

Il linguaggio Javascript

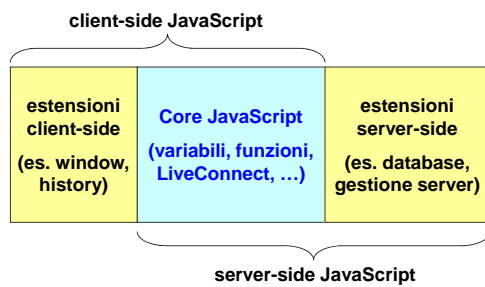
Antonio Lioy
< lioy@polito.it >

Politecnico di Torino
Dip. Automatica e Informatica

Javascript

- linguaggio di script pensato per il web (ma ora adottato anche in altri ambienti, es. Adobe PDF)
- eseguibile lato client o lato server
- introdotto in pagine HTML tramite:
 - il tag <script> nell'head o nel body
 - istruzioni specificate per un event-handler
- eseguito nel momento in cui il browser, leggendo HTML, incontra il codice Javascript (sospensione temporanea dell'azione di interprete HTML)

JavaScript core ed estensioni

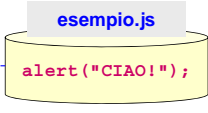


Il tag <script>

- due parametri principali:
 - type="text/javascript"
 - src="URI" (lo script è presente alla URI indicata)
- esempi:

```
<script type="text/javascript">
alert ("CIAO!");
</script>
```

```
<script
  type="text/javascript"
  src="esempio.js">
</script>
```



Sintassi

- Javascript è case-sensitive
- ogni istruzione è terminata da
 - fine riga (vincolo: una sola istruzione per riga)
 - punto-e-virgola (possibili più istruzioni per riga)

```
/* istruzioni terminate da fine riga */
a = 3 + b
c = 2 * a

/* istruzioni terminate con ; */
a = 3 + b ; c = 2 * a;

/* errore! */
a = 3
+ b;
```

Tipi di dati e corrispondenti valori base

- numeri decimali (es. 14, -7, 3.14, 10.7e-4), ottali (es. 016) ed esadecimali (es. 0xE)
- valori Booleani (true false)
- stringhe di caratteri, racchiuse tra apici doppi o singoli (es. "ciao mamma" 'ciao babbo')
- oggetti
- il valore speciale null (variabile non inizializzata)
- il valore speciale undefined (variabile non definita)
- il valore speciale NaN (Not A Number)

Commenti

- stile C++: da "//" sino a fine riga
- stile C: tutto il testo compreso tra "/*" e "*/"
- esempi:

```

<script type="text/javascript">
// commento di una sola riga in stile C++
/*
commento che occupa quattro righe
in stile C
*/
</script>
```

Commenti HTML e JavaScript

- JS riconosce il commento HTML <!-- come commento single-line
- JS non riconosce la sequenza di chiusura del commento HTML (è quindi necessario premettere il commento JS //)
- usato in passato per nascondere JS ai browser che non lo supportano (ed ignorano il tag <script> perché sconosciuto) – oggi meno necessario

```

<script type="text/javascript">
<!--
    document.writeln("<p>Ciao!></p>")
//-->
</script>
```

Variabili

- identificate tramite il loro nome:
 - deve iniziare con un carattere alfabetico, \$ o _
 - può poi contenere caratteri alfanumerici, \$ e _
 - es. costo, \$1, studente_12345, _2009q1
- non sono tipate (al contrario di Java, C, ...) ma:
 - prendono un tipo al momento della loro inizializzazione
 - possono cambiare tipo (automaticamente) per adattarsi al contesto in cui vengono usate

Creazione di una variabile

- **creazione esplicita tramite l'istruzione "var" (con o senza un valore iniziale):**
 - var totale;
 - var totale = 0;
 - var saluto = "ciao mamma!"
- **creazione implicita assegnando un valore alla variabile:**
 - totale = 0;
- **se si usa una variabile senza averle prima assegnato un valore:**
 - undefined / NaN (se dichiarata con "var")
 - runtime error (se non dichiarata)

