Javascript

Giuseppe Della Penna
Università degli Studi di L'Aquila

dellapenna@univaq.it
http://www.di.univaq.it/gdellape

Usare Javascript

Script nella pagine HTML

- Per incorporare uno script in una pagina HTML, si utilizza il tag <script> con attributo type impostato al valore "text/javascript".
  - (!) Per compatibilità con i browser più vecchi, il codice all'interno del tag script viene a volte posto tra tag di commento HTML: <!-- e -->.
- E' anche possibile caricare script esterni, lasciando il tag <script> vuoto e specificando la URI dello script tramite l'attributo src.
  - (!) Per compatibilità con alcuni browser, non scrivere mai il tag script vuoto nella forma abbreviata, ma inserire sempre al suo interno del testo, ad esempio "/* */".
- Poiché gli script contengono caratteri riservati di XML, andrebbero sempre racchiusi in **sezioni CDATA**.
  - (!) Per compatibilità con alcuni browser, è necessario commentare (con //) le linee contenenti l'apertura e la chiusura della sezione CDATA.
- Si possono inserire e importare più script diversi all'interno dello stesso documento.
- Esistono inoltre alcuni attributi HTML in cui si può incorporare del codice:
  - Gli attributi per la gestione degli eventi, come onclick, possono contenere frammenti di codice (ma non dichiarazioni), da eseguire al verificarsi dell'evento.
  - L'attributo href del tag <a> può fare riferimento a una funzione javascript con la sintassi: "javascript:nomefunzione(argomenti)". In questo caso, il click del link eseguirà la chiamata alla funzione.

Script nella pagine HTML

Esempi

```
<script type="text/javascript">
//<![CDATA[
   var s = “pluto”;
//]]>
</script>

<script type="text/javascript“ src=“script.js”>
/* */
</script>
```

Usare Javascript

Esecuzione degli Script nella pagine HTML

- Il tag <script> può apparire sia nella <head>, dove viene normalmente posto per la maggior parte degli script, sia in qualunque punto del <body>.
- Tutte le funzioni e le variabili dichiarate negli script diventano disponibili (quindi possono essere usate e chiamate) non appena il parser analizza il punto della pagina in cui sono dichiarate.
- Se uno script contiene codice immediato, cioè scritto al di fuori di funzioni, questo viene eseguito non appena il parser analizza il punto della pagina in cui il codice compare.
  - (i) In questo modo, ad esempio, si può fare in modo che uno script venga valutato solo dopo che l'elemento HTML a cui si riferisce è stato caricato.
- Gli script possono utilizzare liberamente funzioni e variabili dichiarate in altri script inseriti nella stessa pagina.

Tipi di Dato

Javascript gestisce quattro diversi tipi di dato:

- **Numeri**
  - Non c'è distinzione tra interi e reali.
  - Sono costanti numeriche tutte le espressioni che rappresentano un numero valido, con e senza virgola.
  - (i) Le funzioni *parseInt* e *parseFloat* possono essere usate per convertire stringhe in numeri.
- **Booleani**
  - Le costanti booooeane sono *true* e *false*.
- **Stringhe**
  - Le stringhe sono particolari oggetti Javascript. Possono essere create implicitamente, attraverso una costante di tipo stringa, o esplicitamente tramite il costruttore dell'oggetto String.
  - Sono costanti di tipo stringa i valori racchiusi tra virgolette (singole o doppie).
- **Oggetti**
  - Gli oggetti sono un tipo di dato molto comune in javascript.
  - Le variabili di tipo oggetto contengono in effetti dei *riferimenti* ad oggetti. Più variabili possono quindi fare riferimento allo *stesso* oggetto.
- **Null**
  - Il tipo nullo ha un unico valore, null.

Variabili

- Le variabili javascript sono identificate da sequenze alfanumeriche il cui primo carattere deve essere alfabetico.
- Le variabili non hanno un tipo: questo viene dedotto automaticamente dal valore assegnato alla variabile, e può cambiare di volta in volta.
- Le variabili possono essere dichiarate scrivendo "**var** *nome*"
  - Il valore iniziale di una variabile è sempre lo speciale valore *undefined*.
  - E' anche possibile inizializzare la variabile durante la sua dichiarazione scrivendo "**var** *nome* = *valore*"
- Se si assegna un valore a una variabile non dichiarata, javascript la crea automaticamente. Questa pratica è **sconsigliata** perché le variabili create automaticamente **sono sempre globali**.
- Se si cerca di leggere il valore di una variabile mai dichiarata né assegnata, questo risulterà *undefined*.

Variabili

Esempi

```
var o = new Object(); //o è una variabile di tipo Object (vuota)

var s = "pluto"; //s è una variabile String con valore "pluto"

var n = 3; //n è una variabile Number con valore 3

m = n; //m è una variabile globale di tipo Number con valore 3

t = "paperino" //t è una variabile globale di tipo String con valore "paperino"

u = v //u ha valore undefined (in quanto v non è a sua volta definita)

var b = (3>2) //b è una variabile Boolean con valore true
```

Operatori

- **+ (somma)**
  Oltre che a numeri, può essere applicata a stringhe, nel qual caso diventa l'operatore di concatenazione. Se in una somma almeno un operando è una stringa, gli altri vengono convertiti anch'essi in stringhe.
- **- (differenza), / (quoziente), * (prodotto), % (modulo)**
- **= assegnamento, +=, -= (assegnamento con somma/differenza)**
  Gli assegnamenti con somma/differenza hanno la stessa semantica dei loro corrispondenti in C o Java. L'assegnamento con somma può essere applicato anche a stringhe, esattamente come l'operatore di somma.
- **++ (incremento), -- (decremento)**
- **>> (shift right), << (shift left), & (and), | (or), ^ (xor), ~ (not)**
  Effettuano bit-a-bit tra operandi numerici.

