

MODULO 2

PARTE 4

Introduzione alla programmazione client-side (con Javascript)

Goy - a.a. 2012/2013

Programmazione Web

1

Il fil rouge del MODULO 2

-
- | | |
|---|---|
| gli oggetti del DOM... | PARTE 4: Introduzione alla progr. client-side <ul style="list-style-type: none">• Javascript• DOM e gestori di eventi |
| classi, istanze, metodi...
la classe XMLHttpRequest
le API della classe Map | PARTE 5: Programmare sul Web 2.0: tagging, AJAX e Open API <ul style="list-style-type: none">• Un esempio di applicazione interattiva: tagging system• L'interazione asincrona tra client e server: uso di tecnologie AJAX• Utilizzo di open API (google chart, google maps) |
| interfaccia e implementazione,
principi dell'OOP,
costruire ed usare una classe | PARTE 6: Approfondimenti di PHP <ul style="list-style-type: none">• Uso del modello object-oriented in PHP (esempi: galleria fotografica; upload di file)• Applicazioni basate su PHP (e non solo): l'esempio dei CMS per il Web• Note |

Goy - a.a. 2012/2013

Programmazione Web

2

Javascript

- Javascript è un **linguaggio di scripting**, tipicamente utilizzato **client-side**
- Nonostante la somiglianza nel nome, è un linguaggio **completamente distinto da Java**
- Come tutti i linguaggi di scripting, è **interpretato**: il sorgente non deve essere compilato per essere eseguito
- L'interprete di Javascript è generalmente contenuto all'interno del **browser**
- A differenza di HTML, **Javascript è case-sensitive**:
⇒ **pippo** ≠ **Pippo** ≠ **PIPP0** ≠ **pipPo**
- Purtroppo possono esserci **incompatibilità e differenze** tra i **diversi browser** (interpretano versioni un po' diverse di Javascript ⇒ non è scontato che gli script funzionino nello stesso modo su browser diversi: talvolta su alcuni browser non funzionano!)

Javascript: tipi

Javascript è un linguaggio **debolmente tipato**:

- Il **tipo** delle variabili (e dei parametri/argomenti delle funzioni) non viene dichiarato esplicitamente, ma definito implicitamente al primo assegnamento:
totale = 0; → *totale* è di tipo *Number*
- Javascript **converte** automaticamente i tipi durante l'esecuzione (quando possibile)

Tipi principali in Javascript:

- *Number*: interi e decimali (virgola mobile); es: 3, 7.67
- *Boolean*: valori booleani (vero/falso); es: *true*, *false*
- *String*: sequenze di caratteri; es: "pippo", "CV_45ie"

Javascript: variabili e ;

Le **dichiarazioni** delle **variabili globali** non sono obbligatorie, ma si possono fare con la keyword *var* (ma senza tipo!):

```
var totale;
```

Le **dichiarazioni** delle **variabili locali** sono obbligatorie, e devono essere fatte con la keyword *var* (ma senza tipo!):

```
var prezzo_scontato;
```

Le **inizializzazioni** sono facoltative

NB: molti linguaggi permettono di **compattare dichiarazione e inizializzazione in un'unica istruzione**; per es:

```
var totale = 0.0;
```

ATTENZIONE!

Tutte le istruzioni Javascript dovrebbero **terminare con punto-e-virgola ;**

Javascript: operatori e commenti

Aritmetici: `+, -, *, /,`
`++ (i++ → i = i+1), -- (i-- → i = i-1)`

Di confronto: `==, !=, >, >=, <, <=` [\[vedi lucido successivo\]](#)

Booleani: `&&, ||, !` [\[vedi lucido successivo\]](#)

Concatenazione (di stringhe): `+`
`("Buongiorno" + " a tutti" → "Buongiorno a tutti")`

Assegnamento: `= (totale = prezzo+iva),`

I **commenti** sono come in Java:

```
// commento su una riga
/* commento su + righe
  commento su + righe
  commento su + righe */
```

I commenti vengono saltati dall'interprete Javascript (NON vengono interpretati)

Javascript: operatori di confronto

<code>==</code>	uguale (numeri e stringhe)
<code>!=</code>	diverso (numeri e stringhe)

<code>></code>	maggiore
<code>>=</code>	maggiore o uguale
<code><</code>	minore
<code><=</code>	minore o uguale

Javascript: operatori booleani

<code>&&</code> = AND (sia l'uno che l'altro)
<code> </code> = OR (o l'uno o l'altro, o entrambi)
<code>!</code> = NOT (il contrario di)

→

A	B	A && B
V	V	V
V	F	F
F	V	F
F	F	F

A	B	A B
V	V	V
V	F	V
F	V	V
F	F	F

A	!A
V	F
F	V

dove: V = vero (*true*) e F = falso (*false*)

Javascript: <SCRIPT> tag - I

Il codice di un programma Javascript viene incluso in un file HTML per mezzo del tag <SCRIPT>, per es:

```
<BODY>
```

```
...
```

```
<SCRIPT language="JavaScript">
```

```
  prima istruzione;  
  seconda istruzione;  
  terza istruzione;  
  ...
```

l'interprete HTML lo
gira all'interprete
Javascript

```
</SCRIPT>
```

```
...
```

```
</BODY>
```

In particolare (per convenzione)...