Costanti

- **tramite l'istruzione "const" si può dichiarare una variabile con valore fisso:**
 - const iva = 0.21;
 - const autore = "A.Lioy";
- **all'interno di una stringa si possono usare:**
 - caratteri ISO-8859-1
 - caratteri Unicode
 - sequenze di escape:
 - \r \n \t \' \" \\
 - \nnn(ottale) \xNN (hex) \uNNNN (Unicode hex)
 - attenzione all'uso (se stringa usata in HTML allora può richiedere codifica, es. ` ;)

Input ed output

- **JS è un linguaggio di scripting, pensato per essere eseguito lato client (in un browser) o lato server**
- **le funzioni di I/O dipendono dall'ambiente di esecuzione**
- **per il lato client si possono usare:**
 - per l'input
 - i dati provenienti da una finestra pop-up di input
 - i dati provenienti da un form HTML (tramite DOM)
 - per l'output
 - un finestra pop-up di output
 - la pagina HTML (mentre viene "costruita" o tramite DOM)

Pop-up di I/O

- **window.alert(*messaggio*)**
 - apre un pop-up bloccante contenente il messaggio ed un pulsante per conferma di lettura (OK)
- **window.prompt(*prompt_msgf* , *valore_iniziale*)**
 - apre un pop-up bloccante contenente il messaggio di prompt, un campo di input (vuoto o col valore iniziale specificato) e due pulsanti per inserire la risposta (OK) o per non fornirla (Cancel)
 - restituisce il valore introdotto, oppure null nel caso l'utente abbia premuto Cancel o chiuso il pop-up

Pop-up di I/O

- **window.confirm(*messaggio*)**
 - apre un pop-up bloccante contenente il messaggio e due pulsanti (per rispondere in modo affermativo o negativo)
 - restituisce un valore Booleano (true se la risposta è affermativa, false altrimenti)
- **a rigor di termini sono tutte implementazioni JS dell'oggetto window (DOM livello 0) e relativi metodi**

Pop-up di I/O: esempio di alert

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
  "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Pop-up di I/O: esempio 0</title>
  <script type="text/javascript">
    window.alert("Attenzione! questa pagina usa JS")
  </script>
</head>
<body>
  <p>Fine dell'esempio.</p>
</body>
</html>
```

alert.html

Pop-up di I/O: esempio 1

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Pop-up di I/O: esempio 1</title>
  <script type="text/javascript">
    var n = window.prompt("Nome?", "nessuno")
    window.alert(n);
  </script>
</head>
<body>
  <p>Fine dell'esempio.</p>
</body>
</html>
```

nome1.html

Pop-up di I/O: esempio 2

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
  <title>Pop-up di I/O: esempio 2</title>
</head>
<body>
  <p>Inizio dell'esempio.</p>
  <script type="text/javascript">
    var n = window.prompt("Nome?", "nessuno")
    window.alert("Ciao "+n);
  </script>
  <p>Fine dell'esempio.</p>
</body>
</html>
```

nome2.html

Output tramite HTML

- si può usare l'oggetto DOM "document" con uno dei seguenti metodi:
 - write(text)
 - inserisce il testo
 - writeln(text)
 - inserisce il testo seguito da CR LF
 - il testo è inserito nel punto ove si incontra lo script

Output tramite HTML: esempio

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Output tramite HTML: esempio</title>
</head>
<body>
<script type="text/javascript">
var n = window.prompt("Nome?", "nessuno");
document.writeln("<p>Ciao "+n+"</p>");
</script>
</body>
</html>
```

nome3.html

Operatori relazionali e logici

descrizione	simbolo
uguaglianza (valore)	==
identità (valore e tipo)	===
disuguaglianza (valore)	!=
non identità (valore e tipo)	!==
maggiore di / maggiore o uguale a	> >=
minore di / minore o uguale a	< <=
appartenenza	in
AND logico	&&
NOT logico	!
OR logico	

Operatori e valori Booleani

- i seguenti valori sono equivalenti a Falso:
 - false
 - 0
 - NaN
 - la stringa vuota ""
 - null
 - undefined
- qualunque altro valore è equivalente a Vero
- attenzione quindi ai confronti:
 - (27 == true) fornisce valore Vero
 - (27 === true) fornisce valore Falso

Operatori aritmetici

descrizione	simbolo
addizione	+
incremento unitario	++
sottrazione	-
decremento unitario	--
moltiplicazione	*
divisione (floating-point)	/
modulo (resto della divisione intera)	%