Operatori

- **&& (and), || (or), ! (not)**
  Utilizzati per combinare espressioni booleane.
- **in (appartenenza)**
  Può essere applicato a oggetti o array (operando destro), per controllare se contengono se la proprietà o l'indice dati (operando sinistro).
- **> (maggiore), < (minore), >= (maggiore o uguale), <= (minore o uguale), == (uguale), != (diverso)**
  Funzionano anche con le stringhe, per le quali si considera l'ordinamento lessicografico.
- **typeof(…) (controllo tipo)**
  Restituisce una stringa contenente il nome del tipo del suo argomento
- **void(…) (statement nullo)**
  Esegue il codice passato come argomento, ma non restituisce l'eventuale valore di ritorno
- **eval(…) (valutazione script)**
  Esegue lo script passato nell'argomento stringa e restituisce il suo valore

Operatori

Esempi

```
var s = "tre " + 2; //s è la stringa "tre 2"

s += " uno"; //s è la stringa "tre 2 uno"

s > "ciao"; //l'espressione vale true, in quanto il valore di s è lessicograficamente successivo a "ciao"

typeof(s); //restituisce "string"

var o = {pippo: 1};
"pippo" in o; //l'espressione vale true, in quanto pippo è una proprietà di o

void(0); //statement nullo (utile in HTML per prevenire azioni di default)
void(f(x)); //esegue f(x) ed ignora il suo valore di ritorno

eval("f(x)"); //esegue lo script, chiamando f(x) e restituendo il valore di ritorno della chiamata
eval("3+1"); //restituisce 4
eval("var s = 1"); //dichiara globalmente la variabile s e le assegna il valore 1.
```

Strutture di Controllo

Esecuzione condizionale

- Javascript dispone del costrutto **if** con la stessa sintassi di Java:

  if (*espressione***) {***corpo***} else {***corpo-else***}**

- Un'espressione guardia non booleana viene convertita in valore booleano come segue:
  - Se l'espressione ha un valore numerico diverso da zero è true.
  - Se l'espressione ha un valore stringa non vuoto è true.
  - Se l'espressione ha un valore oggetto è true.
  - In tutti gli altri casi (numerico zero, stringa vuota, valore undefined o null) l'espressione è false.
- (i) Per eseguire delle istruzioni solo se una determinata variabile o proprietà è definita e non vuota, è sufficiente scrivere **if (variabile) {…}**

Strutture di Controllo

Esecuzione condizionale

- Javascript dispone del costrutto **switch** con la stessa sintassi di Java:

  switch (*espressione*) {
  case *v1*: *istruzioni*
  case *v2*: *istruzioni*
  default: *istruzioni*
  }

- L'espressione viene valutata e confrontata con i valori dei diversi **case**. Vengono quindi eseguite le istruzioni *a partire dal primo **case*** con lo stesso valore dell'espressione. Se nessun case è selezionato, vengono eseguite le istruzioni del **default**, se presenti.
- Se si desidera limitare l'esecuzione a un gruppo di istruzioni, è necessario introdurre la parola chiave **break**.

Strutture di Controllo

Esempi

```
var s = "valore";
var b = 0;

//costrutto if con condizione di tipo "misto"
if (s && b > 0) { s = "ok" } else { b = 1; }

//costrutto switch su stringa
switch (s) {
   case "ok": …
   break; //questo case finisce qui
   case "error": … //questo case continua sul default
   default: …
}
```

Strutture di Controllo

Loops

- Javascript dispone dei costrutti **for**, **while** e **do…while**, con la stessa sintassi di Java:

  for (*inizializzazione*; *condizione*; *aggiornamento*) {*corpo*}

  while(*condizione*) {*corpo*}

  do {*corpo*} **while**(*condizione*)
- Nel ciclo **for** vengono eseguite le istruzioni di *inizializzazione*, quindi se la *condizione* è vera viene eseguito il *corpo* dell'istruzione e di seguito le istruzioni di *aggiornamento*. Se la *condizione* è ancora vera il ciclo continua.
- Nel ciclo **while** il *corpo* viene eseguito se e finché la *condizione* è vera.
- Nel ciclo **do…while** il *corpo* viene eseguito almeno una volta, in quanto la *condizione* è testata al termine della sua esecuzione.
- Nel corpo dei loop è possibile usare le parole chiave **break** e **continue** rispettivamente per interromperne l'esecuzione o per saltare direttamente al ciclo successivo.

Strutture di Controllo

Il loop for…in

- Una speciale forma del costrutto **for** permette di iterare tra tutte le proprietà di un oggetto:

  for (*proprietà* **in** *oggetto*) {*corpo*}

- Ad ogni iterazione, la stringa *proprietà* conterrà il nome della successiva proprietà dell'*oggetto*
  - (i) Sarà quindi possibile accedere alla proprietà scrivendo *oggetto*[*proprietà*] (sintassi array).
  - (!) Poiché i metodi sono proprietà con un particolare tipo, anche questi verranno elencati dal loop.

