Linguaggio Javascript

PROF. DIOMAIUTA CRESCENZO

Introduzione

- JavaScript è un linguaggio di scripting lato client utilizzato per rendere dinamico il codice HTML.
- Il documento HTML viene generato staticamente.
- Il codice JavaScript è inglobato nel documento HTML ma viene eseguito dinamicamente solo al momento della richiesta del web client (browser).
- Principali usi:
  - posizionamento dinamico degli oggetti
  - validazione campi dei moduli
  - effetti grafici
- Javascript non è compilato ed è ad alto livello

Sintassi

- Javascript è case-sensitive, ovvero distingue le lettere maiuscole dalle minuscole
- Commento multilinea /* e */
- Commento su singola linea //

Definizione di variabili

- Prima di essere adoperata una variabile deve essere definita

```
var nome;
```

- La definizione stabilisce il nome della variabile, mentre il tipo dipende dall'assegnazione

```
eta = 25;          //intero
nome = "Paolo Rossi"; //stringa
```

- Il tipo di una variabile dipende dall'ultima assegnazione, quindi una variabile può cambiare tipo nel corso del suo ciclo di vita

```
x = 15; //intero
…
x = "b"; //stringa
```

- JavaScript è un linguaggio "weakly typed"

Tipi di dato predefiniti

- Number
- Boolean
- Null
- String
- Date
- Array

Number e Boolean

- Una variabile di tipo Number assume valori numerici interi o decimali

```
var x = 15;     //intero
var y = -6,8; //decimale
```

- Sono definite le operazioni aritmetiche fondamentali ed una serie di funzioni matematiche di base

- Una variabile di tipo Boolean assume i soli valori della logica booleana vero e falso

```
var controllo = true; //valore  logico vero
var consegnato = false; //valore logico falso
```

- Sono definite le operazioni logiche fondamentali (AND, OR e NOT)

Null e String

- Si tratta di un tipo che può assumere un unico valore

```
var a = null;
```

- Serve ad indicare che il contenuto della variabile è non significativo

```
var genere = "f";
var descrizione = null;
```

- Una variabile di tipo String contiene una sequenza arbitraria di caratteri
- Un valore di tipo Stringa è delimitato da apici (' ') o doppi-apici (" ")

```
var nome = "Paolo Rossi";
var vuota = ""; //Stringa vuota

var vuota2 = new String(); //Stringa vuota

var stringa = 'Anche questa è una stringa';

var stringa2 = new String("Altra stringa");
```

Date e Array

- Una variabile di tipo Date rappresenta un istante temporale (data ed ora)

```
var adesso = new Date();
var dataGMA = new Date(2012,10,23);
var dataStringa = new Date("Gen 1 2016");
```

- E' definito l'operatore di sottrazione (−) tra due date che restituisce la differenza con segno espressa in millisecondi
- Un array è un vettore monodimensionale di elementi di tipo arbitrario

```
var v = new Array(); //Vettore vuoto
var v = new Array("Qui","Quo","Qua");
var v = new Array("Lun","Mar","Mer" ,"Gio" ,"Ven" ,"Sab" ,"Dom");
```

- Non è necessario specificare la dimensione

Operatori su stringhe

- L'unica operazione possibile in un'espressione è la concatenazione di stringhe

```
nomeCompleto = titolo + " " + nome + " " + cognome

indirizzo = recapito + ", " + numCivico +  " " + CAP + " - " + citta + " (" +
prov+ ")"
```

Operatori su vettori

- L'operatore definito sui vettori è l'accesso ad un determinato elemento dato l'indice

```
v[3] = 10;
a[i] = b[i]*c[i];
p = v[1];
```

- Il primo elemento di un vettore ha sempre l'indice 0 (come in C/C++ e Java)

Istruzione if

- Una costruzione alternativa prevede la presenza di una seconda istruzione da eseguire nel caso la condizione risulti falsa

```
if(cond)
   instr_then
else
   instr_else
```

```
if(scelta=="NO") {
    // Se la scelta è NO …
}
else
{
    // Altrimenti …
}
```

Istruzione for e while

- Inizializzare a `0` gli `n` elementi del vettore `a`

```
for(var i=0; i<n; i++) a[i]=0;
```

- Copiare gli `n` elementi del vettore `a` nel vettore `b`

```
for(var i=0; i<n; i++) b[i]=a[i];
```

- L'istruzione **`instr`** viene eseguita finché la condizione **`cond`** risulta essere verificata

```
while(cond)
    instr
```