Operatori di assegnazione

descrizione	simbolo	esempio	equivalenza
assegnazione	=	a = 5	
assegn. con somma	+=	a += 5	a = a + 5
assegn. con sottrazione	-=	a -= 5	a = a - 5
assegn. con prodotto	*=	a *= 5	a = a * 5
assegn. con divisione	/=	a /= 5	a = a / 5
assegn. con modulo	%=	a %= 5	a = a % 5

Le stringhe di caratteri

- particolarmente importanti perché qualunque input fornito dall'utente tramite browser è una stringa
- operatori:
 - assegnazione (=)
 - confronto in ordine alfabetico (== != > >= < <=)
 - concatenazione (+ +=)
- attenzione! se un'istruzione contiene stringhe, numeri ed il simbolo +, tutto viene trattato come stringhe; si suggerisce perciò l'uso delle parentesi:

```
ris = "N=" + 5 + 2; // ris = "N=52"
ris = "N=" + (5 + 2); // ris = "N=7"
ris = "N=" + 5 - 2; // ris = NaN
```

Conversioni stringhe - numeri (I)

- **Number(oggetto)**
 - restituisce una rappresentazione numerica dell'oggetto o NaN
- **String(oggetto)**
 - restituisce una rappresentazione come stringa di caratteri dell'oggetto o undefined o null

```
n = Number("2"); // n = 2
n = Number("2.3"); // n = 2.3
n = Number("2,3"); // n = NaN
n = Number("2mila"); // n = NaN
n = Number("2 mila"); // n = NaN
```

Conversioni stringhe - numeri (II)

- **parseInt(stringa [, base])**
 - restituisce un numero intero o NaN
 - possibile specificare la base numerica (default: 10)
- **parseFloat(stringa)**
 - restituisce un numero floating-point o NaN
- **parseInt e parseFloat considerano solo la parte iniziale, fermandosi al primo carattere non valido**

```
n = parseInt("10",2); // n = 2
n = parseInt("2.3"); // n = 2
n = parseInt("2,3"); // n = 2
n = parseInt("2mila"); // n = 2
n = parseInt("duemila"); // n = NaN
```

Proprietà e metodi dell'oggetto String (I)

- **importanti perché usabili anche su variabili stringa**
- **length**
 - lunghezza N della stringa (indice da 0 a N-1)
- **charAt(pos)**
 - il carattere in posizione pos
- **charCodeAt(pos)**
 - il codice numerico Unicode del carattere in posizione pos
- **indexOf(searchString [, start])**
 - posizione della stringa cercata (a partire da start o dall'inizio)
 - -1 se la stringa non viene trovata

Proprietà e metodi dell'oggetto String (II)

- **lastIndexOf(searchString [, start])**
 - posizione della stringa cercata (a partire da start o dalla fine)
 - -1 se la stringa non viene trovata
- **slice(begin [, end])**
 - crea una nuova stringa coi caratteri presenti da begin a end (escluso) oppure alla fine
 - usare end negativo per indicare posizioni dalla fine
- **substring(begin [, end])**
 - estrae i caratteri da begin a end (o alla fine)
- **substr(begin [, length])**
 - estrae i caratteri da begin per la quantità length

Proprietà e metodi dell'oggetto String (III)

- **toLowerCase()**
 - restituisce i caratteri convertiti in minuscolo
- **toUpperCase()**
 - restituisce i caratteri convertiti in maiuscolo

Test su valori errati

- **non si può fare un confronto con NaN o altri valori limite, ma si possono usare funzioni che testano questi casi**
- **isFinite(number)**
 - vero se il numero non è pari a +/- infinito o NaN
- **isNaN(number)**
 - vero se il numero ha il valore NaN
- **typeof(x)**
 - restituisce una stringa che esprime il tipo di dato attualmente corrispondente a X
 - risposte possibili: boolean, function, number, object, string, undefined

Controllo di flusso

- strutture di controllo utili per eseguire un programma in modo non sequenziale
 - if
 - if/else
 - while
 - do/while
 - for
 - for/in

Controllo di flusso "if" / "if-else"

- esecuzione condizionale di istruzioni in base al valore di una condizione Booleana
 - if
 - if/else

```
if ( condizione )
{
  ... istruzioni
}
```

```
if ( condizione )
{
  ... istruzioni
}
else
{
  ... istruzioni
}
```