Strutture di Controllo

Esempi

```
var o = new Object();

//itera tra le proprietà dell'oggetto
for (p in o) {
   o[p]; //preleva il valore della proprietà indicizzata dal loop (e dovrebbe usarla…)
}

var i = 0;

//ciclo for standard
for (i=0; i<10; ++i) { j=j+1; }

//ciclo while
while(i>0) { i=i+1; }

//ciclo do
do { i=i+1; } while(i>0);
```

Funzioni

Dichiarazione

- In Javascript è possibile creare nuove funzioni con le seguenti sintassi:
  - Dichiarazione di funzione: **function** *nome*(*parametri*) {*corpo*}
  - Funzioni anonime: **function**(*parametri*) {*corpo*}
  - Oggetti funzione: **new Function**("*parametri*", "*corpo*")
- Le diverse sintassi presentano specifiche caratteristiche e limitazioni:
  - Una funzione dichiarata *con un nome* può essere richiamata in ogni punto del codice tramite il suo nome.
  - Una funzione anonima o creata col costruttore **Function** deve essere assegnata a una variabile (o proprietà di un oggetto) per essere utilizzata.
- Il **nome** di una funzione può essere qualsiasi nome valido per una variabile.
- I **parametri** della funzione, se presenti, sono dichiarati tramite una lista di nomi (di variabile) separati da virgole.
  - (!) Le parentesi dopo il nome della funzione vanno sempre inserite, anche se la lista dei parametri è vuota.
- Il **corpo** della funzione è costituito da una sequenza di istruzioni Javascript valide.
  - Ogni istruzione è separata dalla successiva da un punto e virgola.
  - Nel corpo è possibile utilizzare i parametri tramite il nome delle variabili ad essi associate.

Funzioni

Esempi

```
//funzione senza parametri, dichiarazione diretta
function f() {
   var i;
}

//funzione con due parametri, dichiarazione diretta
function g(a,b) {
   var c = a + b;
}

//funzione anonima assegnata a una variabile
var h1 = function(a) {return a+1;}

//oggetto funzione assegnato a una variabile
var h2 = new Function("a","return a+1;");
```

Funzioni

Riferimento

- Le funzioni Javascript sono in realtà variabili con valore di tipo **Function**.
- Per fare riferimento a una funzione è sufficiente usare il suo nome, o un'espressione equivalente che abbia valore di tipo **Function**.
- Una volta ottenuto il riferimento a una funzione è possibile:
  - Chiamare la funzione passandole dei parametri.
  - Passare come argomento una funzione ad un'altra funzione.
  - Assegnare una funzione a una o più variabili.
  - Accedere a tutti gli elementi della funzione, per modificarla o ridefinirla, tramite le proprietà di **Function**.
  - Verificare se una funzione è definita come si farebbe con qualsiasi variabile, ad esempio testandola con un **if**(nome_funzione).

Funzioni

Chiamata

- Per richiamare una funzione, si accoda la lista dei parametri, tra parentesi, all'espressione che fa riferimento alla funzione stessa:

  nome_funzione(*argomenti*)

  espressione_con_valore_Function(*argomenti*)

- Gli argomenti sono una lista di espressioni valide separate da virgole.
  - (i) è possibile omettere uno o più parametri al termine della lista. In questo caso, tali parametri varranno *undefined* nel corpo della funzione.
  - (!) se la funzione non ha parametri, è comunque necessario specificare le due parentesi dopo il nome.

Funzioni

Esempi

```
function f() { var i; }
var h1 = function(a) {return a+1;}
var h2 = new Function("a","return a+1;");

f(); //ritorna undefined

var r = h1(3); //r=4

var r2 = h2(4); //r=5
```

Funzioni

Passaggio di Parametri

- Il passaggio dei parametri alle funzioni Javascript avviene in maniera diversa a seconda del tipo del parametro stesso:
  - I tipi booleano, stringa, numero e null sono passati *per valore*. Nella funzione, cioè, è presente una copia del valore usato come argomento. Cambiamenti locali alla funzione non influenzano il valore dell'argomento usato nella chiamata alla funzione stessa.
  - Il tipo oggetto è passato *per riferimento*. La manipolazione del contenuto dell'oggetto si riflette sull'oggetto usato come argomento.

Funzioni
Ritorno

- Le funzioni restituiscono il controllo al chiamante al termine del loro blocco di istruzioni.
- E' possibile *restituire un valore* al chiamante, in modo da poter usare la funzione in espressioni più complesse, utilizzando la sintassi

  return *espressione*

- L'*espressione* può essere di qualsiasi tipo. Essa viene valutata e il valore risultante è restituito.
  - (i) Se la funzione non esegue una return, Javascript sottintende un "return undefined" implicito.

Funzioni

Closures

- Una **closure** (chiusura) è, tecnicamente, un'espressione (tipicamente una funzione) *associata a un contesto che valorizza le sue variabili libere*.
- Tutto il codice Javascript viene eseguito in un contesto, compreso quello globale.
- In particolare, ogni esecuzione di una funzione ha un contesto associato.
- Una *closure* si crea proprio a partire da una funzione, quando quest'ultima restituisce come valore di ritorno una nuova funzione creata dinamicamente (cioè con uno dei tre costrutti visti in precedenza).

Funzioni

Comportamento delle closures

- Una *closure*, cioè una funzione creata all'interno di un'altra funzione e poi restituita, *mantiene il contesto di esecuzione della funzione che l'ha creata*.
- Questo significa che il contesto di ciascuna chiamata della funzione "generatrice" non viene distrutto all'uscita della funzione, come avviene in generale, ma conservato in memoria.
- La *closure* potrà fare riferimento (in lettura e scrittura) ai parametri e alle variabili dichiarate nel contesto della funzione che l'ha creata.
- Poiché ogni chiamata a funzione ha un suo contesto distinto, i valori "visti" dalla *closure* non saranno influenzati da successive chiamate alla funzione generatrice.