Definizione di funzioni

- In JavaScript è possibile definire una o più funzioni all'interno di un programma

```
function name(arg0, arg1, …, argn-1)
{
    …
}
```

- La funzione definita è identificata da **name** e dipende dagli argomenti $\mathbf{arg_0}$, $\mathbf{arg_1}$, …, $\mathbf{arg_{n-1}}$

Definizione di funzioni: Esempio

- Somma di due numeri

```
function somma(a, b) {
    return a + b;
}
```

- La funzione viene “invocata” all’interno di un’espressione

```
var s = somma(10, 2);
```

Variabili locali e globali

- Si consideri il seguente frammento di codice

```
function f(…) {
    … var x = 1;
    …
}
var x = -1;
…
f(…);
```

- La variabile globale `x` continua a valere -1

Funzioni predefinite

- In JavaScript sono presenti alcune **funzioni predefinite**
  - `isNaN(v)`: verifica se `v` non è un numero
  - `isFinite(v)`: verifica se `v` è finito
  - `parseFloat(str)`: converte `str` in un numero decimale
  - `parseInt(str)`: converte `str` in un numero intero

NOTA: non posso creare classi ma solo usare quelle presenti.

Oggetti

- Un oggetto è un elemento caratterizzato da uno stato rappresentato mediante proprietà e da un insieme di azioni (o metodi) che può eseguire
- Oggetti caratterizzati dagli stessi metodi e dalle stesse proprietà, ma non dallo stesso stato, sono detti della stessa classe
- JavaScript è un linguaggio orientato agli oggetti, tuttavia non possiede il costrutto di classe
- Molti tipi di dato fondamentali sono, in effetti, degli oggetti (String, Date, Array,…)

Proprietà e metodi

- Una proprietà di un oggetto è assimilabile ad una variabile

```
cliente.nome = "Paolo Bianchi";
x = ordine.precentualeIVA;
```

- Un metodo, invece, è simile ad una funzione

```
tot = ordine.calcolaTot();
```

Proprietà e metodi

- Esistono due sintassi alternative per accedere alle proprietà degli oggetti

```
oggetto.proprieta
oggetto["proprieta"]
```

- La seconda è utile quando il nome della proprietà viene determinato durante l'esecuzione del programma

Oggetto di tipo String

- Proprietà
  - `length` lunghezza della stringa
- Metodi
  - `charAt(pos)` carattere alla posizione `pos`
  - `subString(start, end)` sottostringa dalla posizione `start` alla posizione `end`
  - `toUpperCase()/toLowerCase()` converte la stringa in maiuscolo/minuscolo
  - `indexOf(str, pos)` posizione della prima occorrenza della string `str` cercata a partire dalla posizione `pos`

Oggetto di tipo Array

- Proprietà
  - `length` lunghezza del vettore
- Metodi
  - `sort()` ordina gli elementi del vettore
  - `reverse()` inverte l'ordine degli elementi del vettore

Oggetto di tipo Date

- Metodi
  - `getXXX()` restituisce il valore della caratteristica *XXX* della data (es. *getFullYear()*).
  - `setXXX(val)` imposta il valore della caratteristica *XXX* della data (es. *setFullYear(2013,1,1)*);
  - `toString()` restituisce la data come stringa formattata

Oggetto di Math

- Proprietà
  - `E` costante di Eulero
  - `PI` pi greco
- Metodi
  - `abs(val)` valore assoluto
  - `ceil(val)/floor(val)` troncamento
  - `exp(val)` esponenziale
  - `log(val)` logaritmo
  - `pow(base, exp)` elevamento a potenza
  - `sqrt(val)` radice quadrata

Integrazione con i browser web

- La caratteristica principale di JavaScript è di essere "integrabile" all'interno delle pagine web
- In particolare consente di aggiungere una logica procedurale alle pagine rendendole "dinamiche" (ma non è il dinamismo del tipo "interazione" con database)
- A differenza di altre tecnologie, JavaScript funziona completamente sul client
- I campi di impiego tradizionali sono
  - Validazione dell'input dell'utente e controllo dell'interazione
  - Effetti visivi di presentazione

Integrazione con i browser web

- L'integrazione degli script all'interno di una pagina HTML avviene in due modi
  - Associando l'esecuzione di funzioni JavaScript agli **eventi** collegati alla pagina che si intende gestire
  - Accedendo dalle funzioni JavaScript alle proprietà degli oggetti che costituiscono la pagina