Javascript: <SCRIPT> tag - II

- Le **definizioni** delle funzioni vengono generalmente incluse nella sezione <HEAD>... <HEAD>:

```
<HEAD>
```

```
...
```

```
<SCRIPT language="JavaScript">
```

```
  definizione della funzione calcolaPrezzo
```

```
</SCRIPT>
```

```
</HEAD>
```

- I **richiami** alle funzioni (predefinite nel linguaggio o definite dal programmatore) avvengono dove occorre, nel *body* della pagina HTML:

```
<BODY>
```

```
...
```

```
<SCRIPT language="JavaScript">
```

```
  risultato = calcolaPrezzo(..., ...);
```

```
</SCRIPT>
```

```
...
```

```
</BODY>
```

Javascript: esempio di funzione - I

VEDI es-funz1.html

Nel file **es-funz1.html**:

Definizione della funzione:

```
<HEAD>
...
<SCRIPT language="JavaScript">
  function calcolaPrezzo(prezzo_intero, sconto) {
    var risultato = 0;
    risultato = prezzo_intero - (prezzo_intero * sconto) / 100;
    return risultato;
  }
</SCRIPT>
</HEAD>
```

parametri (formali)
NB: in Javascript non dichiaro esplicitamente il tipo

keyword che precede l'espressione (in questo caso una variabile) che deve essere restituita come risultato: l'interprete valuta l'espressione (in questo caso la variabile) e restituisce il valore così ottenuto

keyword che precede la definizione di funzione in Javascript

Goy - a.a. 2012/2013

Programmazione Web

11

Javascript: esempio di funzione - II

Invocazione della funzione:

```
<BODY>
...
<SCRIPT language="JavaScript">
  var prezzo_iniziale = prompt("Qual'è il prezzo di partenza?", "");
  var tasso_sconto = prompt("Qual'è il tasso di sconto da applicare?", "");
  var ris = calcolaPrezzo(prezzo_iniziale, tasso_sconto);
  document.write("<P>Prezzo scontato:" + ris + "</P>");
</SCRIPT>
...
</BODY>
```

Goy - a.a. 2012/2013

Programmazione Web

12

Javascript: esempio di funzione - III

```
var prezzo_iniziale = prompt("Qual'è il prezzo di partenza?", "");
```

prompt → funzione predefinita:

Input = 2 parametri (di tipo string):

- scritta fissa e
- valore predefinito (default) per la casella di testo



Output = ciò che l'utente scrive

Quando viene invocata:

- fa apparire sullo schermo la finestra di dialogo e aspetta
- quando l'utente clicca sul pulsante OK, "cattura" la scritta dell'utente e la restituisce come risultato

NB: il risultato viene assegnato alla variabile **prezzo_iniziale**

Javascript: esempio di funzione - IV

```
var ris = calcolaPrezzo(prezzo_iniziale, tasso_sconto);
```

invocazione della funzione *calcolaPrezzo*:

devo passare alla funzione i *parametri attuali*;

da come sono usati nella definizione della funzione,

so che devono essere di tipo *Number*;

ciò che io voglio è passargli il prezzo iniziale e il tasso di sconto
forniti dall'utente nelle finestre di dialogo;

tali valori sono contenuti nelle variabili **prezzo_iniziale** e
tasso_sconto

Cosa fa l'interprete quando incontra questa istruzione?

- valuta i parametri (cioè guarda nelle scatole e prende i valori)
- applica la funzione **calcolaPrezzo**, cioè esegue le istruzioni contenute nella sua definizione)
- assegna il risultato alla variabile **ris**

Javascript: esempio di funzione - V

```
document.write("<P>Prezzo scontato:" + ris + "</P>");
```

document.write → funzione predefinita:

Input = un parametro (di tipo string);

nell'es. tale parametro è una concatenazione (+) di 3 stringhe:

- "<P>Prezzo scontato:" [fissa]
- **ris** [stringa contenuta nella (valore della) variabile *ris*]
- "</P>" [fissa]

Output = nessuno

Quando viene invocata:

- stampa sulla pagina Web la stringa passata come parametro;
nell'es. stamperà: <P>Prezzo scontato: 127.2</P>

Condizioni booleane

Una **condizione booleana** è un'espressione che ha valore **vero** (*true*) o **falso** (*false*)

Le condizioni booleane sono espressioni composte da:

- costanti
- variabili
- operatori di confronto
- operatori logici

Per esempio:

$3 > 5 \rightarrow \text{false}$

$3 < 5 \rightarrow \text{true}$

$x = y$ [dato $x=33.3$ e $y=20.7$] $\rightarrow \text{false}$

$x = y$ [dato $x=\text{"Pippo"}$ e $y=\text{"PIPPO"}$] $\rightarrow \text{false}$

$z \ \&\& \ (x \leq y)$ [dato $z=\text{true}$, $x=10$, $y=10$] $\rightarrow \text{true}$

$!z \ || \ (x \neq y)$ [dato $z=\text{true}$, $x=10$, $y=12$] $\rightarrow \text{true}$

Condizionali - *if ... else* - I

Avendo a disposizione espressioni che esprimono condizioni booleane, possiamo utilizzare il costrutto ***if - else*** per esprimere un'**azione condizionale**, cioè per controllare una condizione e agire di conseguenza:

```
if (condizione1) {  
    sequenza_di_azioni_1  
}  
else if (condizione2) {  
    sequenza_di_azioni_2  
}  
...  
else {  
    sequenza_di_azioni_n  
}
```

se la *condizione1* è vera esegui
la *sequenza_di_azioni_1*
altrimenti
se la *condizione2* è vera esegui
la *sequenza_di_azioni_2*
...
altrimenti esegui
la *sequenza_di_azioni_n*