Funzioni

Closures: Esempi

- Un uso comune per le closure è fornire parametri a una funzione che verrà eseguita in seguito: è il caso ad esempio delle funzioni passate come argomento a *setTimeout* (che verrà illustrata più avanti).
- Se dobbiamo passare una funzione come argomento, o assegnarla a una variabile, non possiamo fornirgli parametri, ma al posto della funzione possiamo usare una *wrapper closure* che la chiama con i parametri desiderati.
- Si vedano gli esempi seguenti…

Funzioni

Closures: Esempi

```
function f(x) {
   return x+variabile_globale; //NOTA: il valore di ritorno NON dipende solo dai parametri
}
//vorremmo assegnare a una proprietà p di o la FUNZIONE che restituisce f(3)
o.p = f(3);
o.p(); //ERRATO: o.p non punta a una funzione, ma al valore calcolato come f(3) al momento dell'esecuzione dell'istruzione precedente

o.p = f;
o.p(); //ERRATO: o.p è un riferimento a f, per cui necessita di un parametro (avremmo dovuto scrivere o.p(3))

function closureGenF(y) {
   return function() {return f(y);}
}
o.p = closureGenF(3);
o.p() //CORRETTO: verrà chiamata f(3)!
```

Funzioni

Closures: Esempi

```
//dobbiamo assegnare un handler per l'evento onclick a un elemento del DOM HTML
htmlelement.onclick = f; //NOTA: anche qui non si possono passare parametri!

//Accade spesso che si usino piccole varianti dello stesso handler per elementi diversi
//tali varianti sono semplici da costruire a tempo di esecuzione (e spesso è necessario).
//Ad esempio, vogliamo associare a certi elementi degli handler che colorino in rosso un elemento
ad essi associato quando vengono cliccati.

function clickHandler(oToHighlight) {
   return function(e) {
      oToHighlight.style.backgroundColor="red";
   }
}

element1.onclick = clickHandler(linkedelement1);
element2.onclick = clickHandler(linkedelement2);
```

Oggetti Javascript

- Javascript **non è un linguaggio object oriented**, e il suo concetto di oggetto è molto più simile a quello di un array associativo.
  - (i) Gli oggetti Javascript contengono metodi, che possono tuttavia essere considerati anch'essi valori, in quanto non sono altro che oggetti di classe **Function**.
- In Javascript non si possono definire **classi**, ma solo speciali funzioni dette **costruttori** che creano oggetti aventi determinati membri. Il nome della funzione costruttore è considerato il nome della classe dell'oggetto.
- Non esiste vera ereditarietà negli oggetti Javascript, e non è possibile dichiarare delle gerarchie. Tuttavia, Javascript contiene una classe base predefinita, chiamato **Object**.
- Gli oggetti si creano utilizzando l'operatore *new* applicato alla loro **funzione costruttore**: o = **new Object**();
- Un metodo di creazione alternativo consiste nell'utilizzo del costrutto { "proprieta": valore, … } , che crea un oggetto con le proprietà date.

Oggetti Javascript

Le proprietà

- Le proprietà di un oggetto Javascript possono contenere valori di qualsiasi tipo.
- Per accedere a una proprietà, si possono usare due sintassi:
  - Sintassi "a oggetti": *oggetto.proprietà*
  - Sintassi "array": *oggetto*["*proprietà*"]
- È disponibile lo speciale costrutto **for…in** per iterare tra le proprietà di un oggetto.
- E' possibile verificare se un oggetto ha una determinata proprietà con l'espressione booleana *proprietà* **in** *oggetto*.
- Se si tenta di leggere il valore di una proprietà non definita in un oggetto, si ottiene il valore *undefined* (come per ogni variabile non assegnata).
- È possibile **aggiungere dinamicamente proprietà** agli oggetti semplicemente assegnando loro un valore.
  - (!) Non è possibile aggiungere proprietà a variabili che non siano di tipo oggetto (oggetti predefiniti o creati con *new*)

Oggetti Javascript

Proprietà-Esempi

```
var o = new Object();

var v = o.pippo; //v è undefined

o.pluto = 3; //adesso o ha una proprietà "pluto", di tipo Number, con valore 3

v = o.pluto; //adesso v è una variabile di tipo Number e vale 3

v.paperino = "ciao"; //è un errore: una variabile può accettare l'aggiunta di proprietà solo se è di tipo oggetto

var o2 = {"pippo": "ciao", "pluto": 3}; //creazione implicita di un oggetto

v = o2["pluto"]; //equivalente a v =o2.pluto

var nome = "pippo"; //adesso nome è una variabile di tipo String e vale "pippo"
v=o2[nome]; //accesso a una proprietà con nome dinamico assegnato a una seconda variabile
```

Oggetti Javascript

I Metodi

- I metodi di un oggetto Javascript sono semplicemente proprietà di tipo *Function*.
  - (i) **Function** è un oggetto predefinito Javascript, e può essere utilizzato direttamente, ad esempio per creare funzioni anonime.
- Per **accedere** a un metodo si possono usare le stesse sintassi viste per le proprietà.
- Per **chiamare** un metodo basta accodare la lista dei parametri, tra parentesi, all'espressione di accesso al metodo.
- Per aggiungere un metodo a un oggetto, è sufficiente creare una proprietà col nome del metodo ed assegnarvi:
  - Una funzione già definita
  - Una funzione anonima: **function**(*parametri*) {*corpo*}
- I metodi possono essere aggiunti in qualsiasi momento a un oggetto, esattamente come le proprietà.
- I metodi di un oggetto, per far riferimento alle proprietà dell'oggetto in cui sono definiti, devono utilizzare la parola chiave **this**: **this**.proprietà.
  - (!) Omettendo **this**, Javascript cercherà le variabili col nome dato all'interno del metodo o tra le variabili globali!

