti come proprietà

La maggior parte delle proprietà dell'oggetto window sono a loro volta altri oggetti. I più importanti e utilizzati saranno trattati nel dettaglio più avanti in questo modulo e nei prossimi capitoli.

51.2 Metodi

51.2.1 alert(), confirm() e prompt()



Una finestra di confirm, questa volta in francese

Due di questi metodi, alert() e prompt(), sono già stati trattati in precedenza; è simile invece l'uso del metodo confirm(), in quanto anch'esso mostra una finestra di dialogo. La sua sintassi è:

window.confirm(“testo”);

Il metodo mostra una finestra di dialogo che mostra il testo indicato come parametro e due pulsanti, *OK* e *Annulla*), e restituisce true se l'utente clicca su OK, false negli altri casi.
Questo metodo serve per chiedere conferme all'utente prima di effettuare operazioni lunghe o importanti. Siccome un sito internet viene potenzialmente visitato da utenti di tutto il mondo, il testo sui pulsanti varia a secondo della lingua usata dal browser quindi *Annulla* in francese diventa *Annuller*, in inglese *Cancel* e via dicendo.

51.2.2 blur() e focus()

Questi due metodi rispettivamente tolgono il focus dalla finestra del browser e spostano il focus sul browser. Questo serve quando ad esempio sono aperte più finestre contemporaneamente.

51.2.3 resizeTo() e resizeBy()

Questi due metodi servono per ridimensionare la finestra.

Il primo è assoluto e richiede due valori interi positivi che indicano la larghezza e l'altezza che si vogliono ottenere.

Il secondo metodo invece è relativo alle dimensioni correnti della finestra e accetta due valori interi positivi e negativi che indicano di quanto si voglia ingrandire o rimpicciolire la finestra.

//rimpicciolisce la finestra del browser window.resizeTo(200, 200);

51.2.4 moveTo() e moveBy()

Questi metodi servono per spostare a proprio piacere la finestra del browser sullo schermo.

Con il primo metodo è possibile specificare le coordinate dello schermo da impostare come posizione dell'angolo superiore sinistro della finestra

Il secondo metodo richiede un valore relativo che indichi di quanto spostare la finestra in orizzontale e in verticale.

Attenzione: lo spostamento della finestra porterà al ripristino delle dimensioni della finestra se questa si trova ingrandita a tutto schermo.

//sposta la finestra del browser nell'angolo superiore window.moveTo(0,0);

51.2.5 scrollTo() e scrollBy()

Questi due metodi servono per effettuare lo *scrolling* della pagina.

Il primo metodo richiede le coordinate x e y della finestra del punto che si desidera visualizzare con lo scrolling.

Il secondo metodo è relativo e richiede un numero positivo o negativo che indichi di quanto scrollare la pagina in orizzontale o in verticale

//sale di 200 px window.scrollBy(−200);

51.2.6 print()

Questo metodo serve per stampare la pagina corrente e non richiede parametri.

51.2.7 open()

Questo metodo apre una nuova istanza del browser, apre cioè una nuova finestra del browser in uso. La sua sintassi è:

window.open(“url”, “nome”, “parametri”)

Il primo parametro è l'indirizzo della pagina che sarà caricata nella nuova finestra; passando una stringa vuota si otterrà una pagina vuota (in pratica, viene aperta about:blank).

Il secondo parametro indica il nome della finestra che potrà essere usato ad esempio come valore per l'attributo target dei link o dei moduli HTML o ancora da altre istruzioni JavaScript.

Il terzo parametro è invece una stringa di parametri costruita con la sintassi:

nomeParametro=valore,nomeParametro2=valore2,

I principali parametri sono (non sono tutti compatibili con tutti i browser):

La funzione restituisce un riferimento all'oggetto window appena aperto. Ad esempio:

finestra = window.open(“pagina.html”, “finestra1”, “toolbar=no,location=yes”); finestra.defaultStatus = “Benvenuti nella nuova finestra!";

Potremo poi usare il nome finestra1 nel codice HTML:

Link

Possiamo anche creare una generica funzione, per poi creare nuove finestre partendo dai link:

function apriFinestra(url, nome, width, height) { var params = “width:" + width + ";height=" + height; window.open(url, nome, params); }

e poi per usarla la inseriamo per esempio in un link:

Apri

In questo modo, grazie alla sintassi “javascript:*istruzione*", cliccando sul link verrà eseguita la funzione appena creata.

Dal codice della finestra aperta si potrà ottenere un riferimento alla finestra che la ha aperta tramite la proprietà window.opener.

51.2.8 close()

Chiude una finestra del browser.

Prima di completare la descrizione degli oggetti del BOM, ci occuperemo finalmente di un modo per rendere la nostra pagina HTML veramente dinamica: l'utilizzo degli **eventi**.

Un evento in JavaScript permette di eseguire determinate porzioni di codice a seconda delle **azioni dell'utente** o a **stati della pagina**: è possibile ad esempio collegare del codice al *click* di un oggetto oppure al ridimensionamento della pagina.

52 Gli eventi in JavaScript

Di seguito è elencata una lista dei possibili eventi verificabili in JavaScript; da notare che gli eventi si riferiscono sempre ad un oggetto specifico (successivamente sarà spiegato come).

52.1 Eventi del mouse

- **onclick** e **ondblclick** si verificano quando l'utente fa click o doppio click sull'elemento in questione
- **onmousedown** e **onmouseup** si verificano quando l'utente schiaccia il pulsante del mouse e quando lo rilascia
- **onmouseover** e **onmouseout** si verificano quando l'utente sposta il mouse dentro l'oggetto in questione e quando lo sposta fuori
- **onmousemove** si verifica quando l'utente muove il cursore dentro l'oggetto

52.2 Eventi della tastiera

- **onkeypress** si verifica quando l'utente preme un tasto sulla tastiera quando il *focus* è sull'oggetto specificato
- **onkeydown** e **onkeyup** si verificano quando l'utente schiaccia un tasto e lo rilascia

52.3 Eventi degli elementi HTML

- **onfocus** e **onblur** si verificano quando l'utente sposta il focus sull'oggetto in questione e quando toglie il focus dall'oggetto

- **onchange** si verifica quando l'utente modifica il contenuto dell'oggetto (è applicabile solo ai campi di modulo)
- **onsubmit** e **onreset** si possono applicare solo a moduli HTML e si verificano quando l'utente invia il form (pulsante di *submit*) o quando lo resetta (pulsante *reset*)
- **onselect** si verifica quando l'utente seleziona del testo

52.4 Eventi della finestra

- **onload** e **onunload** si verificano quando la pagina viene caricata o quando viene chiusa
- **onresize** si verifica quando l'utente ridimensiona la finestra del *browser*
- **onscroll** si verifica allo scrolling della pagina

53 Eventi come attributi HTML

Il modo più semplice di definire un evento è quello di inserirlo come **attributo agli elementi HTML** della pagina stessa. Ad esempio:

Nell'esempio associamo il codice JavaScript che mostra l'alert al click dell'utente sull'immagine HTML.

Ovviamente possiamo anche fare riferimento a funzioni o variabili dichiarate precedentemente nel corso della pagina HTML:

Tramite questo sistema possiamo usufruire dell'oggetto this per riferirci all'elemento HTML a cui è stato applicato. Ad esempio:

Con questa riga di HTML e qualche istruzione JavaScript creiamo quello che si chiama un *rollover*, cioè l'effetto di cambiamento dell'immagine quando il mouse passa sopra di essa.

54 Eventi come proprietà

Gli eventi possono essere anche impostati **tramite il BOM**, cioè tramite JavaScript. Ad esempio:

window.onload = “alert('pagina in caricamento...');";

Questo può risultare comodo per aggiungere eventi quando non si possono modificare direttamente gli oggetto HTML, ad esempio nel caso di script esterni creati per essere utilizzati più volte.

Attenzione a non incappare in questo tipo di errori:

function saluta () { alert(“pagina in caricamento”); }; window.onload = saluta(); //non funziona

In questo codice l'intenzione è quella di mostrare un avviso al caricamento della pagina; scrivendo saluta(), però, il browser lo valuta come funzione; nel nostro caso, poiché tale funzione non restituisce alcun valore, al caricamento della pagina non succederà nulla. Per ottenere il risultato desiderato è necessario utilizzare la sintassi:

window.onload = saluta; //ora funziona

In questo modo facciamo riferimento non al risultato della funzione bensì al suo codice vero e proprio.