Modello ad Oggetti

- Un browser web esporta verso JavaScript un modello ad oggetti della pagina e dell'"ambiente" in cui la pagina è visualizzata
- Una funzione JavaScript adopera tali oggetti invocando i metodi e accedendo alle proprietà
- Il modello ad oggetti, a differenza del linguaggio, non è standard

Oggetto navigator

- L'oggetto `navigator` rappresenta l'istanza del browser in cui lo script è in esecuzione
- Proprietà
  - `appCodeName` codice identificativo del browser
  - `appName` nome del browser
  - `appVersion` numero di versione

Oggetto window

- Questo oggetto rappresenta la finestra in cui il documento corrente viene visualizzato
- Una funzione può accedere alle proprietà della finestra corrente, ma può creare e manipolare nuove finestre (pop-up)
- Proprietà
  - `title` titolo della finestra
  - `statusbar` testo mostrato sulla barra di stato
  - `location` URL del documento visualizzato
  - `outerHeight`, `outerWidth` dimensioni esterne
  - `innerHeight`, `innerWidth` dimensioni interne

Oggetto window

- Modificare il titolo della finestra corrente

```
window.title = “Questo è il nuovo titolo”;
```

- Accedere ad un nuovo documento

```
window.location = “http://www.google.com/”;
```

- Calcolare l’area in pixel della finestra

```
var area = window.innerWidth * window.innerHeight;
```

Oggetto window

- Metodi
  - `open(location, title)` apre una nuova finesta
  - `alert(message)` visualizza il messaggio in una finestra di dialogo (utile per il debug) (Posso renderla modale, cioè bloccare tutto il resto fin quando c'è il popup)
  - `confirm(message)` visualizza il messaggio e richiede una conferma all'utente
  - `moveTo(x, y)` sposta la finestra alle coordinate indicate
  - `resizeTo(width, height)` dimensiona la finestra

Oggetto window

- Visualizzazione di un messaggio

```
window.alert("Attenzione si è verificato un errore");
```

- Visualizzazione del browser in uso

```
window.alert("Sei connesso con " + navigator.appName + " versione " + navigator.version);
```

Oggetto window

- Richiesta conferma all'utente

```
if(confirm("Vuoi proseguire con l'operazione?")) {
    //L'utente ha risposto SI
    …
}
else {
    //L'utente ha risposto NO
    …
}
```

Oggetto history

- Rappresenta la sequenza di pagine visitate dall'utente
- Tale sequenza è rappresentata mediante un vettore
- Metodi
  - `back()` torna alla pagina precedente
  - `forward()` passa alla pagina successiva

Oggetto document

NICE!

- Rappresenta il documento HTML che costituisce la pagina visualizzata
- Non è possibile accedere a tutti gli elementi del documento
- Tuttavia è possibile accedere agli elementi dei moduli (form) ed alle proprietà di visualizzazione