Condizionali - *if ... else* - II

VEDI [es-funz2.html](#)

Per es., supponiamo ora di voler realizzare la prima parte della nostra funzione *calcolaPrezzo* e cioè quella che controlla il valore del parametro *prezzo_intero* e se è minore di 10 azzera il tasso di sconto; anzi, complichiamo un po' di più la faccenda e diciamo anche che se il prezzo iniziale è minore di 100, lo sconto non può superare il 20%; aggiungiamo, nella definizione della funzione (vedi file [es-funz2.html](#)):

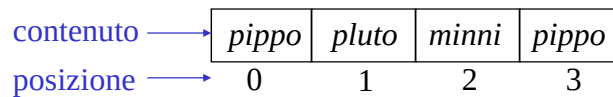
```
if (prezzo_intero<10) {  
    sconto = 0;  
}  
else if ((prezzo_intero<100) && (sconto>20)) {  
    sconto = 20;  
}  
else {  
}
```

Liste - I

Una **lista** è una **sequenza ordinata di elementi**; per es: un elenco di link in una pagina Web, l'elenco degli iscritti ad un esame, ecc.

Alcune operazioni che posso voler fare su una lista:

- sapere quant'è lunga (da quanti elementi è composta)
- stampare (visualizzare) tutti gli elementi
- aggiungere un elemento
- cancellare un elemento
- ...



Liste - II

Per rappresentare una **lista** in Javascript possiamo usare un **array**:

Per **creare** un array (per es. per inizializzare una variabile di tipo array "mettendo nella scatola" un array vuoto):

```
var lista = new Array();
```

Per **scrivere** un contenuto in posizione *i*:

```
lista[i] = "pippo";
```

Per **leggere** il contenuto in posizione *i*:

```
var elem = lista[i];
```

NB: gli array iniziano a contare le posizioni da 0!!! Quindi:

Per scrivere il primo elemento dell'array (cioè l'el. in pos. 0):

```
lista[0] = "pippo";
```

Per leggere il primo elemento dell'array (cioè l'el. in pos. 0):

```
var elem = lista[0];
```

Liste - III

Per sapere quanti elementi ci sono in un array:

```
var lunghezza = lista.length;
```

Per (sovra)scrivere l'ultimo elemento dell'array:

```
lista[lunghezza-1] = "topolino";
```

Per leggere l'ultimo elemento dell'array:

```
var elem = lista[lunghezza-1];
```

NB: gli array, quando scrivo un elemento, non... "fanno spazio"! Quindi:

<i>pip</i>	<i>pl</i>	<i>min</i>	<i>pip</i>
0	1	2	3

```
lista[3] = "eta beta";
```

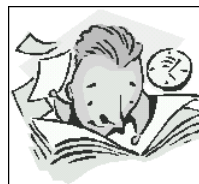
<i>pip</i>	<i>pl</i>	<i>min</i>	<i>eta beta</i>
0	1	2	3

Cicli - I

Supponiamo di avere una **lista** di link (magari letti da un database...) e di volerli visualizzare su una pagina Web

In pratica dobbiamo scorrere la lista e, elemento per elemento, leggerlo e visualizzarlo

Questa operazione è un caso particolare di un'esigenza molto comune: quella di **scorrere** una lista, **prendendo in considerazione gli elementi, uno per volta** (immaginate un prof. che, in classe, scorre il registro col dito, valutando nome per nome i suoi alunni per decidere chi interrogare)



Cicli - II

Per **scorrere una lista**, dobbiamo implementare un **ciclo**:

- parti dal primo elemento della lista
- per ogni elemento, controlla se è quello che cerchi
- se lo è, hai finito (con successo), altrimenti prosegui
- se sei arrivato al fondo e non hai trovato ciò che cercavi, hai finito (senza successo)

I cicli sono generalmente basati sul concetto di **indice** (*i*):
l'indice scorre lungo la lista indicando, via via, posizioni successive

Per fare questo in Javascript ci sono varie opzioni;
ne vediamo due:

- Il costrutto ***for*** e il costrutto ***while***

Cicli – *for* - I

```
for (inizio; test; incremento) {  
    sequenza_di_azioni  
}
```

esegui la *sequenza_di_azioni* a partire da *inizio*,
finché *test* è vero, avanzando ad ogni passo di
quanto è indicato da *incremento*; cioè:

- ***inizio*** = la posizione iniziale dell'indice
- ***test*** = una condizione che, finché è vera (ha valore *true*)
fa sì che il ciclo prosegua; quando non è più vera (ha valore
false), provoca l'uscita dal ciclo
- ***incremento*** = incremento dell'indice ad ogni ciclo

Cicli – for - II

Supponiamo di avere una lista di link (magari letti da un database...) e di volerli visualizzare su una pagina Web

Nel file **es-for.html**:

```
<SCRIPT language="JavaScript">
var urls = new Array();
urls[0] = "www.di.unito.it";
urls[1] = "www.scidecomcult.unito.it";
urls[2] = "www.celi.it";
for (var i=0; i<urls.length; i++) {
    document.write("<P><A HREF='http://' + urls[i] + '>"
        + urls[i] + "</A></P>");
}
</SCRIPT>
```