Esempio di costrutto "if-else"

```
<script type="text/javascript">
var t_mis = window.prompt("Temperatura misurata?");
if (t_mis <= 0)
  alert("l'acqua e' ghiacciata");
else if (t_mis >= 100)
  alert("l'acqua e' vapore");
else
  alert("l'acqua e' allo stato liquido");
</script>
```

acqua.html

Selezione multipla: l'istruzione "switch"

- è una forma abbreviata di cascata di "if-else"
- uso di "break" per non continuare col caso successivo
- "default" se non si ricade in nessun caso esplicito

```
switch ( espressione )
{
  case valore1: ... istruzioni;
    break;
  case valore2: ... istruzioni;
    break;
  ...
  default: ... istruzioni;
}
```

Esempio di costrutto "switch"

```
<script type="text/javascript">
var frutto = window.prompt("Quale frutto vuole?");
switch (frutto) {
case "pera":
  alert ("pere a 2 Euro/kg"); break;
case "mela":
  alert ("mele a 1.5 Euro/kg"); break;
case "banana":
  alert ("banane a 1 Euro/kg"); break;
default:
  alert ("spiacenti, non abbiamo "+frutto);
}
</script>
```

Controllo di flusso "while"

- struttura per ripetere un blocco di istruzioni finché una condizione è e rimane vera
- le istruzioni del ciclo possono quindi essere eseguite zero o più volte

```
while ( condizione )
{
  ... istruzioni
}
```

Esempio di ciclo "while"

```
<script type="text/javascript">
// conto alla rovescia
var x = 5;
while (x >= 0)
{
    alert(x);
    x--;
}
</script>
```

Controllo di flusso "do-while"

- struttura simile al while con la differenza che il controllo si fa alla fine del ciclo e quindi il ciclo viene sempre eseguito almeno una volta

```
do
{
    ... istruzioni
} while ( condizione );
```

Esempio di ciclo "do-while"

```
<script type="text/javascript">
var ris;
do {
    ris = window.prompt(
        "Scrivi 'ciao' o resti bloccato qui");
} while (ris != "ciao");
</script>
```

Controllo di flusso "for"

- struttura per ripetere blocchi di istruzioni finché una condizione rimane vera
- specifica:
 - un'azione di inizializzazione
 - una condizione
 - un'azione da ripetere alla fine di ogni ciclo (tipicamente un incremento/decremento dell'indice associato al ciclo)

```
for ( inizializzazione ; condizione; azione_ripetitiva )  
{  
  ... istruzioni_da_ripetere  
}
```

Esempio di ciclo "for" numerico

```
<script type="text/javascript">  
/*  
calcolo della somma  
dei primi 10 numeri naturali  
*/  
var totale = 0;  
for (var i=1; i <= 10; i++)  
{  
  totale = totale + i;  
}  
alert("Somma dei numeri [1...10] = "+totale);  
</script>
```

Istruzioni "break" e "continue"

- l'istruzione "break" (oltre ad essere usata nello switch) interrompe l'esecuzione del ciclo in cui è contenuta; l'esecuzione continua dalla prima istruzione successiva al ciclo
- l'istruzione "continue" blocca l'esecuzione del passo corrente e fa iniziare immediatamente l'esecuzione del prossimo passo del ciclo in cui è contenuta

Array (vettori)

- in JS sono oggetti e quindi devono essere istanziati (con una dimensione iniziale che può essere cambiata dinamicamente)
- possono essere indicizzati da un intero o da una stringa (array associativi)
- possiedono proprietà e metodi per inserire, cancellare e reperire gli elementi
- esempio:

```
var Vettore = new Array(10);
for (var i=0; i<10; i++) {
  Vettore[i] = "Test " + i;
}
```