Oggetti Javascript

Metodi-Esempi

```
var o = new Object();

o.metodo1 = function(x) {return x;} //aggiunge la funzione specificata come metodo1 all'oggetto

o["metodo2"] = f; //aggiunge la funzione f (se esistente) come metodo2 all'oggetto

o.metodo1 //questa espressione restituisce l'oggetto Function che rappresenta il meotodo1

o.metodo1(3);
o["metodo1"](3); //due chiamate equivalenti al meotodo1

var o2 = {"pippo": "ciao", "pluto": 3, "metodo3": function(x) {return x;}} //definizione di un metodo
all'interno della sintassi abbreviata di creazione

var o3 = new Object();
o3.metodo3 = o.metodo1 //il metodo3 dell'oggetto o3 è una copia del metodo1 dell'oggetto o
```

Oggetti Javascript

Funzioni Costruttore

- Una funzione costruttore è un tipo speciale di funzione all'interno della quale
  - Si utilizza la parola chiave **this** per *definire* le proprietà di un nuovo oggetto.
  - Non si ritorna alcun valore
- Le funzioni costruttore possono essere usate come argomento per l'operatore **new**, esattamente come i nomi degli oggetti standard di Javascript:

  new *funzione*(*parametri*)
- (!) Le funzioni costruttore non dovrebbero mai essere richiamate direttamente.

Oggetti Javascript

Funzioni Costruttore

- Quando si usa un costruttore con **new**, Javascript crea un oggetto vuoto derivato da **Object** ed applica ad esso la funzione.
- All'interno del costruttore, **this** punta al nuovo oggetto.
- In questo modo, il costruttore può popolare il nuovo oggetto, aggiungendo proprietà e metodi attraverso **this**.
- (!) Va ricordato che i metodi inseriti in un oggetto, per fare riferimento alle proprietà dell'oggetto stesso, devono riferirvisi attraverso **this**.

Oggetti Javascript

Funzioni Costruttore-Esempi

```
function myObject(a) {
   this.v = a+1;
   this.w = 0;
   this.m = function(x) {return this.v+x;}
}

/*
L'oggetto o avrà due proprietà (v e w), una delle quali inizializzata tramite il parametro della funzione
costruttore, e un metodo (m) che restituisce il valore della proprietà v sommata al suo argomento
*/

var o = new myObject(2);
o.m(3); //ritorna 6;

o.getW = function() {return this.w;} //aggiunta dinamica di membri all'oggetto (NON al costruttore)
o.getV = function() {return v;} //SBAGLIATO! Fa riferimento alla variabile GLOBALE v!
```

Oggetti Predefiniti Javascript

String

- Gli oggetti **String** sono usati in Javascript per contenere stringhe di caratteri. Possono essere creati implicitamente, utilizzando una costante stringa, o esplicitamente tramite il costruttore:

  s = **new String**(*valore*)

- I principali metodi e proprietà della classe **String** sono i seguenti:
  - **length**
    restituisce la lunghezza della stringa.
  - **charAt(***posizione***)**
    restituisce il carattere (stringa di lunghezza uno) alla posizione data (base zero).
  - **charCodeAt(***posizione***)**
    come charAt, ma restituisce il codice ASCII del carattere.
  - **indexOf***(s,offset)*
    restituisce la posizione della prima occorrenza (a partire da offset, se specificato) di s nella stringa. Restituisce -1 se s non è una sottostringa della stranga data (a partire dall'ofset).
  - **lastIndexOf***(s,offset)*
    come indexOf, ma restituisce la posizione dell'ultima occorrenza.
  - **substr***(os[,l])*
    restituisce la sottostringa di lunghezza l (default, la massima possibile) che inizia os caratteri dall'inizio della stringa
  - **substring***(os,oe)*
    restituisce la sottostringa che inizia os caratteri e termina a oe caratteri dall'inizio della stringa
  - **toLowerCase()**
    ritorna la stringa convertita in minuscolo
  - **toUpperCase()**
    ritorna la stringa convertita in maiuscolo

Oggetti Predefiniti Javascript

RegExp e String

- Javascript riconosce le espressioni regolari scritte nella sintassi Perl.
- Per descrivere un'espressione regolare costante è sufficiente usare la sintassi */espressione/*.
  - (i) Espressioni regolari variabili possono essere create tramite il costruttore **RegExp**.
- E' possibile usare le espressioni regolari in vari metodi della classe **String**:
  - **match***(r)*
    restituisce l'array con le sottostrighe che hanno fatto match con l'espressione regolare r.
  - **replace***(r,s)*
    sostituisce tutte le sottostringhe cha fanno match con r con la stringa s.
  - **search***(r)*
    restituisce la posizione della prima sottostringa che fa match con r, o -1 se non ci sono match.
  - **split***(r[,m])*
    divide la stringa in una serie di segmenti definiti dai separatori specificati con l'espressione r e li restituisce come array. Se si indica una lunghezza massima m, allora l'ultimo elemento dell'array conterrà la rimanente parte della stringa.
- (!) Per default, Javascript interrompe il processo di matching su una stringa appena trova un riscontro per l'espressione regolare. Per trovare tutti i riscontri possibili, usare il modificatore /g
- (i) Per rendere l'espressione *case insensitive*, usare il modificatore /i

Oggetti Predefiniti Javascript

Array

- Gli Array sono oggetti javascript predefiniti e possono contenere valori di qualsiasi tipo.
- Per **creare** un array si possono usare le seguenti sintassi:
  (costruttore con parametri multipli) *v* = **new Array**(*e1*,*e2*,*e3*,…)
  (costruttore simbolico) *v* = [*e1*,*e2*,*e3*,…]
- Per **accedere** a un elemento di un array si usa la sintassi comune *variabile_array*[*indice*]
- E' possibile verificare se un indice è presente nell'array con l'espressione booleana *indice* **in** *variabile_array*.
- I principali metodi e proprietà della classe **Array** sono i seguenti:
  - **length**
    restituisce la dimensione dell'array
  - **concat***(e1,e2,e3,…)*
    aggiunge gli elementi dati alla fine dell'array.
  - **join(***separatore***)**
    converte l'array in una stringa, concatenando la versione **String** di ciascun elemento ed usando il separatore dato (default ",").
  - **reverse()**
  - inverte l'ordine dell'array
  - **slice(***os[,l]***)**
  - restituisce il sotto array di lunghezza l (default, la massima possibile) che inizia all'indice os.
  - **sort***([sortfun])*
  - ordina l'array. La funzione opzionale sortfun può essere usata per specificare un criterio di ordinamento non standard.