Cicli – for - III

urls =

www.di.unito.it	www.scidecomcult.unito.it	www.celi.it
-----------------	---------------------------	-------------

for (**var i=0;** **i<urls.length;** **i++**)

inizio = la posizione iniziale dell'indice;
dichiaro una variabile e la inizializzo alla posizione da cui
voglio partire (generalmente la prima...)

test = una condizione che, finché è vera (ha valore *true*) fa sì
che il ciclo prosegua; quando non è più vera (ha valore *false*),
provoca l'uscita dal ciclo; in questo caso, finché l'indice *i* è
minore della lunghezza della lista

incremento = incremento dell'indice ad ogni ciclo
i++ è una scorciatoia per **i=i+1**: l'interprete valuta l'espressione
a destra dell'uguale e assegna il valore ottenuto alla variabile *i*
(l'effetto è che se *i* valeva 2, dopo questa operazione varrà 3)

Cicli – for - IV

Quando l'interprete **incontra il ciclo** (l'istruzione *for*) **per la prima volta**:

- inizializza l'indice (**var i=0;**)
- valuta il test (**i<urls.length**) → **0 < 3** → *true*
- esegue le istruzioni [\[vedi dopo\]](#)
- incrementa l'indice (**i++**)
- ripete il ciclo

Quando l'interprete **ripete il ciclo** (incontra l'istruzione *for* per la seconda, terza, ... volta):

- valuta il test (**i<urls.length**) → **1 < 3** → *true*
- esegue le istruzioni [\[vedi dopo\]](#)
- incrementa l'indice (**i++**)
- ripete il ciclo

Finché... (**uscita dal ciclo**)

- valuta il test (**i<urls.length**) → **3 < 3** → *false*
- si ferma (prosegue con l'istruzione successiva al *for*)

Cicli – for - V

Vediamo nel dettaglio le istruzioni all'interno del ciclo dell'es.

```
document.write("<P><A HREF='http://' + urls[i] + "'>"  
              + urls[i] + "</A></P>");
```

↓
scrivi sulla pagina Web la stringa composta concatenando:

- "<P><A HREF='http://'" _
- `urls[i]` _
- "'>" _
- `urls[i]` _
- "</P>" _

NB `urls[i]` indica l'elemento in posizione *i*-esima nell'array `urls`:

- all'inizio *i*=0, `url[i]=url[0]`="www.di.unito.it"
- al secondo giro *i*=1, `url[i]=url[1]`="www.scidecomcult.unito.it"
- al terzo giro *i*=2, `url[i]=url[2]`="www.celi.it")
- al quarto giro *i*=3, quindi il ciclo si ferma

Cicli – while - I

```
while (condizione) {
    sequenza_di_azioni
}
```

finché la *condizione* è vera
esegui la *sequenza_di_azioni*

Riscriviamo il ciclo precedente utilizzando il costrutto *while*

Nel file **es-while1.html**:

```
<SCRIPT language="JavaScript">
var urls = new Array();
urls[0] = "www.di.unito.it";
urls[1] = "www.scidecomcult.unito.it";
urls[2] = "www.celi.it";
var i=0;
while (i<urls.length) {
    document.write("<A HREF='http://"+ urls[i] + "'>"
        + urls[i] + "</A><BR>");
    i++;
}
</SCRIPT>
```

Cicli – while - II

Se usiamo un *while* per scorrere una lista, **l'indice dobbiamo inizializzarlo** fuori (prima) del ciclo: **var i=0;**

Quando l'interprete **incontra il ciclo** (l'istruzione *while*):

- valuta il test (**i<urls.length**) → **0 < 3** → *true*
- esegue le istruzioni
- **NB:** l'incremento dell'indice (**i++**), nel caso del *while*, deve essere l'ultima istruzione del ciclo: non dimenticatevelo!!!
- ripete il ciclo

Finché... (**uscita dal ciclo**)

- valuta il test (**i<urls.length**) → **3 < 3** → *false*
- si ferma (prosegue con l'istruzione successiva al *while*)

Attenzione! Se la condizione di entrata nel ciclo è sempre vera, il ciclo **non termina** (cioè prosegue l'esecuzione all'infinito): in gergo si dice che "**va in loop**" (questo succede, per es, se vi dimenticate di incrementare l'indice...)

Cicli – *while* - III

Nota:

Dagli esempi precedenti vediamo che i cicli *for* e *while* si **equivalgono**; per es, quando doverte scorrere una lista, potete usare l'uno o l'altro, secondo le vostre preferenze

Per scorrere una lista è generalmente più semplice da usare il *for*, mentre il *while* è, potremmo dire, più versatile e può essere usato per scopi diversi dalla gestione delle liste

Per es, supponiamo di voler richiedere la password ad un utente e di voler insistere finché non è corretta

Cicli – *while* - IV

VEDI es-while2.html

Nel file *es-while2.html*:

```
<SCRIPT language="JavaScript">
  var pswd = "geronimo";
  var risposta = "";
  while (risposta != pswd) {
    risposta = prompt("Parola d'ordine?", "");
  }
  /* questa istruzione viene eseguita solo
     quando l'interprete è uscito dal ciclo (cioè
     quando l'utente ha scritto la parola corretta) */
  document.write("<P>Benvenuto!</P>");
</SCRIPT>
```