Esempio vettore con indice numerico

```
<script type="text/javascript">
// array per conversione
// da voto italiano a voto europeo
var it2eu = new Array(32) // 0 ... 30 30L
for (var i=0; i<18; i++) it2eu[i] = "D"
for (var i=18; i<24; i++) it2eu[i] = "C"
for (var i=24; i<29; i++) it2eu[i] = "B"
for (var i=29; i<=31; i++) it2eu[i] = "A"

var voto = prompt("Voto italiano?")
alert("Voto europeo = " + it2eu[voto])
</script>
```

voto.html

Esempio vettore con indice non numerico

```
<script type="text/javascript">
// vocabolario italiano - inglese
var vocab = new Array()
vocab["giallo"] = "yellow"
vocab["rosso"] = "red"
vocab["verde"] = "green"

var colore = prompt("Colore?")
alert( colore + " = " + vocab[colore] )
</script>
```

vocab1.html

Esempio vettore con indice non numerico

```

<script type="text/javascript">

// vocabolario italiano - inglese
var vocab = new Array()
vocab["giallo"] = "yellow"
vocab["rosso"] = "red"
vocab["verde"] = "green"

var colore = prompt("Colore?")
if (typeof(vocab[colore]) != "undefined")
  alert( colore + " = " + vocab[colore] )
else
  alert("Spiacente, traduzione non disponibile")
</script>

```

vocab2.html

Controllo di flusso "for-in"

- scandisce gli elementi di un vettore (senza conoscerne la dimensione)
 - l'indice assume tutti i valori numerici (0 ... length-1)
- scandisce le proprietà di un oggetto (senza conoscere il nome delle proprietà)
 - l'indice assume tutti gli id delle proprietà, nell'ordine in cui sono dichiarate

```

for (x in vettore) {
  ... vettore[x] ...
};

```

```

for (x in oggetto) {
  ... oggetto[x] ...
}

```

Esempio di "for-in" con vettori

```

var vettore = new Array(10);

// ciclo esplicito su tutti gli elementi
for (var i=0; i<vettore.length(); i++)
  vettore[i] = "test"+i;

// ciclo esplicito ma piu' efficiente
for (var i=0, n=vettore.length(); i<n; i++)
  vettore[i] += "!";

// ciclo implicito su tutti gli elementi
for (var i in vettore)
  document.writeln(vettore[i]+"<br>");

```

Esempio di "for-in" con oggetti

```

var myObject = new Object();
myObject.name = "Antonio";
myObject.age = 24;
myObject.phone = "5551234";

...

// ciclo implicito su tutte le proprietà
for (var prop in myObject)
{
  document.writeln("<p>myObject." + prop
    + " = " + myObject[prop] + "</p>");
}

```

Funzioni

- le informazioni passate alle funzioni si chiamano **parametri**
- i parametri vengono specificati tra parentesi dopo il nome della funzione
- tra le istruzioni si può usare **"return"** per terminare l'esecuzione della funzione, eventualmente restituendo un valore tramite **"return(valore)"**

```

function nome_funzione (par1, par2, ...)
{
  ... istruzioni ...
}

```

Esempi di funzioni

```

function somma (a, b) { return(a+b); }
document.write(somma(1,2));

```

```

function minoreDi (a, b) {
  if (a < b) return (true) else return (false);
}

var a=5;
var b=2;
if (!minoreDi(a,b))
  document.write(a + "non e' minore di " + b);

```

Variabili locali e globali

- **variabile dichiarata all'interno di una funzione:**
 - accessibile solo alle istruzioni nella funzione stessa
 - distrutta automaticamente al termine della funzione
- **variabile dichiarata all'esterno di qualunque funzione:**
 - accessibile a tutte le istruzioni dello script (incluse quelle nelle funzioni richiamate dallo script)
 - distrutta automaticamente al termine dello script

Variabili locali e globali: esempio

```

<script type="text/javascript">
var i=2; // variabile globale
function print_var()
{
  var j=4; // variabile locale a print_var()
  alert("print_var(): i="+i);
  alert("print_var(): j="+j);
}
print_var();
alert("i="+i);
alert("j="+j );
document.writeln("<p>Fine dello script.</p>")
</script>
<p>Fine della pagina.</p>

```

glocal.html

Funzioni e parametri

- **una funzione può essere richiamata con meno parametri di quelli definiti**
 - i parametri mancanti sono undefined, ma (solo in questo caso) il loro uso non genera errore
 - hanno un valore non definito che si propaga se usato (es. genera NaN in un calcolo aritmetico)
- **una funzione può essere richiamata con più parametri di quelli definiti**
 - i parametri in eccesso sono ignorati
- **una funzione può accedere tutti i suoi parametri tramite il vettore arguments[] che contiene arguments.length valori distinti**