Oggetti Predefiniti Javascript

Array-Esempi

```
var a1 = new Array(10,20,30); //dichiarazione con construtto new

var a2 = [“a”,”b”,”c”]; //dichiarazione implicita

for (i in a1) { a1[i]; } //itera nell’array (ma considera anche tutte le altre eventuali proprietà/metodi)

for (i=0; i<a1.length; ++i) { a1[i]; } //itera tra gli elementi dell’array

if (4 in a1) { a1[4] = a1[4]+1; } //usa un elemento solo se presente

/*
Nota: per creare un array associativo, è sufficiente creare dinamicamente delle proprietà (chiavi) in un oggetto Object vuoto
*/
```

Oggetti Predefiniti Javascript

L'oggetto Date

- L'oggetto **Date** permette di manipolare valori di tipo data e ora. Dispone di diversi costruttori:
  - **Date**() inizializza l'oggetto alla data/ora corrente.
  - **Date**(*y*,*m*,*d*,*hh*,*mm*,*ss*) inizializza l'oggetto alla data/ora *d*/*m*/*y hh*:*mm*:*ss*.
  - **Date**(*stringa*) tenta di riconoscere la *stringa* come una data e inizializza l'oggetto di conseguenza.
- Gli oggetti Date possono essere confrontati tra loro con i normali operatori di confronto.
- I metodi degli oggetti Date permettono di leggerne e scriverne tutti i membri:
  - Ad esempio, **getYear**, **getMonth**, **setYear**, **setMonth**, **getDay** (restituisce il *giorno della settimana*), **getDate** (restituisce il *giorno del mese*), **setDate** (imposta il giorno del mese: se il valore passato è maggiore del massimo consentito, *la funzione gestisce automaticamente l'incremento del mese/anno della data*)

Oggetti Predefiniti Javascript

L'oggetto Date-Esempi

```
//genera un saluto adeguato all'ora corrente e lo assegna alla variabile saluto
var giorni = ["lun","mar","mer","gio","ven","sab"];
var mesi = ["gen","feb","mar","apr","mag","giu","lug","ago","set","ott","nov","dic"];
var oggi = new Date();

var data = giorni[oggi.getDay()] + " " + oggi.getDate() + " " + mesi[oggi.getMonth()] + " "
+oggi.getFullYear();

var saluto;
if (oggi.getHours() > 12)  saluto = "Buona sera, oggi è il "+data;
else saluto = "Buongiorno, oggi è il "+data;



//calcola una la data di 70 giorni nel futuro
futuro = new Date();
futuro.setDate(domani.getDate()+70);
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto window

- Quando Javascript è usato all'interno del browser, sono disponibili alcuni oggetti particolari, relativi al browser stesso e alla pagina visualizzata.
- L'oggetto **window** è il punto di accesso a tutti gli oggetti esposti dal browser. Si tratta *dell'oggetto predefinito* per lo scripting, il che significa che tutte le sue proprietà e i suoi metodi sono accessibili a livello globale, senza bisogno di specificare esplicitamente l'oggetto window.
- L'interfaccia di window contiene alcune funzionalità molto utili, tra cui
  - Il metodo **alert**(*messaggio*), mostra il *messaggio* dato in un dialog box (con il solo bottone OK).
  - Il metodo **confirm**(*messaggio*), mostra il *messaggio* dato in un dialog box con I bottoni OK e Cancel. La funzione ritorna true se l'utente preme OK, false altrimenti.
  - Il metodo **prompt**(*messaggio*,*default*) mostra il *messaggio* dato in un dialog box, insieme a un campo di input con valore iniziale *default*. Se l'utente preme OK, il contenuto del campo (anche vuoto) di input viene restituito dalla funzione. Altrimenti la funzione restituisce null.
  - I metodi **setTimeout e setInterval** permettono di impostare un timer (si veda dopo)
  - La proprietà **document** permette di accedere al documento HTML visualizzato.
  - Altre proprietà, come ad esempio **statusbar**, sono supportate in maniera diversa dai browser.

Oggetti Predefiniti Javascript

L'oggetto window-Esempi

```
//esegue un'azione solo se l'utente clicca OK
if (window.confirm("Sei sicuro di voler lasciare questa pagina?")) {…}

//chiede all'untente di inserire un dato e lo avverte se non è stato specificato nulla
var citta = window.prompt("Luogo di Nascita","L'Aquila");
if (!citta) window.alert("Non hai specificato il luogo di nascita!");
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto document