In questo caso non abbiamo nessuna lista: la condizione di entrata nel ciclo (*risposta != pswd*) è vera finché la parola scritta dall'utente (*risposta*) è diversa dalla password (*pswd*), impostata a "geronimo"

Proposte di esercizi... – I



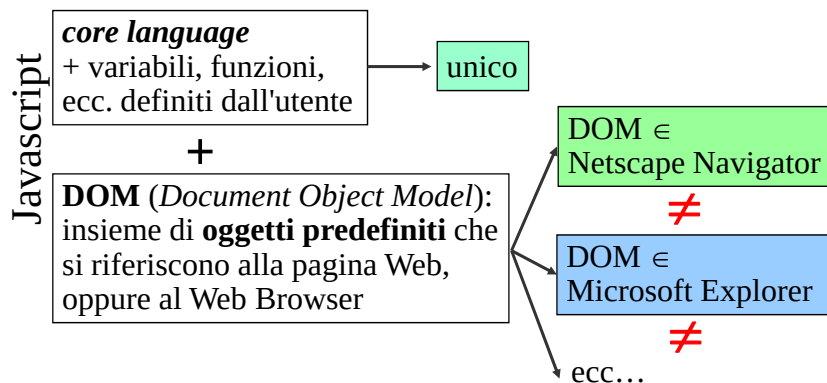
- Modificate il ciclo *while* precedente facendo in modo che consenta all'utente al massimo *n* tentativi (es. 3 o 5...)

Potete, per es, definire una nuova variabile, **tentativi**, che viene incrementata di 1 ad ogni tentativo fallito dell'utente... Quando la variabile **tentativi** raggiunge *n*, il ciclo deve fermarsi

NB Bisognerebbe anche dare all'utente un messaggio diverso: "Benvenuto!" nel caso in cui inserisca la parola d'ordine corretta, "Ingresso non autorizzato" (o quel che preferite) se supera *n* tentativi... (suggerimento: testate il valore della variabile **risposta** dopo il *while*...)

[vedi [sol_es_1.html](#)]

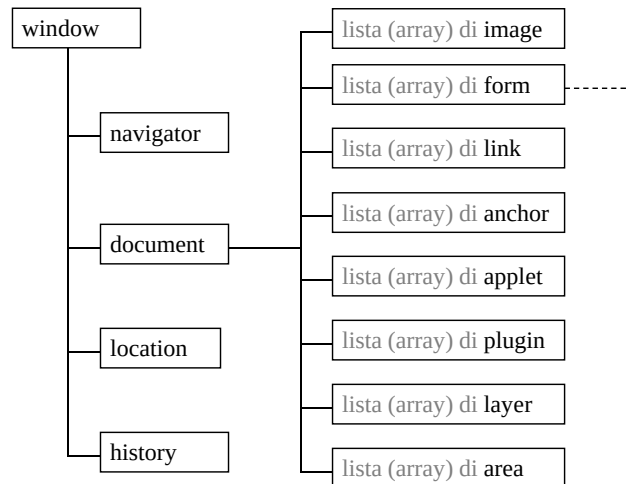
Javascript: DOM - I



NB: Il DOM è **definito dal browser**! E la definizione è fatta **separatamente** da per Netscape, Explorer, Mozilla, ecc.!
⇒ possono nascere incompatibilità...

Javascript: DOM - II

Il DOM è organizzato secondo una **gerarchia**:



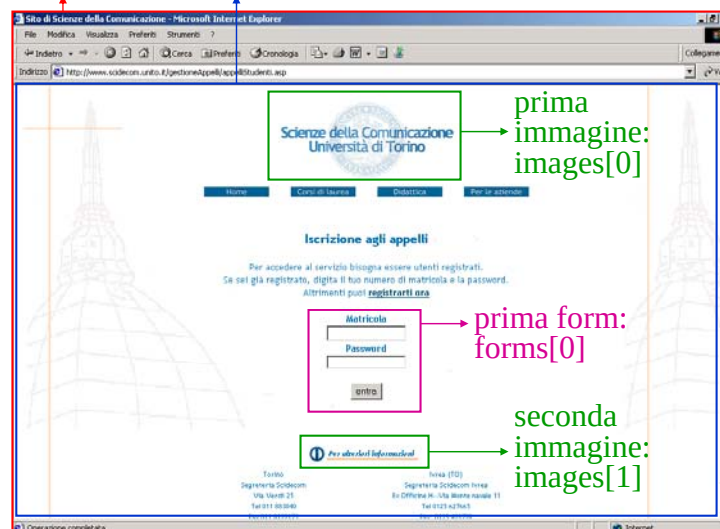
Goy - a.a. 2012/2013

Programmazione Web

35

Javascript: DOM - III

window **document**



Goy - a.a. 2012/2013

Programmazione Web

36

Javascript: DOM - IV

window = la finestra corrente del browser (vedremo degli es.)

FIGLI DI **WINDOW**:

navigator = il browser (in quanto applicazione)

Per es. per sapere quale browser si sta utilizzando (Explorer/ Netscape)

document = il contenuto della finestra (vedremo degli es.)

location = informazioni sull'URL corrente

Per es. per caricare nella finestra un URL differente

history = elenco degli URL visitati di recente

Per es. per tornare alla pagina Web precedente

FIGLI DI **DOCUMENT**:

lista (array) di image = lista delle immagini contenute nella pagina (vedremo degli es.)

lista (array) di form = lista dei moduli (*form*) contenute nella pagina (vedremo degli es.)