Un altro metodo consiste nell'assegnare alla funzione un **oggetto function** creato sul momento:

window.onload = function () { alert(“pagina in caricamento”); };

55 Restituire dei valori

Quando si collega del codice ad un evento, è possibile impostare un **valore di ritorno**: se questo è false, verrà annullata l'**azione predefinita**:

cliccami!

Cliccando sul link verrà mostrato un messaggio di conferma; se si clicca il pulsante *annulla* la funzione restituirà false e di conseguenza restituirà false il codice collegato all'evento; in questo modo si elimina l'azione predefinita per l'evento click su un link, cioè non verrà caricata la pagina.

Questa funzionalità serve soprattutto per la **convalida dei form** : si collega all'evento onsubmit del form, una funzione che restituisca false nel caso di errori nella compilazione dei campi.
Ovviamente non si può fare completo affidamento su questa funzionalità, in quanto i JavaScript possono essere facilmente disabilitati dall'utente.

Oltre alle proprietà elencate nel modulo precedente, l'oggetto globale window espone altre importanti proprietà, tra le quali **l'oggetto window.document**.
Questo oggetto permette di ottenere un riferimento al documento HTML e agli elementi in essa contenuti.

56 Metodi

L'oggetto espone solo pochi metodi:

- **open()** e **close()** rispettivamente aprono e chiudono un flusso di informazioni per l'uso dei metodi elencati sotto
- **write()** e **writeln()** scrivono nella pagina la stringa (formattata in HTML) passata come parametro:
  - se la pagina è già stata caricata:
    - se è stato aperto il flusso di informazioni, il testo verrà inserito alla fine della pagina
    - se il flusso non è stato aperto, il testo sostituirà il testo della pagina
  - se la pagina non è ancora stata caricata, se si usano blocchi <script> all'interno del <body>, le informazioni si inseriscono nel punto in cui è presente lo script

Ad esempio:

<body> Testo
 <script type="application/javascript"> document.write('link'); </script> Testo </body>

genererà il codice:

<body> Testo
link Testo </body>

57 Proprietà

Le proprietà dell'oggetto document possono essere relative:

- agli attributi dell'elemento <body> della pagina
- al contenuto vero e proprio del corpo del documento

- altre proprietà del documento HTML

Nel primo gruppo, troviamo proprietà come:

- fgColor e bgColor: indicano rispettivamente il colore di primo piano e di sfondo della pagina. Corrispondono agli attributi text e background.
- linkColor, alinkColor, valinkColor: il colore rispettivamente dei link allo stato normale, dei collegamenti attivi e dei collegamenti visitati. Corrisponde agli attributi HTML link, alink e vlink.

Tra le proprietà del documento HTML la più rilevante è document.title, che permette di accedere al titolo del documento (etichetta <title>.

58 Accedere ai link nella pagina

Il primo attributo che ci permette realmente di accedere agli elementi della pagina è l'array links che restituisce un riferimento agli oggetti <a> presenti nel documento:

var l = document.links[0]; //restituisce un riferimento al primo link nella pagina alert('Nella pagina ci sono ' + document.links.length + ' link'); //ricordiamoci che è un array

Ciascun oggetto link presenta le seguenti proprietà:

- href: restituisce o imposta il valore dell'attributo href del link. A partire da esso derivano alcuni attributi relativi all'URL a cui punta il link:
  - hostname: indica il nome dell'host dell'URL
  - pathname: indica il percorso oggetto indicato dell'URL
  - port: indica la porta dell'URL
  - protocol: indica il protocollo della risorsa a cui punta l'URL (per esempio FTP o HTTP)
  - search: restituisce l'eventuale *querystring* (ad esempio pagina.htm?**name=Mario&cognome=Rossi**)
- target: il valore dell'attributo target del link

Ad esempio possiamo decidere di impostare il target di tutti i link nella pagina a _blank:

for (i in document.links) { document.links[i].target = "_blank"; //imposta l'attributo target }

59 L'array images

Analogamente all'array links, esiste anche un array images che permette di accedere a tutte le immagini presenti nella pagina.
Rispetto a links, questo array presenta una differenza: è possibile accedere agli elementi della pagina anche attraverso il loro attributo name. Ad esempio:

Possiamo accedere a quell'immagine tramite l'indice numerico oppure tramite il nome, in questo modo:

document.images["logo"]

L'oggetto image espone come proprietà gli attributi di qualsiasi immagine HTML, come src, border, ecc... Presenta inoltre un'interessate proprietà complete che restituisce true se l'immagine è già stata caricata dal browser.

60 Accedere ai moduli

Per ultimo trattiamo, in quanto introduce un argomento piuttosto vasto, l'array document.forms[] che, come si potrà facilmente intuire, permette di accedere a tutti gli elementi <form> nella pagina (moduli HTML).

Per accedere a ciascun form nella pagina possiamo utilizzare tre metodi:

document.forms[0] //accede al primo form nella pagina document.forms["dati"] //accede al form con name="dati" document.dati //forma abbreviata

Il metodo presenta due interessanti metodi:

- **reset()** che resetta i valori dei campi nei moduli (come cliccare sul pulsante "reset" del form)
- **submit()** invece invia il form (come cliccare sul pulsante "submit" o "invia" del modulo)

Questo da la possibilità di inviare il modulo, per esempio al semplice click di un collegamento oppure di anticipare il normale invio dei dati:

<form name="opz"> <select name="opzione" onchange="document.opz.submit()"> <option value="modifica">modifica</option> <option value="elimina">elimina</option> <option value="...">...</option> </select> </form>

In questo modo il form si invia automaticamente quando l'utente sceglie un'opzione dal menu a discesa, senza l'utilizzazione del pulsante di invio.

L'oggetto presenta anche alcune proprietà corrispondenti agli attributi dell'oggetto HTML <form>, quali **action**, **encoding**, **method**, **name** e **target**.

60.1 I campi dei moduli

Analizziamo infine, tra le proprietà dell'oggetto form, l'utile array elements[] che permette di accedere a tutti i campi contenuti form:

document.dati.elements[0] //accede al primo elemento del form document.dati.elements["opzione"] //accede al campo con name="opzione" document.dati.opzione //forma abbreviata

L'oggetto restituito da questo array dipende dal tipo di oggetto a cui accede: potrà essere un campo di testo, un pulsante oppure una *textarea*. Tuttavia tutti i campi condividono:

- una proprietà **form** che restituisce un rimando al form genitore;
- una proprietà **value** che permette di ottenere o impostare al valore del campo, in modo diverso a secondo dei singoli tipi di campo;
- la proprietà **type** che ne specifica il tipo;
- la proprietà **name** che ne restituisce il nome (attributo HTML name);
- due metodi **focus()** e **blur()** che rispettivamente attribuiscono e tolgono il *focus* dall'elemento

60.1.1 Caselle di testo

Questi elementi sono creati utilizzando il tag HTML <input> impostando l'attributo type ai valori *text*, *password* o *hidden*: la differenza è che il secondo mostra degli asterischi al posto dei caratteri reali e il terzo è invisibile, ma noi li tratteremo insieme in quanto condividono proprietà e metodi.

Questi tre tipi di moduli restituiscono come attributo **type** il valore rispettivamente *text*, *password* e *hidden*. Hanno poi due proprietà, **size** e **maxlength**, che corrispondono ai rispettivi attributi HTML.

Per quanto riguarda i metodi, invece, presentano **select()**, che seleziona l'intero testo contenuto nel campo (ovviamente non serve per i campi di testo nascosti).

L'evento associato normalmente a questi campi è **onchange**, che viene chiamato dopo la modifica del valore nel campo.

60.1.2 Aree di testo

Le aree di testo sono create tramite l'utilizzo del tag HTML <textarea> e sono rappresentante dall'omonimo oggetto *textarea* in JavaScript.

Questi oggetti presentano proprietà analoghe ai campi di testo: **value** imposta o restituisce il valore della casella di testo, **rows** e **cols** corrispondono ai rispettivi attributi HTML.
L'oggetto *textarea* permette di inserire anche degli "a capo" nel testo, ma in questo caso bisogna stare attenti, in quanto il carattere di accapo è espresso diversamente a seconda del sistema operativo in uso: su Windows è \n\r, su Linux è \n mentre su Mac è \r (\ è il carattere di commutazione visto in precedenza per gli apici singoli e doppi).

60.1.3 Pulsanti

Per quanto riguarda i pulsanti dei moduli, questi possono essere di tre tipi: *submit*, *reset* e *button*; tuttavia espongono lo stesso modello oggetto, che in realtà è molto semplice, in quanto non presenta proprietà o metodi particolari oltre a quelli condivisi da tutti i campi di modulo.