- L'oggetto **document**, accessibile tramite la proprietà omonima di **window**, rappresenta il documento visibile nel browser.
- La maggior parte dei metodi e delle proprietà offerti dall'oggetto document provengono dall'interfaccia **Document**, che verrà trattata nell'ambito del *Document Object Model*. Esistono tuttavia alcune proprietà utili proprie del solo oggetto document, ad esempio
  - La proprietà **location** contiene la URL di provenienza del documento corrente.
  - La proprietà **lastModified** contiene la data dell'ultimo aggiornamento del documento.
  - Il metodo **open()** apre uno stream per la successiva scrittura di testo nel documento tramite **write()** e **writeln()**. All'apertura dello stream il contenuto corrente del documento viene cancellato.
  - Il metodo **write(***testo*) e **writeln**(*testo*) accodano *testo* (o *testo* seguito da un ritorno a capo) al documento corrente. Se non è stata chiamata la **open()**, ne viene generata una implicita. (!) *Questi metodi non si possono usare in XHTML*.
  - Il metodo **close**() chiude lo stream di scrittura aperto con **open**() e forza la visualizzazione di quanto scritto nel documento con **write**() e **writeln**(). Ogni successiva operazione di scrittura genererà una nuova open() implicita.
- (i) L'oggetto document fornisce anche un sistema "proprietario" di Javascript per l'accesso alla struttura del documento visualizzato. Nei browser moderni, con supporto al W3C DOM, l'uso di questo sistema è tuttavia fortemente sconsigliato.

Oggetti Predefiniti Javascript

L'oggetto document-Esempi

```
//crea un documento contenente una semplice tabella
var i,j;
document.open();
document.write("<table border='1'>");
for(i=0;i<10;++i) {
   document.write("<tr>");
   for(j=0;j<10;++j) {
      document.write("<td>"+i+","+j+"</td>");
   }
   document.write("</tr>");
}
document.write("</table>");
document.close();
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest

- L'oggetto **XMLHttpRequest**, originariamente introdotto da Internet Explorer, è ora supportato da tutti i browser più diffusi.
- Il suo scopo è quello di *permettere al codice Javascript l'esecuzione di richieste HTTP verso il server* (proprio come farebbe il browser) e la gestione dei dati risultanti.
- Questo oggetto è alla base delle tecniche **AJAX**, con cui gli script che controllano una pagina web possono dialogare col server senza la necessità di "cambiare pagina".
- **(!)** Per motivi di sicurezza, l'oggetto XMLHttpRequest può effettuare connessioni *solo con l'host a cui appartiene la pagina* in cui ha sede lo script!

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: istanziazione

- *L'interfaccia di XMLHttpRequest* è standard, ma esistono sistemi browser-dipendenti per accedere a questo oggetto.
- Se nel browser è definito il costruttore omonimo (*typeof XMLHttpRequest != "undefined"*), è sufficiente eseguire una new:

  var xhr = new XMLHttpRequest();

- In Internet Explorer, si usa il costruttore ActiveXObject (*typeof ActiveXObject != "undefined"*) passandogli la stringa di identificazione dell'oggetto, che può essere "MSXML2.XmlHttp.6.0" (preferita) o "MSXML2.XmlHttp.3.0" (vecchie versioni del browser):

  var xhr = new ActiveXObject("MSXML2.XmlHttp.3.0");

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: istanziazione - Esempio

```
function createRequest() {
   var ACTIVEXIDs=["MSXML2.XmlHttp.6.0","MSXML2.XmlHttp.3.0"];
    if (typeof XMLHttpRequest != "undefined") {
        return new XMLHttpRequest();
    } else if (typeof ActiveXObject != "undefined") {
       for (var i=0; i < ACTIVEXIDs.length; i++) {
          try {
             return new ActiveXObject(ACTIVEXIDs[i]);
          } catch (oError) {
             //l'oggetto richiesto non esiste: proviamo il successivo
          }
          alert("Impossibile creare una XMLHttpRequest");
       }
   } else {
       alert("Impossibile creare una XMLHttpRequest");
    }
}
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: uso

- Il *pattern d'uso di XMLHttpRequest* è duplice, a seconda che si scelga la modalità di chiamata sincrona o asincrona:
- **Modalità sincrona**: la richiesta al server blocca lo script (e la pagina associata) finché non viene ricevuta la risposta.
- **Modalità asincrona**: la richiesta viene inviata, e lo script continua la sua esecuzione, venendo poi avvisato dell'arrivo della risposta tramite un *evento*.

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: uso sincrono

- Si prepara la richiesta usando il metodo ***open***, a cui si passano il verbo HTTP e la url da chiamare. Il terzo parametro deve essere *false* per avviare una richiesta sincrona:

  xhr.open("GET","http://pippo",false);

- Si invia la richiesta con il metodo ***send***, che risulta bloccante:

  xhr.send(null);

- Si controlla se la richiesta ha restituito un errore HTTP tramite la proprietà ***status***, ad esempio

  if (xhr.status !=404) {…}

- Si accede ai dati restituiti dal server (se necessario) tramite la proprietà ***responseText***

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: uso sincrono - Esempio

```
var req = createRequest();

req.open("GET",requrl,false);

req.send(null);

if (req.status!=404) {
   alert(req.responseText);
} else {
   alert("errore");
}
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: uso asincrono

- Si prepara la richiesta usando il metodo ***open***, a cui si passano il verbo HTTP e la url da chiamare. Il terzo parametro deve essere *true* per avviare una richiesta sincrona:

  xhr.open("GET","http://pippo",true);
- Si imposta l'*handler* da chiamare quando la richiesta sarà stata servita tramite la proprietà ***onreadystatechange***:

  xhr.onreadystatechange = function() {…};
- Si invia la richiesta con il metodo ***send*** (la chiamata ritorna immediatamente il controllo allo script):

  xhr.send(null);
- All'interno dell'event handler dichiarato, si verifica prima di tutto se la richiesta è stata servita controllando che la proprietà ***readyState*** sia uguale a 4:
  - If (xhr.readyState == 4) {…};
- Se la richiesta è stata servita, si può controllare se ha restituito un errore HTTP tramite la proprietà ***status*** e poi accedere ai dati restituiti dal server tramite la proprietà ***responseText***, come già illustrato.
- In ogni momento, è possibile invocare il metodo ***abort*** per interrompere la richiesta HTTP in corso.

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest: uso asincrono - Esempio