Javascript: oggetti (proprietà e funzioni) - I

Oggetto = collezione di

- **proprietà** (variabili) - **NB:** sono a loro volta oggetti...!
- **funzioni** (metodi, operazioni)

Per **accedere alle proprietà** di un oggetto:

`window.status = 'hello!';`^[*]

nome oggetto nome proprietà

⊙ *dot notation*
(notazione a punto)

Per **invocare le funzioni** di un oggetto:

`window.print();` ^{funzione}

`window.document.write("<P>Ciao!</P>");`

nome oggetto

funzione

Per **accedere agli oggetti contenuti in una lista** (array):

`window.document.images[0].src = 'sole.gif';`

la proprietà **src** della prima immagine della pagina (dettagli + avanti...)

[*] Le versioni più recenti dei browser non consentono più di sovrascrivere i messaggi sulla status bar con Javascript!

Javascript: oggetti (proprietà e funzioni) - II

NB

Quando vedete l'invocazione di una funzione apparentemente senza alcun oggetto, per es:

```
prompt("Come ti chiami?", "boh");
```

si tratta in realtà di una funzione dell'oggetto *window*:

```
window.prompt("Come ti chiami?", "boh");
```

Quando vedete un riferimento all'oggetto *document* **non** preceduto da un riferimento all'oggetto *window*, per es:

```
document.write("<P>Ciao!</P>");
```

si tratta in realtà di:

```
window.document.write("<P>Ciao!</P>");
```

Cioè il riferimento all'oggetto *window* è **implicito**

Javascript: eventi - I

I programmi Javascript sono tipicamente "guidati dagli eventi" (**event-driven**):

- **Eventi** = azioni dell'utente sulla pagina Web: ogni volta che l'utente scrive qualcosa in una casella, preme un pulsante, ridimensiona una finestra ecc... genera un "evento"
⇒ un programma Javascript deve contenere un **gestore di eventi** (**event handler**), che sia in grado di ricevere e interpretare le azioni dell'utente (eventi)
- Il **DOM** fornisce una serie di **gestori di eventi predefiniti**: l'accadere di un evento nella pagina Web mette automaticamente in azione il corrispondente gestore di eventi

Javascript: eventi - II

Per es:

`<A HREF="#" onClick = "window.print()" >`

↓ tag HTML

evento (Click) ↓ innesca il corrispondente gestore

codice Javascript che viene eseguito dal gestore innescato

- 1) "#" → resta nella pagina in cui si trova (salta al Top)
- 2) URL → va alla pagina indicata (carica la nuova pagina)
- 3) Se aggiungiamo:


```
<A HREF="#"
onClick="window.print(); return false;">
```

 il browser resta nella pagina corrente, senza saltare al Top!

Javascript: eventi - III

NB: come abbiamo visto nell'es. precedente, un'istruzione Javascript può essere inserita all'interno di un tag HTML, (anziché essere racchiusa nei tag `<SCRIPT>...</SCRIPT>`)

In questi casi, **il gestore di evento invocato farà riferimento al tag in cui si trova l'istruzione**. Per es:

```
<A HREF="#" onClick="window.print()" >
→ quando l'utente fa click (onClick) sul link (<A...>)

<FORM NAME="modulo" onSubmit="alert('Ciao!');" >
→ quando l'utente invia (onSubmit) il modulo (<FORM...>)

<INPUT TYPE="text" NAME="login" onFocus=...;">
→ quando l'utente porta il cursore (onFocus) nel campo di testo (<INPUT TYPE="text"...>)

<BODY onLoad="jump()" >
→ quando l'utente carica (onLoad) la pagina (<BODY...>)
```

Javascript: accesso agli oggetti nella pagina

Lista dei **moduli** (<FORM...>) contenuti in una pagina:

```
window.document.forms[i]
```

Lista delle **immagini** (<IMG...>) contenute in una pagina:

```
window.document.images[i]
```

Ma possiamo utilizzare l'**attributo NAME**:

```
<FORM NAME="modulo_iscriz">  
<INPUT TYPE="text" NAME="matricola">  
→ window.document.modulo_iscriz.matricola=...  
  
<IMG SRC="pippo.gif" NAME="pippo">  
→ window.document.pippo.src=...
```

NB: non usate spazi bianchi nei nomi interni (valori degli attributi NAME e ID)!

Oppure l'**attributo ID**:

```
<INPUT TYPE="text" ID="matricola">  
→ window.document.getElementById("matricola")=...  
  
<IMG SRC="pippo.gif" ID="pippo">  
→ img=window.document.getElementById("pippo");  
img.src=...
```

Javascript: interazione con form - I

Con Javascript è possibile **interagire con i moduli**, cioè:

- intercettare gli eventi che "accadono" nei campi del modulo
- modificare i (scrivere nei) campi del modulo

Vedremo i seguenti **esempi**:

- intercettare il passaggio del mouse su un link e di conseguenza scrivere in un campo di testo di un form
- intercettare il click del mouse su un link e di conseguenza (de)selezionare tutte le checkbox di un form
- impostare dinamicamente le voci di un menu (a), intercettare il click del mouse sul pulsante di invio di un form e di conseguenza mostrare un alert (b)



NB usateli in modo... creativo!!!