L'utilizzo di JavaScript con questi pulsanti è più che altro legato agli eventi del mouse (analizzati nel modulo precedente).

60.1.4 Pulsanti di opzione

Iniziamo ad analizzare ora alcuni elementi più complicati: i *checkbox* e i pulsanti *radio* (corrispondono rispettivamente ad <input type="checkbox" /> e <input type="radio" />).

Nel primo la proprietà più utile non è *value* come negli altri casi, bensì **checked** che indica se la casella è stata marcata o meno.

Per quanto riguarda i *radio*, il discorso è simile, ma bisogna anche considerare che diversi elementi di opzione normalmente condividono lo stesso attributo name; per accedere quindi ai singoli elementi è necessario quindi appoggiarsi ad un array:

// supponiamo di avere 5 opzioni con name="età" document.dati.età[1].checked=true; //seleziona la seconda opzione alert(document.dati.età.value); //restituirà il valore del campo // cioè il valore della casella selezionata alert(document.dati.età[3].value); //restituisce il valore del quarto elemento

60.1.5 Liste

Analizziamo per ultimo gli elementi <select>, che servono per creare liste e menu a tendina. Prendiamo il seguente spezzone di codice HTML in un ipotetico form chiamato "dati":

<select name=città size=5> <option value=mi>Milano</option> <option value=to>Torino</option> <option value=pe>Pescara</option> <option value=pa>Palermo</option> </select>

Questo codice crea un oggetto *select* accessibile come sempra con la sintassi document.dati.città. Questo espone alcune proprietà interessanti:

- **value** è il valore del campo del modulo (nel nostro caso potrà essere *mi*, *to*, ecc...)
- **options** è un array contenente un riferimento a ciascun oggetto *option* appartenente all'elenco
- **selectedIndex** restituisce l'indice dell'elemento selezionato dall'utente (Milano sarà 0, Torino 1, ecc.)

Ogni oggetto *option* espone le seguenti proprietà:

- **value** indica il suo valore
- **text** indica il testo che mostrerà all'utente
- **index** restituisce la sua posizione nell'array.

Ad esempio, associamo all'evento *onchange* del campo “città" alla seguente funzione:

function mostraCittà() { var testo = document.dati.città.options[document.dati.città.selectedIndex].text; }

Cliccando su ciascuna opzione otterremo così un'alertbox contenente la città selezionata.

Inserire e aggiungere elementi Per inserire o aggiungere nuovi elementi è sufficiente creare un nuovo oggetto *option* e assegnarlo ad una posizione dell'array:

var nuovaOpzione = new Option(“Bologna”, “bo”); document.dati.città.options[0] = nuovaOpzione;

L'esempio appena visto aggiunge la città “Bologna” come primo elemento della lista. Il browser si occuperà poi di far scalare le posizioni degli altri elementi dell'array. Per eliminare un'opzione è sufficiente assegnare all'elemento dell'array il valore *null*:

document.dati.città.options[2] = null;

La seguente funzione aggiunge una città come ultimo elemento nella lista:

function aggiungiCittà () { var testo = prompt(“Che città vuoi aggiungere?"); var sigla = prompt(“Inserisci la sigla della città"); var nuovaCittà = new Option(testo, sigla); document.dati.città.options[document.dati.città.options.length] = nuovaCittà }

Oltre all'oggetto document, l'oggetto window dispone di altre proprietà che restituiscono oggetti particolarmente utili.

61 location

Questo oggetto restituisce informazioni relative all'indirizzo della pagina caricata.

Le sue proprietà sono le stesse che per l'oggetto link trattato nel modulo precedente
Dispone inoltre di due interessanti metodi:

- **reload**(): ricarica la pagina
- **replace**(): carica l'URL indicata come parametro sostituendola in cronologia alla voce corrente. Usando invece location.href = url vengono salvate entrambe le pagine in cronologia.

62 history

permette di accedere alla cronologia delle pagine visitate dall'utente. Dispone di una sola proprietà **length** che restituisce il numero di pagine nella cronologia. Dispone poi dei metodi:

- **go**(): carica la pagina nella posizione indicata rispetto alla pagina corrente; ad esempio:

history.go(−2); //carica la seconda pagina caricata prima di quella corrente history.go(3); //come premere 3 volte il pulsante “avanti” del browser

- **back**() e **forward**(): caricano la pagina precedente e successiva (corrispondono a history.go(−1) e history.go(1))

Spesso l'oggetto history è usato in associazione agli eventi della pagina per creare bottoni “indietro":

<< Indietro

63 navigator

Questo oggetto consente di ottenere informazioni riguardo al browser in uso dall'utente e al suo sistema operativo. Possiede interessanti proprietà:

- **appName** è una stringa che restituisce il nome del browser (Opera, Netscape - valore restituito anche da Firefox -, Microsoft Internet Explorer...).
- **appVersion** contiene dati sul browser, quali la versione della *release* e il sistema operativo sul quale sta girando.
- **plugins** restituisce un array dei *plugin* installati.

Tramite la prima proprietà, possiamo verificare il browser in uso per agire in modo diverso ed eliminare così alcuni problemi di compatibilità:

var ns = (navigator.appName.indexOf(“Netscape”) > −1) //se l'utente naviga con Firefox/Netscape, contiene true var mie = (navigator.appName.indexOf(“Microsoft”) > −1) //microsoft internet explorer var wie = (navigator.appName.indexOf(“Windows”) > −1) //windows internet explorer var opera = (navigator.appName.indexOf(“Opera”) > −1) //etc... if (ns) { //codice per netscape/firefox } else if (mie || wie) { //codice per ie } else if (opera) { //codice per opera } else { //codice per altri browser }

Normalmente è possibile omettere i controlli per browser quali Opera, che normalmente mantengono una grande compatibilità con Firefox, in quanto entrambi seguono le stesse convenzioni del W3C; spesso il problema è trovare una soluzione compatibile anche con IE. Attenzione: navigator.appName e navigator.appVersion sono deprecati. Non fare affidamento su appName perchè in molti browser ritorna come valore 'Netscape'.

64 screen

Questo oggetto permette di ottenere informazioni sullo schermo dell'utente e sulla sua configurazione. Tra le sue proprietà ricordiamo:

- **height** e **width** che indicano la risoluzione verticale e orizzontale dello schermo in pixel
- **colorDepth** (profondità del colore) indica il numero di bit utilizzati per i colori (ad esempio, 2 indica che l'utente sta navigando con uno schermo in bianco e nero - fatto altamente improbabile, perché ormai la maggior parte dei monitor supportano la configurazione 16 o 32 bit).

Il **DOM** è una tecnologia standard del W3C per muoversi nella struttura di documenti XML.
In relazione a JavaScript questo è molto interessante, in quanto un documento HTML ben formato è in realtà un documento XML, e tramite il DOM di JavaScript è possibile accedere a *qualsiasi* elemento nella pagina per modificarne le proprietà o per attivarne i metodi. Il DOM definisce anche un modello oggetto più evoluto del BOM che tratteremo quindi in modo più approfondito.

65 BOM vs DOM

Ci si potrebbe a questo punto chiedere dove si sovrappongano il DOM che andremo ad analizzare e BOM che abbiamo appena visto e, soprattutto, perché abbiamo esaminato anche il BOM, nonostante sia meno evoluto.

Il problema sorge in realtà dall'evoluzione dei browser, che hanno creato dei propri standard prima della definizione del W3C. In realtà il DOM non è altro che un'evoluzione del BOM sviluppato inizialmente da browser come Netscape o Internet Explorer con l'avvento di JavaScript.

Solo successivamente il DOM è stato standardizzato, ma si continua ancora oggi ad usare il BOM, per problemi di compatibilità (anche se ormai questi sono praticamente minimi) o anche perché quest'ultimo è più “intuitivo” e non richiede di capire la gerarchia XML della pagina.