```
var req = createRequest();

req.open("GET",requrl,true);

req.onreadystatechange = function () {
   if (req.readyState==4) {
        if (req.status!=404) {
         alert(req.responseText);
        } else {
         alert("errore");
        }
   }
};

req.send(null);
```

Oggetti Predefiniti Javascript per i Browser

L'oggetto XMLHttpRequest e JSON

- Spesso, quando si scambiano dati con uno script tramite la XMLHttpRequest, accade che il server debba passare a Javascript *strutture dati complesse*, e non semplice testo o HTML.
- In questi casi, è utile usare la notazione **JSON**: in pratica, le strutture dati vengono trascritte testualmente usando la notazione "breve" Javascript per la definizione di oggetti ed array.
  - Ad esempio, la stringa che segue definisce (e in Javascript crea) un array contenente due record aventi come campi "id" e "nome": [{"id":1, "nome":"pippo"},{"id":2, "nome":"pluto"}]
- Una volta ricevuti questi dati in forma testuale, è possibile trasformarli in vere strutture dati all'interno dello script con una istruzione del tipo:

```
dati = new Function("return "+xhr.responseText)();
```

Gestione delle Eccezioni

- Nelle versioni più recenti di Javascript è stato introdotto anche un **sistema di gestione delle eccezioni in stile Java**.
- Un'eccezione segnala un *imprevisto*, spesso un *errore*, all'interno della normale esecuzione del codice.
- Un'eccezione può venire sollevata dalle librerie di Javascript o dal codice scritto dall'utente, attraverso la parola chiave **throw**.
- Per gestire le eccezioni, è possibile avvalersi del costrutto **try…catch…finally**.

Gestione delle Eccezioni

Gli handler

- Una volta sollevata, un'eccezione risale lo *stack* di Javascript finché non viene gestita.
  - Ciò significa che un'eccezione generata in una funzione, se non viene gestita all'interno di quest'ultima, si propagherà alle sue funzioni chiamanti, fino ad arrivare al *runtime* di Javascript.
- Per gestire le eccezioni generate da un certo blocco di codice, è necessario inserire il blocco all'interno del costrutto **try…catch**.
  - Qualsiasi eccezione sollevata all'interno del codice compreso tra **try** e catch verrà passata al codice di gestione dichiarato dopo **catch**.
- Se ci si vuole assicurare che un certo codice sia eseguito *sempre* dopo il blocco protetto da **try…catch**, indipendentemente dal sollevamento di eccezioni, è possibile aggiungere al blocco la clausola **finally**.

Gestione delle Eccezioni

Gli handler - Esempio

```
try {
   … codice…
} catch(e) {
   //le eccezioni generate da javascipt sono oggetti la cui proprietà message riporta
   //il messaggio di errore associato
   alert("Eccezione sollevata: "+ e.message);
}
try { …codice… } catch (eccezione) {…gestione dell'eccezione…}
finally {
   ...codice eseguito in ogni caso prima che l'esecuzioni continui oltre il blocco try…catch...
}
try {
   …codice…
   throw {"nome": "pippo", "valore": 1}; //solleviamo un oggetto qualsiasi come eccezione
} catch (ex) {
   //l'oggetto ex che arriva al blocco catch è quello sollevato con la throw!
}
```

Funzioni Javascript Utili

I timer

- Javascript, tramite l'oggetto **window**, permette di eseguire **azioni temporizzate**. A questo scopo si usano i seguenti metodi.
  - **setTimeout**(*stringa_o_funzione*,*millisecondi*,*arg1*,…,*argN*). La chiamata a questa funzione fa sì che, dopo il numero specificato di *millisecondi*, Javascript esegua il codice dato dal primo argomento, che può essere una *stringa* contenente del codice da valutare o il nome di una *funzione* da chiamare. In quest'ultimo caso, è possibile specificare *opzionalmente* una serie di argomenti (*arg1…argN*) da passare alla funzione. **L'azione viene quindi eseguita una sola volta.**
  - **setInterval(***stringa_o_funzione*,*millisecondi*,*arg1*,…,*argN*). La chiamata a questa funzione fa sì che, ogni *millisecondi*, Javascript esegua il codice dato dal primo argomento, che può essere una *stringa* contenente del codice da valutare o il nome di una *funzione* da chiamare. In quest'ultimo caso, è possibile specificare *opzionalmente* una serie di argomenti (*arg1…argN*) da passare alla funzione. **L'azione viene quindi eseguita periodicamente.**
- Entrambe le funzioni possono essere chiamate più volte, e restituiscono un *timer id* (numerico), tramite il quale è possibile annullare la temporizzazione usando le rispettive funzioni:
  - **clearTimeout**(*id*) per le temporizzazioni avviate con **setTimeout()**
  - **clearInterval**(*id*) per le temporizzazioni avviate con **setInterval()**

Funzioni Javascript Utili

I timer-Esempi

```
function saluta(nome) {
   alert("Ciao "+nome);
}

//Richiede il nome e saluta dopo cinque secondi
var nome = prompt("Come ti chiami?");
if (nome) setTimeout(saluta,5000,nome);


//avverte dell'ora corrente ogni minuto
setInterval("d=new Date(); alert('Ora sono le '+d.getHours()+':'+d.getMinutes())",60000);
```