Javascript: interazione con form - II

Per es:

```
<FORM NAME="modulo">
<INPUT TYPE="text" NAME="come_va" VALUE="Come va?">
</FORM>
<A HREF="#" onMouseOver="
  "window.document.modulo.come_va.value = 'Tutto ok';" >
  Bene!</A>
<A HREF="#" onMouseOver=
  "window.document.modulo.come_va.value = 'Lasciamo
  perdere';">No comment...</A>
```

tag HTML di riferim. (link)
 evento (*MouseOver*)
 ⇒ innesca il corrispondente gestore
 codice Javascript che viene eseguito dal gestore
 innescato: assegna la stringa 'Tutto ok' alla
 proprietà *value* della casella di testo *come_va*,
 del form *modulo*
 NB riferimento agli oggetti fatto attraverso
 l'attributo NAME!

Javascript: interazione con form - III

```
function checkAll(lungh) {
  for (var i=0; i<lungh; i++) {
    document.modulo.elements[i].checked=true;
  }
}
```

elements[] è un array predefinito che contiene tutti gli elementi di
 un form (qui assumiamo che nel form ci siano solo checkbox)

NB riferimento agli oggetti fatto attraverso l'attributo ID!

```
<FORM NAME="modulo">
  Opz 1: <INPUT TYPE="checkbox" NAME="box[]" VALUE="v1"><BR>
  Opz 2: <INPUT TYPE="checkbox" NAME="box[]" VALUE="v2"><BR>
  Opz 3: <INPUT TYPE="checkbox" NAME="box[]" VALUE="v3"> ...
</FORM>
...
```

<A HREF="#" → tag HTML di riferim.

onClick="checkAll(3);" >

Seleziona tutte le opzioni

evento (*Click*)

⇒ innesca il corrisp. gestore

codice Javascript eseguito
 dal gestore innescato:
 richiama la funzione *checkAll*,
 passando come parametro 3

Javascript: interazione con form - IV

```

<FORM NAME="modulo" onSubmit="complimenti();"
return false;">
  <SELECT NAME="menu">
    <OPTION VALUE="mozz">mozzarella</OPTION>
    <OPTION VALUE="toma">toma</OPTION>
    <OPTION VALUE="cast">castelmagno</OPTION>
  </SELECT>
  <INPUT TYPE="Submit" VALUE="OK">
</FORM>
...
<SCRIPT language="JavaScript">
  var regione = prompt("Sei valdostano?", "no");
  if ((regione == "si") || (regione == "sì")) {
    window.document.modulo.menu.options[1].value="font";
    window.document.modulo.menu.options[1].text="fontina";
  }
</SCRIPT>

```

vedi prossimo lucido...

(b) al verificarsi dell'evento (Submit), viene invocata la funzione *complimenti()*

(a) le opzioni del menu stanno in un array: all'opzione in pos. 1 (cioè alla seconda opzione) assegno un nuovo valore (VALUE="font") e modifico il testo (visibile all'utente)

Javascript: interazione con form - V

return false nel tag `<FORM...>`: il click su un pulsante *Submit* fa partire la HTTPrequest, provocando un ri-caricamento della pagina, effetto talvolta indesiderato (spesso vanifica gli effetti degli script client-side)! L'istruzione **return false** fa sì che ciò **non** accada

NB: in alternativa a `options[1]` potevamo usare l'ID, per es:

```

<OPTION ID="pt" VALUE="toma">toma</OPTION>
window.document.getElementById("pt").value="font";
(attenzione che getElementById() è una funzione di document!)

```

```

function complimenti() {
  if (window.document.modulo.menu.selectedIndex == 2) {
    alert("Bravo!");
  }
}

```

selectedIndex == 2 → terza opzione del menu (cioè se seleziono *castelmagno*)...

Proposte di esercizi... – II



- Aggiungete alla pagina, accanto al link "Seleziona tutte le opzioni", un link che **deseleziona tutte le opzioni**
Potete, per es, definire una nuova funzione, **uncheckAll(lungh)**, che assegna **checked=false** a tutte le checkbox e poi invocarla (**onClick**) in un nuovo link...

Proposte di esercizi... – III



- Scrivete uno script javascript che **controlla se un certo campo di testo di un form è vuoto** e se lo è fa comparire un avviso (pop-up)
Suggerimenti:
 - definite una funzione **controlla(campo)** che controlla se il campo è vuoto e se lo è fa comparire un avviso
 - per l'avviso potete usare la funzione predefinita **alert("...")**
 - per identificare il campo, utilizzate la funzione **getElementById()**
 - usate il gestore di eventi **onSubmit** nel tag FORM, congiuntamente all'invocazione della vostra funzione (es: **onSubmit="controlla('email'); return false;"**)
[vedi [sol_es_2.html](#)]

Interazione tra ACTION (PHP) e onSubmit (Javascript)

```
<FORM METHOD="POST" ACTION="identif1.php"
onSubmit = "alert('Benvenuto!'); return false;">
  Login: <INPUT TYPE="TEXT" NAME="login">
  Password: <INPUT TYPE="PASSWORD" NAME="password">
  <INPUT TYPE="Submit" VALUE="OK">
</FORM>
```

Al click sul pulsante **Submit** cosa viene eseguito?

- l'invio di *HTTP request* per *identif1.php* (**ACTION**), oppure
- l'*alert* (**onSubmit**)?