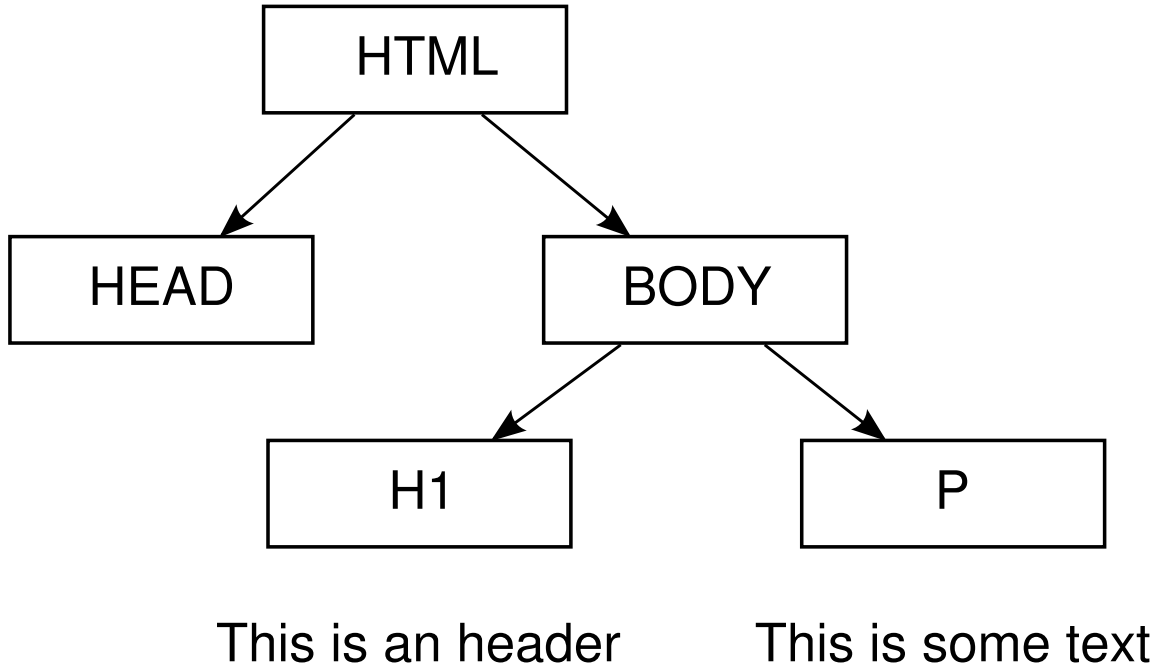
66 La struttura gerarchica di un documento

Per poter lavorare con il DOM è però necessario capire la struttura ad albero di un documento, perché è su questa che si muove. Questa struttura è astratta e parte dalla gerarchia degli elementi in una pagina.

Prendiamo ad esempio questa pagina HTML:

<html> <head></head> <body> <h1>This is an header</h1> This is some text </body> </html>

È un semplice documento HTML contenente un'intestazione e un paragrafo. La sua struttura ad albero sarà la seguente:



Ogni elemento, di qualsiasi tipo esso sia, occupa una posizione ben definita. Ogni elemento nella struttura ad albero viene chiamato "*nodo*", e rappresenta nel modo più astratto ciascun elemento della pagina.
Ad ogni nodo si può associare un nodo *genitore*, tranne che per il nodo radice del documento, nel nostro caso <html>. Ogni nodo può avere dei nodi *figli* (ad esempio l'intestazione e il paragrafo sono figli dell'elemento <body> e dei nodi fratelli (ad esempio è fratello di <h1>).

Questa è un'altra differenza dal BOM: non esiste più un oggetto per ogni elemento nella pagina, ma esiste un oggetto astratto che li rappresenta tutti. Non sarà più quindi necessario studiare *singolarmente* ogni singolo elemento della pagina, ma solo i tipi di nodi più importanti.

66.1 Vari tipi di nodo

L'oggetto *Node* è quindi, come abbiamo detto, il modo più astratto per indicare un elemento della pagina; per i diversi tipi di elemento esistono poi oggetti più specifici, che sono delle *estensioni* dell'oggetto *Node*; tra i più importanti troviamo:

- *Document* è un oggetto radice del documento. L'oggetto *window.document* è un'istanza di questo oggetto.
- *Element* è un nodo che rappresenta un elemento HTML (come)

- *Text* è un nodo di testo, che costituisce l'ultimo nodo figlio di un elemento (ad esempio *This is some text* è un nodo di testo)

Dopo l'introduzione generale al DOM e alla sua struttura, passiamo ora ad analizzare i suoi oggetti per applicare quanto visto.

67 Ottenere un elemento

Vediamo ora il primo passo da compiere quando si lavora con il DOM: ottenere il riferimento ad un elemento HTML e memorizzarlo in una variabile.

Per fare questo, nella maniera più semplice, il DOM utilizza l'attributo id che è possibile assegnare ad ogni elemento:

var par = document.getElementById(“p1”); //ottiene il paragrafo con id="p1”

Per ottenere invece l'elemento root del file HTML, si usa la proprietà document.documentElement.

68 Lavorare su un elemento

Una volta ottenuto l'elemento (questo è il modo più semplice, ma ne esistono anche altri), è possibile modificarne gli attributi o modificare gli stili:

- la proprietà **tagName** restituisce il nome del tag (per esempio *a* o *img*)
- la proprietà **style** permette di impostarne lo stile in modo veloce (questa caratteristica non è in realtà uno standard DOM, ma funziona bene su tutti i browser) tramite il nome della proprietà CSS, ad esempio p1.style.color = "#f00"; imposta a rosso il colore del paragrafo ottenuto prima
- esistono poi tre metodi per impostare o eliminare gli attributi:
  - **setAttribute** imposta un attributo tramite il nome e il valore (p1.setAttribute(“align”, “center”););
  - **getAttribute** restituisce il valore di un attributo tramite il suo nome (p1.getAttribute(“align”););
  - **removeAttribute** rimuove un attributo (tornando così al valore di default) tramite il suo nome (p1.removeAttribute(“align”);)

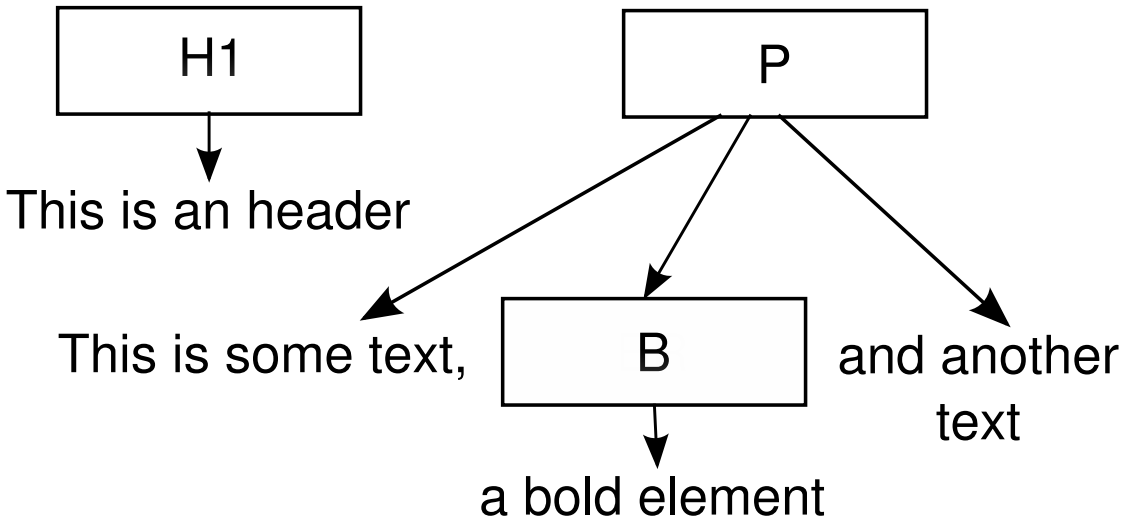
69 Spostarsi nella struttura ad albero

Arriva ora il momento di sfruttare la struttura ad albero che abbiamo visto ora. L'*node* espone infatti delle proprietà e dei metodi che permettono di spostarsi nella struttura ad albero, per ottenere nodi fratelli, genitori o figli:

- **firstChild** e **lastChild** restituiscono il primo e l'ultimo nodo figlio di un elemento (se è uno solo, restituiscono lo stesso elemento).
- **previousSibling** e **nextSibling** restituiscono il precedente e il successivo nodo “fratello” del nodo
- **parentNode** restituisce il nodo genitore
- **ownerDocument** restituisce l'elemento del documento che contiene il nodo (ad esempio <body>)
- **getElementsByTagName()** è un metodo che restituisce un elenco di tutti i nodi figli con il nome specificato (per esempio "*img*"); per accedere ad un elemento si usa la normale notazione degli array[1].

Una volta ottenuto un nodo, è possibile usare la proprietà **nodeValue** che ne restituisce il valore e la proprietà **nodeType** che ne restituisce il tipo come numero.