Argomenti variabili delle funzioni: esempio

```
<script type="text/javascript">
function media()
// calcola la media aritmetica di tutti
// i numeri passati come parametri
{
var total = 0;
var n = arguments.length;
for (var i=0; i<n; i++)
total += arguments[i];
return (total / n);
}
// esempio di uso (media di tre numeri)
alert( media(11,12,16) );
</script>
```

L'oggetto Date

- **new Date()**
 - data e ora attuale (sul sistema che esegue lo script)
- **new Date("Month day, year [HH:MM:SS]")**
 - data e ora indicata (es. "March 25, 2009 22:00:07")
- **new Date(YYYY, MM, DD [, HH, MM, SS])**
 - data e ora indicata (es. 2009, 2, 25, 22, 00, 07)
 - numero del mese: 0=Gennaio, 11=Dicembre
- **se HH, MM o SS sono omessi allora sono zero**
- **attenzione! la rappresentazione della data come stringa dipende dal S.O. su cui è eseguito lo script**
 - meglio quindi impostare individualmente i valori delle proprietà (con setDay(), setHour(), ...)

L'oggetto Date: metodi (!)

- **getDay() / setDay(giorno_settimana)**
 - giorno della settimana (0=Domenica, 6=Sabato)
- **getDate() / setDate(giorno_mese)**
- **getMonth() / setMonth (mese_num)**
 - numero del mese (0=Gennaio, 11=Dicembre)
- **getFullYear() / setFullYear (anno)**
- **getHours() / setHours (ora)**
- **getMinutes() / setMinutes(minuti)**
- **getSeconds() / setSeconds(secondi)**
- **disponibili tutti questi metodi riferiti a UTC:**
 - getUTC...() / setUTC...()

L'oggetto Date: metodi (II)

- **toString()**
 - converte in stringa secondo il formato nativo di JS (ossia anglosassone)
- **toLocaleString()**
 - converte in stringa secondo il formato locale impostato per l'utenza che esegue lo script

L'oggetto Math – proprietà

- **proprietà statiche (da richiedere sull'oggetto Math):**
 - E, costante di Eulero (circa 2.718)
 - LN2, ln(2) (circa 0.693)
 - LN10, ln(10) (circa 2.302)
 - LOG2E, log₂(e) (circa 1.442)
 - LOG10E, log₁₀(e) (circa 0.434)
 - PI, pi greco (circa 3.14159)
 - SQRT1_2, radice quadrata di 1/2 (circa 0.707)
 - SQRT2, radice quadrata di 2 (circa 1.414)

L'oggetto Math – metodi

- **metodi statici (da invocarsi sull'oggetto Math)**

metodo	definizione
abs (x)	valore assoluto
asin (x) acos (x) atan (x)	
atan2 (y, x)	atan (y / x)
sin (x) cos (x) tan (x)	
ceil (x) floor (x)	ceiling, floor
exp (x) pow (x, y) sqrt (x)	e ^x x ^y √x
log (x)	ln(x)
round (x)	arrotondamento
max (x, y) min (x, y)	
random (seed)	valore casuale [0..1[

L'oggetto Number – proprietà

- **proprietà statiche** (da richiedere sull'oggetto Number):
 - MAX_VALUE
 - MIN_VALUE
 - NaN
 - NEGATIVE_INFINITY (overflow negativo)
 - POSITIVE_INFINITY (overflow positivo)

L'oggetto Number – metodi

- **toFixed(num_cifre_frazionarie)**
 - converte in formato non esponenziale
- **toexponential(num_cifre_frazionarie)**
 - converte in formato esponenziale
- **toprecision(num_cifre_significative)**
 - converte alla precisione indicata (eventualmente usando il formato esponenziale se necessario)
- **tutti questi metodi arrotondano il risultato se il numero di partenza ha più cifre del necessario:**

```
var num=5.126
alert(num.toprecision(3)) // visualizza 5.13
alert(num.toprecision(2)) // visualizza 5.1
```

Interazione tra JS e HTML

- l'interazione tra JS client-side, lo UA e la pagina HTML è definita da un'apposita appendice del DOM
- es. appendice E del DOM livello 1:

```
The HTMLDocument object has the following methods:
...
write(text)
  This method returns a void.
  The text parameter is of type DOMString.
...
getElementById(elementId)
  This method returns a Element.
  The elementId parameter is of type DOMString.
getElementsByName(elementName)
  This method returns a NodeList.
  The elementName parameter is of type DOMString.
```