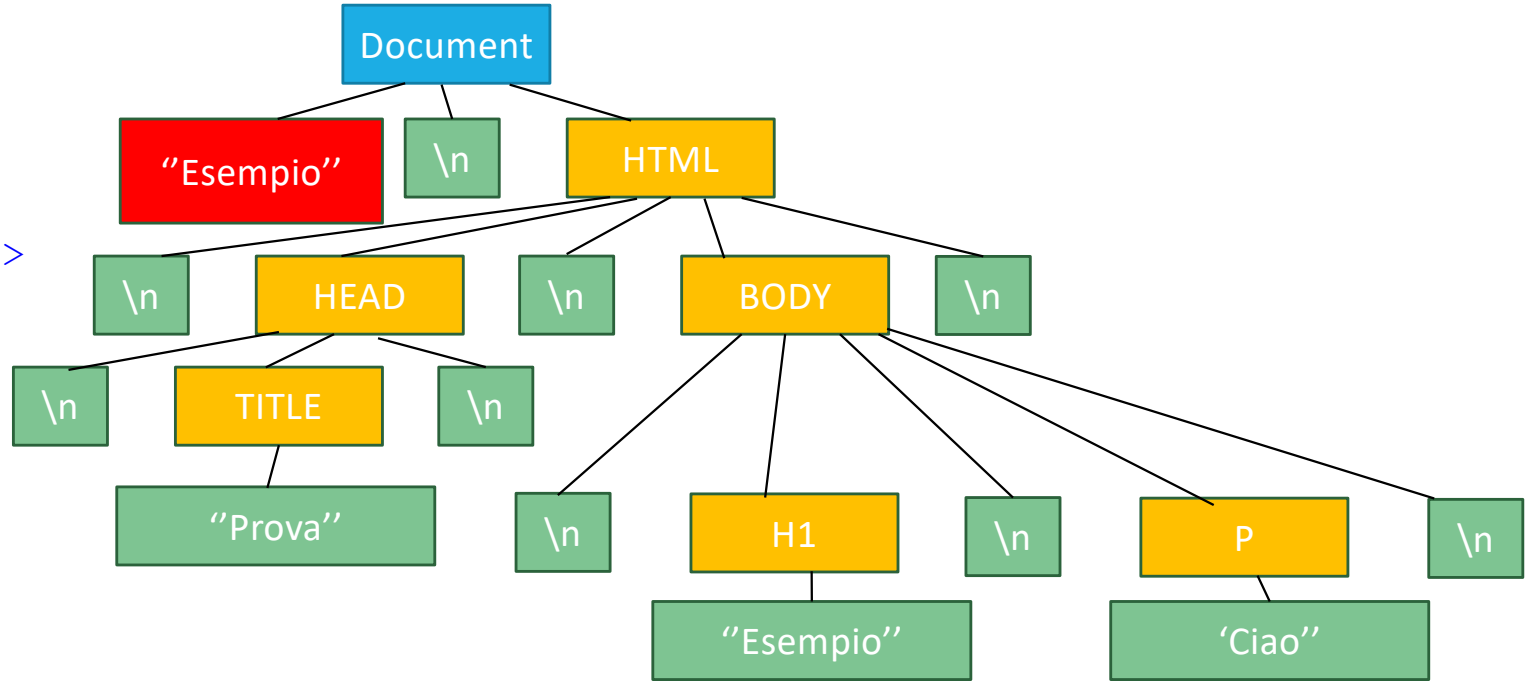
Oggetto document

- Proprietà
  - `bgColor` colore dello sfondo
  - `fgColor` colore del testo
  - `forms` vettore dei moduli presenti nella pagina
  - `title` titolo del documento
  - `URL` indirizzo del documento
- Metodi
  - `write(string)` accoda `string` al documento, serve per la costruzione on-the-fly

Documento HTML = albero

- Nel DOM il documento HTML è rappresentato come un albero:

```
<!-- Esempio -->
<HTML>
    <HEAD>
        <TITLE>Prova</TITLE>
    </HEAD>
    <BODY>
        <H1>Esempio</H1>
        <P> Ciao </P>
        <P> Ciao </P>
    </BODY>
</HTML>
```



Oggetto document

Principali metodi per selezionare elementi:

- `document.getElementById( "titolo" );` -> Restituisce l'elemento (se esiste) il cui attributo ID vale "titolo"
- `document.getElementsByClassName( "classe1" );` -> Restituisce un oggetto NodeList che contiene tutti gli elementi il cui attributo CLASS vale "classe1"
- `document.getElementsByName( "prova" );` -> Restituisce un oggetto NodeList che contiene tutti gli elementi il cui atttributo NAME vale "prova"
- `document.getElementsByTagName( "div" );` -> Restituisce un oggetto NodeList che contiene tutti gli elementi div

Oggetto document

- Supponendo che nel documento HTML sia definito un modulo di nome *modulo*

```
<FORM NAME="modulo"…>
…
</FORM>
```

- Si può accedere a tale oggetto in due diversi modi

```
document.forms["modulo"];
document.modulo;
```

- Ciò è possibile, in generale, per tutti gli elementi del documento con un attributo NAME

Oggetto document

- Dal momento che la proprietà `forms` è di tipo `Array` è possibile accedervi anche tramite l'indice numerico dell'elemento

```
for(var i=0; i<document.forms.length; i++) {
    //Accedi a document.forms[i]
    ... = document.forms[i]; ...
}
```

Oggetto Form

- Un oggetto di questo tipo corrisponde ad un modulo all'interno di una pagina HTML
- Proprietà
  - `action` valore dell'attributo `ACTION`
  - `elements` vettore contenente gli elementi del modulo
  - `length` numero di elementi del modulo
  - `method` valore dell'attributo `METHOD`
  - `target` valore dell'attributo `TARGET`
- Metodi
  - `reset()` azzera il modulo reimpostando i valori di default per i vari elementi
  - `submit()` invia il modulo

Posso cambiare action del mio form a secondo della scelta

Oggetto Form

- Supponendo che l'i-esimo elemento di un modulo `mod` sia denominato `nome_i` è possibile farvi riferimento in 3 modi diversi

```
document.mod.elements[i-1];
document.mod.elements["nome_i"];
document.mod.name_i;
```

- Attenzione l'indice del primo elemento di un vettore è sempre 0 (quindi l'i-esimo elemento ha indice i-1)