Analizziamo ad esempio questo spezzone di HTML:



<h1>This is an header</h1> This is some text, a bold element and another text

Questa è la sua rappresentazione del DOM:

Vediamo ora come agire. La cosa più semplice è partire dall'elemento con un ID univoco:

var p1 = document.getElementById(“p1”); var h1 = document.previousSibling; //ottiene l'elemento H1 alert(p1.nodeValue); //restituisce “null” (un elemento HTML non ha un valore) var testo1 = p1.firstChild; alert(testo1.nodeValue); //restituisce il valore del nodo, ovvero “This is some text” alert(testo1.nextSibling); //restituisce “Object BElement” //in quanto il fratello del primo nodo di testo è l'elemento alert(testo1.nextSibling.firstChild); //restituisce il riferimento al nodo di testo contenuto nel tag

Vediamo un altro esempio, con questo spezzone di codice

 Paragraph 1 Paragraph 2 <h1>An HTML header</h1> Paragraph 3

Usando il metodo getElementsByTagName possiamo ottenere un array di tutti i nodi degli elementi all'interno del div:

var myDiv = document.getElementById(“myDiv”); // ottiene il nodo div var myParagraphs = myDiv.getElementsByTagName('P'); //restituisce un array di tutti i paragrafi // ad esempio possiamo ottenere l'ultimo paragrafo del div semplicemente scrivendo: var mySecondPar = myParagraphs[myParagraphs.length - 1]

Infatti, essendo myParagraphs un array, il primo elemento avrà indice 0 (myParagraphs[0]). Se l'array ha 3 elementi, l'ultimo elemento sarà quindi myParagraphs[2], ovvero la lunghezza dell'array (myParagraphs.length) diminuita di 1.

70 Creare nuovi elementi

La possibilità di gestire qualsiasi elemento della pagina come oggetto dà anche la possibilità di creare nuovi elementi, semplicemente creando nuovi oggetti di nodo e poi inserendoli nella struttura ad albero. Per fare ciò esistono alcuni particolari metodi, prima di tutto dell'oggetto document:

- **createElement**: crea un nodo di elemento con il nome specificato come parametro
- **createTextNode**: crea un nodo di testo con il testo specificato

Una volta creato un nodo, sia esso di elemento o di testo, va aggiunto ai nodi già esistenti:

- **appendChild** aggiunge al nodo il nodo figlio passato come parametro, in ultima posizione
- **insertBefore** inserire il nodo specificato prima del nodo specificato come secondo argomento

La sintassi è: nodo_genitore.insertBefore(nuovo_elemento,vecchio_elemento);

- **removeChild** elimina il nodo figlio indicato come parametro
- **replaceChild** sostituisce il nodo figli indicato nel primo parametro con il secondo
- **hasChildNodes** restituisce *true* se il nodo ha nodi figli.

Vediamo un esempio:

var body = document.body; var link = document.createElement(“a”); var testo = document.createTextNode(“Segui questo link”); link.setAttribute(“href”, “pagina.html”); link.appendChild(testo); body.appendChild(link);

71 Note

[1] http://www.w3schools.com/htmldom/met_doc_getelementsbytagname.asp

Il **modello evento del DOM** in realtà non offre grandi funzionalità in più rispetto a quello del BOM di cui abbiamo visto una parte in precedenza; tuttavia il modello evento del DOM ha un migliore supporto e compatibilità tra i browser e, essendo il DOM l'ultima tecnologia sviluppata in questo campo, l'evoluzione del JavaScript non si baserà sul BOM bensì sul DOM.

72 Analogie con quanto già visto

In realtà molto di ciò che è stato visto per il BOM è riutilizzabile anche con il DOM: sono uguali i nomi degli eventi, il funzionamento del collegamento tra oggetti ed eventi, l'uso dell'oggetto *this*

73 L'oggetto event

La funzionalità interessante che tratteremo ora è l'**oggetto *event***, che permette di ottenere informazioni sull'evento appena scaturito, come l'elemento che lo ha generato o la posizione del mouse, e di eseguire codice JavaScript all'accadere di un evento.

Un evento è “qualcosa” che accade a un elemento HTML. Può essere rappresentato da un'interazione dell'utente, o da un evento che accade nel rendering della pagina (ad esempio la pagina ha finito di caricarsi), o altro. JavaScript può reagire a questi eventi ed eseguire del codice.

Possiamo utilizzarlo all'interno di un elemento HTML, ad esempio:

 link

In questo esempio utilizziamo le due proprietà *screenX* e *screenY* dell'oggetto event, che restituiscono la posizione del cursore del mouse rispetto allo schermo.

Rispetto all'utilizzo di questo oggetto, bisogna fare attenzione quando si richiama da delle funzioni *handler*: l'oggetto event infatti ha una visibilità privata, non può quindi essere richiamato esternamente alla dichiarazione dell'evento (nell'attributo “onclick” per esempio). Per ovviare a questo problema è sufficiente passare l'oggetto event come argomento. Ad esempio, si crea la funzione:

function posizione (e) { alert('Il mouse è alla posizione ' + e.screenX + ', ' + e.screenY + 'dello schermo'); }

Nell'HTML si inserirà:

link

73.1 Proprietà

- **timestamp** (funziona solo su FF) restituisce la data e l'ora in cui si è verificato l'evento;
- **target** e **relatedTarget** (**srcElement** e **toElement** su Internet Explorer) restituiscono rispettivamente il nodo che ha generato l'evento e il nodo su cui probabilmente sarà il mouse dopo che l'evento si sarà concluso (ad esempio è utile per gli eventi *mouseOut*). Quando si usano queste proprietà bisogna prima controllare il browser in uso dall'utente (vedi questa pagina);
- **altKey**, **ctrlKey** e **shiftKey** indicano rispettivamente se è premuto il tasto *alt*, *ctrl* o *shift* mentre avviene evento;
- **button** indica quale pulsante del mouse è stato premuto (0 il tasto sinistro, 1 quello destro);
- **clientX** e **clientY** indicano la posizione del cursore rispetto alla finestra del browser (a partire dall'angolo in alto a sinistra);
- **screenX** e **screenY** indicano la posizione del cursore rispetto allo schermo dell'utente (a partire dall'angolo in alto a sinistra).

74 addEventListner()

addEventListner() permette di impostare una funzione che verrà richiamata al verificarsi dell'evento specificato, sull'elemento specificato:

target.addEventListener(tipo, funzione);

tipo è una stringa che rappresenta il tipo di evento catturato. Può essere applicata ad ogni elemento del DOM, non solo agli elementi HTML.

JavaScript, oltre ai metodi già spiegati relativi all'oggetto String, offre potenti funzioni di manipolazione stringa con le **espressioni regolari**.

Le espressioni regolari sono un particolare linguaggio con il quale è possibile abbinare le stringhe per effettuare ricerche, sostituzioni o semplici confronti.
Ad esempio, se nei precedenti capitoli abbiamo visto come sostituire tutte le “a” in una stringa (con il metodo replace) dopo aver letto questo capitolo potremo sostituire tutte le parole con dei numeri, o anche semplicemente tutte le cifre con un carattere nullo.

Le espressioni regolari di JavaScript sono molto simili a quelle del Perl. In questo capitolo tuttavia ci soffermeremo a spiegarne almeno i fondamenti.

75 Espressioni regolari in JavaScript

L'uso delle espressioni regolari in JavaScript si basa sul'oggetto RegExp:

var myRegex = new RegExp (“testo della regex”, “parametri”); var r = new RegExp (“Ciao *", “g”);

Il primo parametro è l'espressione regolare vera e propria, mentre il secondo (opzionale) è una stringa combinando uno o più parametri espressi come lettere:

- **g** indica un abbinamento globale (ovvero verranno abbinate tutte le istanze che corrispondono all'espressione regolare e non solo la prima)
- **i** indica di non distinguere tra maiuscole e minuscole

Esiste anche una forma abbreviata per la creazione, che useremo più spesso:

var myRegex = /testo della regex/parametri; var r = /[Ee]spressione regolare/ig;

76 Sintassi delle espressioni regolari

L'espressione regolare più semplice che possiamo dichiarare è quella che non contiene caratteri speciali, ma il suo uso sarà abbastanza limitato. Ad esempio l'espressione regolare /JavaScript/ abbinerà **solo** le stringhe “JavaScript”. Per rendere le espressioni regolari invece veramente potenti, usiamo i cosiddetti caratteri speciali, che non identificano il carattere stesso, ma altri caratteri particolari o gruppi di caratteri.