Viene eseguito prima l'*alert* (**onSubmit**) e poi l'invio di *HTTP request* per *identif1.php* (**ACTION**)

NB: NON funziona se si aggiunge **return false;**

⇒ usate **return false;** per bloccare l'invio... se volete bloccarlo!

Javascript: ridirezionamento - I

VEDI es-jump1.html

Un esempio utile: **ridirezionare** l'utente ad un altro URL:

```
</HEAD>
...
<SCRIPT language="JavaScript">
  function jump(){
    window.location.href="http://www.scidecomcult.unito.it";
  }
</SCRIPT>
<BODY onload = "jump()">
```

definiamo una funzione che
assegna alla proprietà href
dell'oggetto location una stringa
contenente l'URL desiderata

evento (*Load*)
⇒ innesca il
 corrisp. gestore

tag HTML di riferim. (body)

codice Javascript eseguito
dal gestore innescato:
invoca la funzione *jump()*

Javascript: ridirezionamento - II

... con timeout:

```
<HTML>
<HEAD>
  <SCRIPT language="JavaScript">
    function jump(){
      window.setTimeout("window.location.href=
        'http://www.scidecomcult.unito.it';", 5*1000);
    }
  </SCRIPT>
</HEAD>
<BODY onLoad = "jump()">
</BODY>
</HTML>
```

la funzione (dell'oggetto *window*) *setTimeout(...,...)* ha 2 argomenti:

- 1) l'azione da eseguire (asigna alla proprietà *href* dell'oggetto *location* l'URL desiderata)
- 2) il tempo (in millisecondi) passato il quale l'azione viene eseguita

Javascript: image swap

Un esempio molto comune di uso di **onMouseOver** (che cattura il passaggio del mouse su un link) è il cosiddetto **image swap** (o *roll-over*) = cambiare l'aspetto di un'immagine (per es. un pulsante) quando il cursore del mouse ci passa sopra; per es:

```
<A HREF="#">
  onMouseOver = "document.il_sole.src='soleRosso.gif';"
  onMouseOut = "document.il_sole.src='sole.gif';" >
  <IMG SRC="sole.gif" NAME="il_sole" >
</A>
```

tag HTML di riferim. (link)

evento (*MouseOver/Out*)
⇒ innesca il corrisp. gestore

codice Javascript che viene eseguito dal gestore innescato: assegna la stringa *'sole.gif'/'soleRosso.gif'* (cioè il nome del file che contiene l'immagine desiderata) alla proprietà *src* dell'immagine *il_sole*

Javascript: immagini

Proviamo, per **esempio**, a fare un menu (link) che mostra diverse immagini:

```
...
<IMG SRC="coltellazzo.jpg" name="diapo"
      height="400" width="400">
...
<A HREF="#"
  onClick="document.diapo.src='coltellazzo.jpg';
  return false;">Torre Coltellazzo</A>
<A HREF="#"
  onClick="document.diapo.src='calaChia.jpg';
  return false;">Chia: spiaggia 1</A>
<A HREF="#"
  onClick="document.diapo.src='colonna.jpg';
  return false;">Nora: rovine romane</A>
<A HREF="#"
  onClick="document.diapo.src='acquaChia.jpg';
  return false;">Chia: spiaggia 2</A>
...
```

Javascript: nuove finestre - I

`window.open(URL, nome, proprietà);` apre una **nuova finestra** (o una nuova scheda) argomento opzionale, si può omettere (vedi prossimo lucido...)

Per es:

```
<A HREF="#" onClick =
  "window.open('http://www.di.unito.it', 'pippo' );
  return false;">nuova finestra!</A>
```

URL = indirizzo della pagina da caricare nella nuova finestra/scheda

nome = identificatore della finestra ⇒ se faccio...

```
<A HREF="#" onClick =
  "window.open('http://www.scidecom.unito.it', 'pippo' );
  return false;">un'altra finestra?</A>
```

... la nuova pagina Web (identificata dall'URL) verrà caricata nella stessa finestra/scheda!

Javascript: nuove finestre - II

Il terzo parametro del metodo *open* è la lista delle **proprietà della nuova finestra** (è **opzionale** e se omesso, la nuova finestra avrà le stesse proprietà di quella corrente); per es:

```
<A HREF="#" onClick =  
  "window.open('http://www.di.unito.it', 'pippo',  
  'scrollbars=yes,resizable=yes,width=730,height=600,  
  status=no,location=no,toolbar=no,menubar=no');  
  return false;">finestra personalizzata...</A>
```

NB se stabilisco delle proprietà, verrà sicuramente aperta una nuova finestra (con quelle proprietà) e NON una scheda

- presenza delle barre di scorrimento
- possibilità di ridimensionare la finestra
- larghezza (in pixel)
- altezza (in pixel)
- presenza della barra di stato (*status*)
- presenza della barra degli indirizzi (*location*)
- presenza della barra dei pulsanti (*toolbar*)
- presenza della barra del menu

Javascript: nuove finestre - III

VEDI es-nuovaFin.html
info.html

Esempio

es-nuovaFin.html

```
...  
<A HREF="#"  
  onClick="window.open('info.html','info_box',  
    'width=200,height=120,  
    status=no,location=no,toolbar=no,menubar=no');  
  info_box.focus();">ulteriori informazioni</A>
```

↓
sposta la finestra in primo piano

info.html

```
...  
<A HREF="#" onClick = "window.close()">chiudi</A>
```

↓
chiude la finestra