Elementi dei moduli

HTML	JavaScript
<INPUT TYPE="text">	Text
<TEXTAREA></TEXTAREA>	Textarea
<SELECT></SELECT>	Select
<INPUT TYPE="checkbox">	Checkbox
<INPUT TYPE="radio">	Radio
<INPUT TYPE="button">	Button

Elementi dei moduli

- Tutti i tipi di elementi possiedono le seguenti proprietà
  - `name` nome dell'elemento
  - `value` valore corrente dell'elemento

(anche `id` e `class` volendo)

- Gli elementi di tipo `Input` possiedono la proprietà `defaultValue` che contiene il valore predefinito del campo (attributo `VALUE` del tag HTML)

Elementi dei moduli

- Gli elementi di tipo `Radio` e `Checkbox` possiedono la proprietà `checked` che indica se l'elemento è stato selezionato
- Gli elementi di tipo `Select` possiedono la proprietà `selectedIndex`, che contiene l'indice dell'elemento selezionato nella lista, e la proprietà `options`, che contiene il vettore delle scelte dell'elenco

Elementi dei moduli

- E’ possibile modificare i valori contenuti negli elementi dei moduli
- Pertanto è possibile utilizzare questi elementi anche per fornire risultati all’utente
- Se un elemento ha scopi esclusivamente di rappresentazione può essere marcato come `READONLY`

Eventi

- Ogni oggetto di un documento HTML "genera" degli **eventi** in risposta alle azioni dell'utente
- Ad esempio, l'evento `click` corrisponde al click del puntatore sull'oggetto
- Per gestire l'interazione con l'utente si associano funzioni JavaScript a particolari eventi

Eventi

- Gli eventi generati da un oggetto dipendono dal tipo di quest'ultimo
- Oggetti `Form`
  - `onSubmit` invio del modulo
  - `onReset` azzeramento del modulo
- Oggetti `Button`
  - `onClick` click del puntatore

Eventi

- Oggetti `Select`, `Text` e `Textarea`
  - `onChange` modifica del contenuto
  - `onFocus` selezione dell'elemento
- Oggetti `Radio` e `Checkbox`
  - `onClick` click del puntatore
  - `onFocus` selezione dell'elemento

JavaScript in documenti HTML

- Il codice JavaScript viene inserito all'interno di una pagina HTML delimitato dal tag

  `<SCRIPT>…</SCRIPT>`

- Per motivi di compatibilità con i vecchi browser il codice è incluso in un commento HTML `<!-- … -->`

Ricorda!

JavaScript in documenti HTML

- Generalmente il codice è incluso all'interno dell'intestazione (<HEAD>) del documento CAMBIA
- Il codice viene eseguito **prima** della visualizzazione del documento.
- In questa sezione si procede con la definizione delle funzioni e delle variabili globali

Esempio

```
<HTML>
    <HEAD>
            <SCRIPT TYPE="text/javascript" LANGUAGE="javascript">
            <!-
            function f(…);
            var v=…;
            …
            //-->
        </SCRIPT>
        …
    </HEAD>
    …
</HTML>
```

JavaScript in documenti HTML

- Eventualmente il codice JavaScript può risiedere in un documento esterno

```
<SCRIPT TYPE="text/javascript" LANGUAGE="javascript" SRC="URL.js">
</SCRIPT>
```

- Ciò è utile per aumentare la modularità e "nascondere" all'utente il codice

Intercettazione eventi

- Per intercettare l'evento *E* di un tag *T* ed associare l'esecuzione di una funzione *f()*

  `<T onE="return f();">`

- Se il risultato della valutazione della funzione è `false` viene interrotta l'esecuzione del commando corrente, ad esempio l'invio di un modulo

Intercettazione eventi

- Ad esempio, la funzione `valida()` verifica se i dati immessi nel modulo `modulo` sono corretti ed eventualmente procede con l'invio di quest'ultimo

```
<FORM NAME="modulo" onSubmit="return valida();" …>
…
</FORM>
```

Validazione modulo di iscrizione: Esempio

- Il modulo viene ridefinito come

```
<FORM NAME="iscrizione" ACTION="" METHOD="POST" onSubmit="return validaForm();">
```

- Nell'intestazione del documento viene definita una funzione `validaForm()`

Validazione modulo di iscrizione: Esempio

```
function validaForm() {
    if(document.iscrizione.nominativo.value==“”)
        {
            alert(“Nominativo obbligatorio. Impossibile procedere.”);
            return false;
        }
    … //Altri controlli
    return true;
}
```