76.1 Testo e punteggiatura

76.2 Un primo esempio

Vediamo un primo esempio per iniziare a capire come utilizzare in concreto le nostre espressioni regolari.
Supponiamo di voler creare un modulo per la registrazione ad un sito e chiediamo all'utente di inserire come *username* una parola che non contenga caratteri speciali, ma solo lettere, numeri e il simbolo di underscore.
Potrebbe venire in mente che per fare questo ci possa tornare utile il metacarattere visto in precedenza \w. Tuttavia questo non è possibile, in quanto \w abbina un *singolo carattere non di parola*: potremmo ad esempio controllare che il nostro nome utente si abbini con \w\w\w\w, ma questo funzionerebbe solo con un *username* di quattro lettere. La cosa più comoda è quindi verificare che nella nostra stringa ci sia un carattere *non* di parola e, in caso ciò sia vero, chiedere all'utente di scegliere un diverso *username*.

Per verificare che una regex abbini una stringa, usiamo il metodo test dell'oggetto RegExp che restituisce true se la regex si abbina, false in caso contrario:

var username = document.form.username.value; //prende il nome dal form var myRegex = /\W/i; //non serve effettuare una ricerca globale //in quanto se c'è anche un solo carattere non di parola l'username non è valido var non_valido = myRegex.test(username); if (non_valido == true) { alert('Nome utente non valido. Riprova'); } else { //prosegui... }

76.3 Caratteri di ripetizione

I seguenti caratteri servono per abbinare gli elementi definiti precedentemente più volte in base a quanto indicato.

Supponiamo ad esempio di aver chiesto all'utente di inserire un CAP e di voler verificare che sia valido:

var cap = document.form.username.value; //prende il nome dal form var myRegex = /\d{5}/; //5 cifre if (myRegex.test(cap)) { //il cap è valido... } else { alert('Inserire un CAP valido'); }

76.4 Posizione

Analizziamo ora alcuni caratteri di posizione, che non specificano un particolare tipo di caratteri, bensì una particolare posizione nella stringa.

[1] Per lavorare con stringhe su più righe si deve impostare la proprietà multiline dell'oggetto RegExp su true (myRegex.multiline = true;).

76.5 Raggruppare i caratteri

Per raggruppare i caratteri nelle espressioni regolari si usano le parentesi tonde: (). Ad esempio, (\d\w)+ abbina una o più sequenze di una cifra e un carattere di parola (ad esempio 5a7v4g ma non 3f5r8. All'interno delle parentesi è anche possibile usare il carattere | per indicare una scelta tra due alternative: (Java|VB)Script abbina sia JavaScript che VBScript.

76.6 Backreference

I *backreference* sono una interessante funzionalità delle espressioni regolari che permette di riferirsi ai caratteri abbinati da un gruppo indicato in precedenza con le parentesi. Per indicare una backreference si usa la sintassi *n*, dove *n* è il numero del gruppo a cui si vuole riferire: il primo gruppo sarà indicato con \1, il secondo con \2, ecc...
Supponiamo ad esempio di volere una stringa formata da un solo carattere di parola ripetuto più volte, qualunque esso sia (ad esempio aaaaaaaa ma non aaabaaa: il primo carattere sarà quindi \w. Per poter fare riferimento al carattere abbinato da \w lo mettiamo tra parentesi: (\w); ora \1 abbinerà il carattere abbinato all'inizio. La nostra regex sarà quindi (\w)\1+.

77 Metodi

Le principale funzioni utilizzate insieme alle espressioni regolari sono le funzioni split(), replace(), search() e match() dell'oggetto String. La maggior parte di esse sono state spiegate nel modulo relativo, ma l'uso delle regexp le arricchisce di molto.

77.1 split()

Il metodo split() divide la stringa in base a quanto indicato nel parametro e restituisce un array che contiene tutte le sottostringhe in cui è stato diviso. Supponiamo ad esempio di voler ottenere in un array le parole contenute in un testo, ad esempio per contarne il numero: dovremo quindi trovare un'espressione regolare che abbini ciascun i caratteri tra una parola e l'altra.

Per fare questo possiamo avvalerci del metacarattere \W che abbina ogni carattere non di parola. Dovremo tuttavia prevedere che ci siano più caratteri non stringa tra le parole (ad esempio ,). Usiamo quindi \W+: var parole = testo.split(/\W+/g);.

77.2 replace()

Il metodo replace () sostituisce il testo abbinato dalla regex indicata come primo parametro con il testo indicato nella seconda.
Si ricordi sempre che per sostituire *tutte* le occorrenze abbinate dalla regex è necessario impostare il parametro globale della regex; supponiamo di voler eliminare tutti i caratteri da una stringa in modo da ottenere un numero intero:

var str = new String('245dfh57w'); var num = str.replace(/\D/, ''); //restituisce “245fh57w” var num2 = str.replace(/\D/g, ''); //restituisce “24557”

Anche nei replace è possibile utilizzare dei *backreference*, usando la sintassi $*n*:

//corregge le diciture “cm n” con “n cm” var str = “La base misura cm 30 e l'altezza cm 50"; var str2 = str.replace(/([mdc]m) (\d+([\.,]\d+)?)/gi, "$2 $1”);

Esaminiamo la nostra regex /([mdc]m) ((\d+(\.\d+)?)/gi:

- [mdc]m serve per abbinare uno tra i tre caratteri *m*, *d* o *c* seguiti da m: in questo modo verranno abbinati *cm*, *dm* e *mm*;
- le parentesi attorno (([mdc]m)) servono per poter usare il backreference;
- \d+ abbina una o più cifre disposte di seguito, che costituiscono la parte intera della nostra misura
- il carattere di punto con davanti la barra non sta a significare il metacarattere punto bensì il punto vero e proprio (.), che delimita i decimali. [\.,] serve per abbinare o un punto o una virgola;
- [\.,]\d+ abbina così la parte decimale della misura (es. .474); questa tuttavia, essendo opzionale (nel caso di numeri interi) è posta tra parentesi e seguita da ? che indica che la parte decimale può essere presente una volta o non esistere affatto;

- (\d+([\.,]\d+)?) è infine tutto tra parentesi per permettere l'utilizzo del backreference;
- gi indicano di effettuare un abbinamento globale non sensibile alle maiuscole.

È possibile abbinare anche delle posizioni:

var str = “Lorem ipsum dolor sit amet"; var str2 = str.replace(/\B/g, '|'); //sostituisce i caratteri non di fine stringa con "|" //ora str2 contiene L|o|r|e|m i|p|s|u|m d|o|l|o|r s|i|t a|m|e|t

77.3 search()

Il metodo search() è una versione avanzata del già visto metodo indexOf(), in quanto supporta le espressioni regolari, restituendo la posizione nella quale viene abbinata la regex indicata come parametro (-1 indica che non è stato trovato alcun abbinamento).

77.4 match()

Il metodo match() abbina la regex nel testo cercato e restituisce un array delle stringhe abbinate. Questo è molto utile in quanto permette di sapere realmente i caratteri abbinati dalla regex.

Ad esempio supponiamo di voler cercare tutti gli indirizzi e-mail presenti in un testo:

var myRegex = /\b\w+@\w+\.\w{2,3}\b/gi; var indirizzi = testo.match(myRegex);

La nostra regex abbina una parola intera (\b...\b) composta da uno o più caratteri parola (\w+) seguiti da una chiocciola, altri caratteri di parola, seguiti da un punto (\.: il backslash serve per evitare di usare il metacarattere punto) e da due o tre caratteri di parola (\w**{2,3}**).

Con l'acronimo **AJAX** (***Asynchronous JavaScript And XML***) si identifica l'uso di un particolare oggetto JavaScript (l'oggetto XMLHttpRequest) che permette di effettuare le cosiddette *richieste asincrone*.

78 Un altro tipo di dinamicità

Finora abbiamo visto come rendere dinamica una pagina nell'ambito dei file in cui viene inserito il file JavaScript. Usando l'oggetto XMLHttpRequest, invece, possiamo rendere dinamica la pagina nell'ambito delle richieste file-server tramite il protocollo HTTP.

In pratica, possiamo caricare, tramite JavaScript, pagine presenti sul nostro web server; questo permette quindi di poter eseguire una grande quantità di operazioni senza dover far caricare all'utente altre pagine.

79 Un esempio

Supponiamo ad esempio di voler creare una tipica pagina “mostra suggerimenti": vogliamo che, ogni volta che l'utente clicca su un pulsante, venga caricato un nuovo suggerimento.

Per fare questo abbiamo implementato sul server una pagina scritta in un linguaggio lato server (come Java, PHP o ASP) che restituisca un suggerimento casuale tra un elenco di suggerimenti contenuti in un database. La pagina restituita dal nostro script lato server non deve contenere la struttura del documento HTML, ma solo il testo del suggerimento.

Con AJAX in questo modo potremmo fare sì che quando l'utente clicca sul pulsante “Mostra suggerimento” tramite l'oggetto XMLHttpRequest richiamiamo il testo restituito dal nostro script lato server (attenzione, non carichiamo il sorgente del PHP o ASP o quello che sia, ma il testo che restituiscono) e lo mostriamo all'interno di un apposito div.

80 Creare l'oggetto XMLHttpRequest

Per istanziare un oggetto XMLHttpRequest usare la seguente sintassi:

xmlhttp=new XMLHttpRequest();

Tutti i browser moderni supportano nativamente l'oggetto XMLHttpRequest. Tuttavia Internet Explorer lo supporta solamente dalla versione 7. Se si vuole garantire compatibilità anche con le versioni precedenti di IE, si può usare la funzione fornita di seguito (che restituisce un oggetto AJAX oppure avvisa l'utente che la funzionalità non è presente sul browser in uso):

function newAjax() { var xmlHttp; try { // Firefox, Opera e Safari xmlHttp=new XMLHttpRequest(); } catch (exc) { // Internet Explorer try { xmlHttp=new ActiveXObject(“Microsoft.XMLHTTP”); } catch (exc) { try { xmlHttp=new ActiveXObject(“Msxml2.XMLHTTP”); } catch (exc) { alert(“Il tuo browser non supporta AJAX!"); return false; } } } return xmlHttp; }

Questa funzione che restituisce un nuovo oggetto XMLHttpRequest utilizza il costrutto try... catch; senza analizzarlo nei dettagli, questo costrutto permette di eseguire una porzione di codice (indicata nel blocco try) e, nel caso questa porzione generi un errore, eseguirne un altro (blocco catch).
In questo modo se la prima riga xmlHttp=new XMLHttpRequest(); non funziona viene richiamato il blocco successivo, che istanzia l'oggetto in modo corretto per le versioni di Internet Explorer precedenti alla 7.

Per creare un nuovo oggetto AJAX, quindi, basterà inserire nel nostro script la funzione sopra indicata e poi la riga:

var ajax = newAjax();

81 Altri progetti

-  **Wikipedia** contiene una voce su **AJAX**

82 Collegamenti esterni

- (IT) Iniziare AJAX | MDN
- (EN) Tutorial ad AJAX su w3schools

In questo capitolo vedremo una semplice applicazione pratica della tecnologia AJAX per creare il programma presentato nel capitolo precedente.

83 Chiedere dati al server

Per effettuare la nostra *richiesta asincrona* al server usiamo il metodo open() dell'oggetto XMLHttpRequest e inviamo successivamente la richiesta con il metodo send():

var ajax = newAjax(); var url = “script.php"; ajax.open(“GET”, url, true); ajax.send(null);

La prima riga crea un nuovo oggetto AJAX (richiede la funzione già creata); viene poi dichiarata una variabile url che contiene il nome dello script sul server che vogliamo richiamare.

Il metodo open() prende come primo parametro il metodo di invio dei dati al server: GET permette di inviare i dati nella *querystring* mentre POST richiede un metodo più sofisticato che non sarà trattato in questa semplice introduzione ad AJAX; il secondo parametro è l'url della pagina che vogliamo richiamare e il terzo indica se vogliamo effettuare una richiesta asincrona (normalmente è impostato su true).

Il metodo send() invia la richiesta creata; il parametro serve per l'invio dei dati con POST, quindi non lo trattiamo.

Nell'url è possibile anche impostare una querystring nel caso si usi il metodo GET, ad esempio script.php?action=mostra&id=5.

84 Leggere i dati dal server

84.1 La proprietà readyState

La proprietà readyState indica lo stato della richiesta che abbiamo effettuato. Può assumere un valore da 0 a 4:

84.2 La proprietà onreadystatechange

La proprietà onreadystatechange dell'oggetto XMLHttpRequest serve per impostare una funzione da richiamare quando viene alterato il valore della proprietà readyState. Ad esempio si può fare in questo modo:

function gestisciAJAX () { //... } ajax.onreadystatechange = gestisciAJAX; //attenzione, non vanno le parentesi! //altro metodo ajax.onreadystatechange = function () { //... }

In questo modo basterà effettuare un controllo per verificare se il readyState è uguale a 4 (la richiesta è stata cioè completata) e quindi leggere i dati del server tramite la proprietà responseText/:

var ajax = newAjax(); ajax.onreadystatechange = function () { if (ajax.readyState == 4) { //fai qualcosa... } }

84.3 La proprietà status

Al termine della richiesta AJAX viene restituito il codice di stato HTTP che dice se la richiesta è terminata in modo corretto o no (per una lista completa dei codici andate qui).

Ecco un esempio:

//la nostra solita funzione... ajax.onreadystatechange = function () { if (ajax.readyState == 4) { // verifico che la richiesta è finita if(ajax.status == 200){ // controllo che la richiesta è terminata in modo corretto // codice da eseguire se la richiesta è andata a buon fine } else { // codice da eseguire se la richiesta ha restituito errore } } }

85 Il nostro esempio

Vediamo quindi il nostro esempio.
Di seguito è mostrato il codice PHP della pagina tips.php che restituisce ogni volta un suggerimento diverso:

$tips = array(//inizializza i suggerimenti “Per aprire il menu, premi ALT+F!", “Ricordati di fare sempre un backup dei tuoi file importanti!" //altri suggerimenti...); $i = array_rand($tips); //un indice casuale echo $tips[$i]; //restituisce un tip a caso

Creiamo quindi un pulsante e un div:

 <input type="button” onclick="mostraTip();" value="Mostra suggerimento” />

Infine nel nostro JavaScript (ricordatevi prima di inizializzare l'oggetto XMLHttpRequest come abbiamo fatto nel capitolo precedente):

function mostraTip() { //la nostra solita funzione... ajax.onreadystatechange = function () { if (ajax.readyState == 4) { if(ajax.status == 200){ // controllo che la richiesta sia avvenuta con successo var resp = ajax.responseText; //prende i dati var tipDiv = document.getElementById('tip'); //ottiene il div tipDiv.innerHTML = resp; //imposta il contenuto } else { var tipDiv = document.getElementById('tip'); //ottiene il div var error = 'Errore la richiesta ajax non è avvenuta con successo' ; tipDiv.innerHTML = error ; //imposta il contenuto di errore } } } ajax.open(“GET”,

“tips.php”, true); ajax.send(null); }

La proprietà innerHTML è molto comoda (anche se non è prevista dalle specifiche del W3C) in quanto permette di impostare il codice HTML contenuto in un elemento del DOM (nel nostro caso il div).

In questo capitolo vedremo come creare nuovi **oggetti e prototipi** e come modificare gli oggetti esistenti in JavaScript. Questa è una delle varie possibilità dei linguaggi che implementano i tipi di dato direttamente come oggetti, consentendo di poter agire sul “cuore” del linguaggio stesso.

86 Creare oggetti personalizzati

Come abbiamo spiegato nel capitolo sugli oggetti di JavaScript, per definire un nuovo prototipo di un oggetto è sufficiente definirne un costruttore, che non è altro che una funzione.

Ad esempio, volendo creare un oggetto “Macchina”, sarà sufficiente creare una funzione Macchina() e instanziarla poi in una variabile:

function Macchina() { //... } var macchina = new Macchina(); //notare in questo caso la distinzione tra maiuscolo e minuscolo

Per fare riferimento alle proprietà dell'oggetto, all'interno della funzione usiamo la parola chiave this:

function Macchina(mod) { //passiamo qualche parametro al costruttore... this.modello = mod; } var macchina = new Macchina(“Fiat Panda”); //posso modificare e accedere alle proprietà macchina.modello = “Fiat 500"; alert(macchina.modello);

Per creare un metodo inseriamo invece una funzione all'interno del costruttore:

function Macchina(mod) { //passiamo qualche parametro al costruttore... this.modello = mod; this.carburante = 0; function rifornisci(euro) { var litri = euro/prezzo_benzina; this.carburante += litri; } }

La creazione di oggetti personalizzati è in realtà piuttosto inutile nei casi presentati dagli esempi, ma può risultare comoda per applicazioni più complesse: l'uso degli oggetti permette un'organizzazione dei dati chiara ed efficiente.

87 Modificare gli oggetti predefiniti

Molto interessante è anche la possibilità di accedere al **prototipo** di un oggetto, permettendo così di modificare anche tutte le sue istanze.

Ad esempio, il seguente spezzone di codice aggiunge a tutti gli array un metodo che prende un valore e restituisce true se è presente nell'array. Per fare questo accediamo al suo prototipo:

Array.prototype.contains = function (element) { for(i in this) { //scorre tutti gli elementi //se trova l'elemento restituisce true ed esce dalla funzione if (this[i] == element) return true; } return false; //questa riga verrà eseguita solo se non viene trovato nulla } var arr = new Array('a', 'b', 'c', 'd', 'e', 'f'); if (arr.contains('g')) { //falso //... } if (arr.contains('a')) { //vero //... }

88 Parole riservate

break //Interrompe cicli o istruzioni case case //Definisce un'istruzione in un costrutto switch catch continue //Continua l'esecuzione di un ciclo default //Definisce un'istruzione di default in un costrutto switch delete else //Definisce un codice alternativo in un costrutto if false //Valore booleano falso o 0 finally for //Ciclo for function //Definisce una funzione if //Costrutto if in //Itera in un array dentro un ciclo instanceof new //Definisce un nuovo oggetto null //Valore nullo return //Definisce il valore ritornato da una funzione/metodo switch //Costrutto switch this //Indica l'elemento corrente throw true //Valore booleano vero o 1 try typeof var //Definisce una nuova variabile void while

//Ciclo while do //Esegue del codice a prescindere dalle condizioni prima di un ciclo while with

89 Debugging

Il termine inglese *debugging* significa “togliere i bachi”, dove i bachi sono i *bug*, gli errori nel codice.

I tipi di errore possono essere diversi: dai semplici **errori di sintassi** (possono essere paragonati ad una frase grammaticalmente scorretta in italiano) come la mancata chiusura di una parentesi, fino agli **errori logici** (come una frase che non ha alcun senso, nonostante sia grammaticalmente corretta); ad esempio, può essere un errore logico dimenticarsi di incrementare una variabile quando serve, oppure utilizzare un algoritmo sbagliato.

89.1 Come evitare alcuni errori comuni

- Leggi attentamente il tuo codice per eventuali errori di battitura
- Assicurati che tutte le parentesi siano chiuse (può aiutare in questo un buon editor)
- Ricorda che JavaScript è *case-sensitive*: Name e name sono due variabili differenti
- Non usare parole riservate come nomi di variabili o di funzioni
- Ricordati di effettuare l'*escape* degli apici singoli/doppi:
- Quando converti i numeri usando la funzione parseInt ricorda che “08” o “09” sono valutati come numeri in base otto, a causa dello zero.
- Ricorda che JavaScript è indipendente dalla piattaforma, ma non lo è dal browser!

89.2 Semplice debugging

Oltre a verificare la lista degli errori comuni, uno dei modi più semplici di effettuare il debugging del proprio codice consiste nell'utilizzare le istruzioni alert mostrando il contenuto di una variabile, oppure semplicemente come “segnaposto” per sapere a che punto si è del codice.

Supponiamo ad esempio che lo script non faccia quanto desiderato, magari fermandosi a metà: come si può individuare il punto esatto in cui lo script si ferma? Il modo più semplice è inserire, prima e/o dopo i blocchi di istruzioni che consideriamo “sensibili” o di cui non siamo certi della correttezza, istruzioni del tipo:

//... alert (“inizio blocco”); //... alert (“fine blocco”); //...

Per esempio, se verrà mostrato il primo messaggio, significa che fino a quel punto è tutto corretto ma se, ad esempio, non viene mostrato il secondo, significa che in mezzo ai due *alert* c'è l'istruzione “buggata”.

Questo sistema è utile ad esempio nei cicli: per controllare ad ogni passaggio l'andamento del programma può essere comodo inserire nel ciclo un *alert* che mostri il valore del contatore e il valore delle altre variabili che intervengono nel programma, così da capire passo per passo cosa avviene effettivamente.

90 Editor e strumenti utili

Un buon editor su Windows per JavaScript (e anche altri linguaggi) è Notepad++ che fornisce evidenziazione della sintassi per JavaScript e non solo.

Per Linux due editor molto diffusi sono Bluefish per Gnome e Quanta Plus per KDE.

Esistono poi alcuni utili estensioni di Firefox per i web-developer, come la Web Developer Toolbar, che aggiunge una toolbar con funzionalità specifiche per gli sviluppatori web, oppure Console2, un “upgrade” della normale console degli errori di Firefox.

91 Link utili

- (IT) javaScript.html.it – Raccolta di guide, articoli e script dedicati
- (EN) w3schools.com – Tutorials, guide e reference.

Grazie a tutti quelli che hanno contribuito.
Se sei uno di questi scrivi il tuo nome qui:

- Ramac (disc.)
- Luca Polpettini (disc.)

92 Fonti per testo e immagini; autori; licenze

92.1 Testo

- **JavaScript/Versione stampabile** *Fonte:* https://it.wikibooks.org/wiki/JavaScript/Versione_stampabile?oldid=315423 *Contributori:* JackPotte

92.2 Immagini

- **File:CC_some_rights_reserved.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/7/79/CC_some_rights_reserved.svg *Licenza:* Public domain *Contributori:* Questo file deriva da: CC SomeRightsReserved.png
Artista originale: **Opera derivata:** Jelte
- **File:Cc-by_new_white.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/1/11/Cc-by_new_white.svg *Licenza:* Public domain *Contributori:* Reference icons : http://creativecommons.org/about/licenses and http://creativecommons.org/licenses/by/2.0/fr *Artista originale:* User:Sting
- **File:Cc-sa_white.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/d/df/Cc-sa_white.svg *Licenza:* Public domain *Contributori:* http://creativecommons.org/licenses/, Opera propria *Artista originale:* Rafał Pocztarski
- **File:HTML_paragraph_DOM.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/7/7a/HTML_paragraph_DOM.svg *Licenza:* Public domain *Contributori:* Opera propria *Artista originale:* Ramac
- **File:Heckert_GNU_white.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/2/22/Heckert_GNU_white.svg *Licenza:* CC BY-SA 2.0 *Contributori:* gnu.org *Artista originale:* Aurelio A. Heckert <aurium@gmail.com>
- **File:JavaScript_AlertBox.jpg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/b/bb/JavaScript_AlertBox.jpg *Licenza:* Public domain *Contributori:* selbst erstellter Screenshot *Artista originale:* S.Möller
- **File:JavaScript_Doppelt_1.jpg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/2/29/JavaScript_Doppelt_1.jpg *Licenza:* Public domain *Contributori:* selbst erstellter Screenshot *Artista originale:* S.Möller
- **File:Js_confirm.PNG** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/2/29/Js_confirm.PNG *Licenza:* MPL 1.1 *Contributori:* Opera propria *Artista originale:* Magik-Orion
- **File:Listato_esempio_javascript.png** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/1/13/Listato_esempio_javascript.png *Licenza:* Public domain *Contributori:* Screenshot from Bluefish *Artista originale:* Myself
- **File:Questionmark_copyright.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/b/bf/Questionmark_copyright.svg *Licenza:* CC-BY-SA-3.0 *Contributori:* Self-published work by Editor at Large *Artista originale:* Editor at Large, Ttog, Stephan Baum
- **File:Searchtool.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/6/61/Searchtool.svg *Licenza:* LGPL *Contributori:* http://ftp.gnome.org/pub/GNOME/sources/gnome-themes-extras/0.9/gnome-themes-extras-0.9.0.tar.gz *Artista originale:* David Vignoni, Ysangkok
- **File:Simpe_HTML_page_DOM.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/8/8b/Simpe_HTML_page_DOM.svg *Licenza:* Public domain *Contributori:* Opera propria *Artista originale:* Ramac
- **File:To_Commons.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/7/78/To_Commons.svg *Licenza:* CC BY-SA 3.0 *Contributori:* Opera propria *Artista originale:*
- pl.wiki: WarX
- **File:Wikipedia-logo-v2.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/8/80/Wikipedia-logo-v2.svg *Licenza:* CC BY-SA 3.0 *Contributori:* File:Wikipedia-logo.svg as of 2010-05-14T23:16:42 *Artista originale:* version 1 by Nohat (concept by Paullusmagnus); Wikimedia.
- **File:Wikipedia-logo.svg** *Fonte:* https://upload.wikimedia.org/wikipedia/commons/8/80/Wikipedia-logo-v2.svg *Licenza:* CC BY-SA 3.0 *Contributori:* File:Wikipedia-logo.svg as of 2010-05-14T23:16:42 *Artista originale:* version 1 by Nohat (concept by Paullusmagnus); Wikimedia.

92.3 Licenza dell'opera

- Creative Commons Attribution-Share Alike 3.0