LEZIONE 7

Che cosa è una variabile e come si dichiara ?

Definire e/o inizializzare una **variabile**

```
var adesso;
var adesso = new Date();
```

Che cosa è una funzione e come la definisco?

Dichiarare e definire una **funzione**

```
function somma(n1, n2){
    return n1 + n2;
}
```

funzione con nome

Dichiarare e definire una **funzione**

```
var somma = function(n1, n2){
    return n1 + n2;
}
```

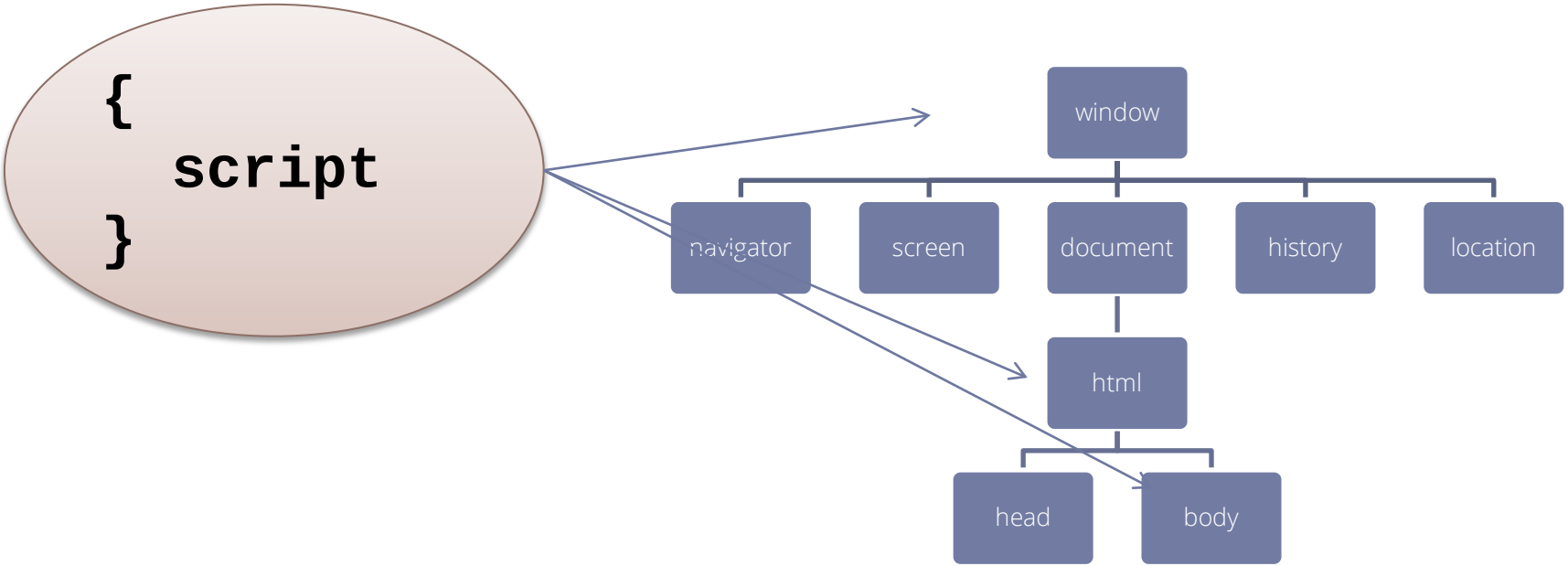
funzione anonima

Che regole deve rispettare il nome di una funzione o di una variabile?

1. Iniziare il nome con una lettera (A-Z o a-z) l'underscore (_) o il segno del dollaro ($).
2. Continuare con un numero qualsiasi di lettere, numeri, "_" 0 "$".
3. Javascript è case sensitive.

Come agisce javascript sulla tua pagina?

JAVASCRIPT AGISCE SUL DOM



Perché javascript possa agire sugli oggetti il documento deve essere completamente caricato!

Come procedo? E perché?

1

Primo metodo.

- Aggiungo il codice tramite il tag <script> subito prima del tag di chisura di body.

JAVASCRIPT DOVE?

```
<!DOCTYPE html>
<html>
    <head>
        <title>La mia prima pagina XHTML</title>
    </head>
    <body>
        <h1>Benvenuto!</h1>
        <p>Questo &egrave; il mondo di XHMTL!</p>
        ...
        <script type="text/javascript" src="mioscript.js"></script>
    </body>
</html>
```

```
// contenuto del file mioscript.js

function eseguiCodice ()
{
  document.getElementById("eval_txt").value =
     eval(document.getElementById("espressione_txt").value);
}

/*
   L'evento onclick può essere assegnato immediatamente
   al bottone con id "eval_btn" perché il DOM è già caricato
*/
document.getElementById("eval_btn").onclick = eseguiCodice;
```

2

Secondo metodo.

- Aggiungo il codice tramite il tag <script> nella sezione <head> della pagina e uso l'evento window.onload per eseguire il codice che accede ad elementi del **dom**.

JAVASCRIPT DOVE?

```
<!DOCTYPE html>
<html>
    <head>
        <title>La mia prima pagina XHTML</title>
        <script type="text/javascript" src="mioscript.js"></script>
    </head>
     <body>
         <h1>Benvenuto!</h1>
         <p>Questo &egrave; il mondo di XHMTL!</p>
     </body>
</html>
```

```
// contenuto del file mioscript.js
  function eseguiCodice ()
  {
    document.getElementById("eval_txt").value =
    eval(document.getElementById("espressione_txt").value);
  }
  /* Per assegnare l'evento onclick devo aspettare il
     caricamento del DOM */
  function caricamentoPagina()
  {
    document.getElementById("eval_btn").onclick = eseguiCodice;
  }
  window.onload = caricamentoPagina;
```

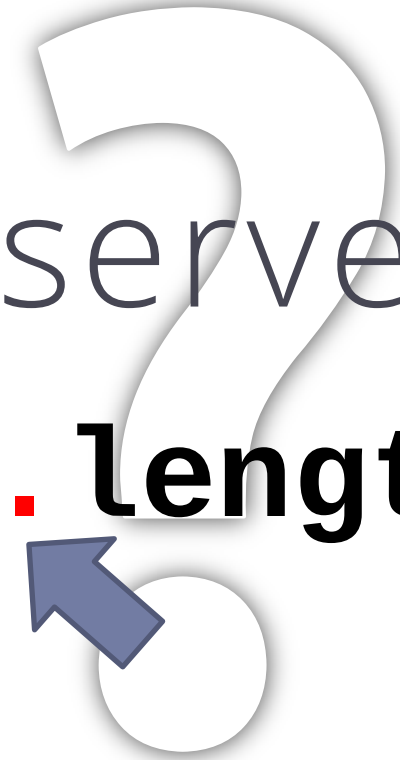
A cosa serve il ";"?

```
var nome;
var oggi;
```

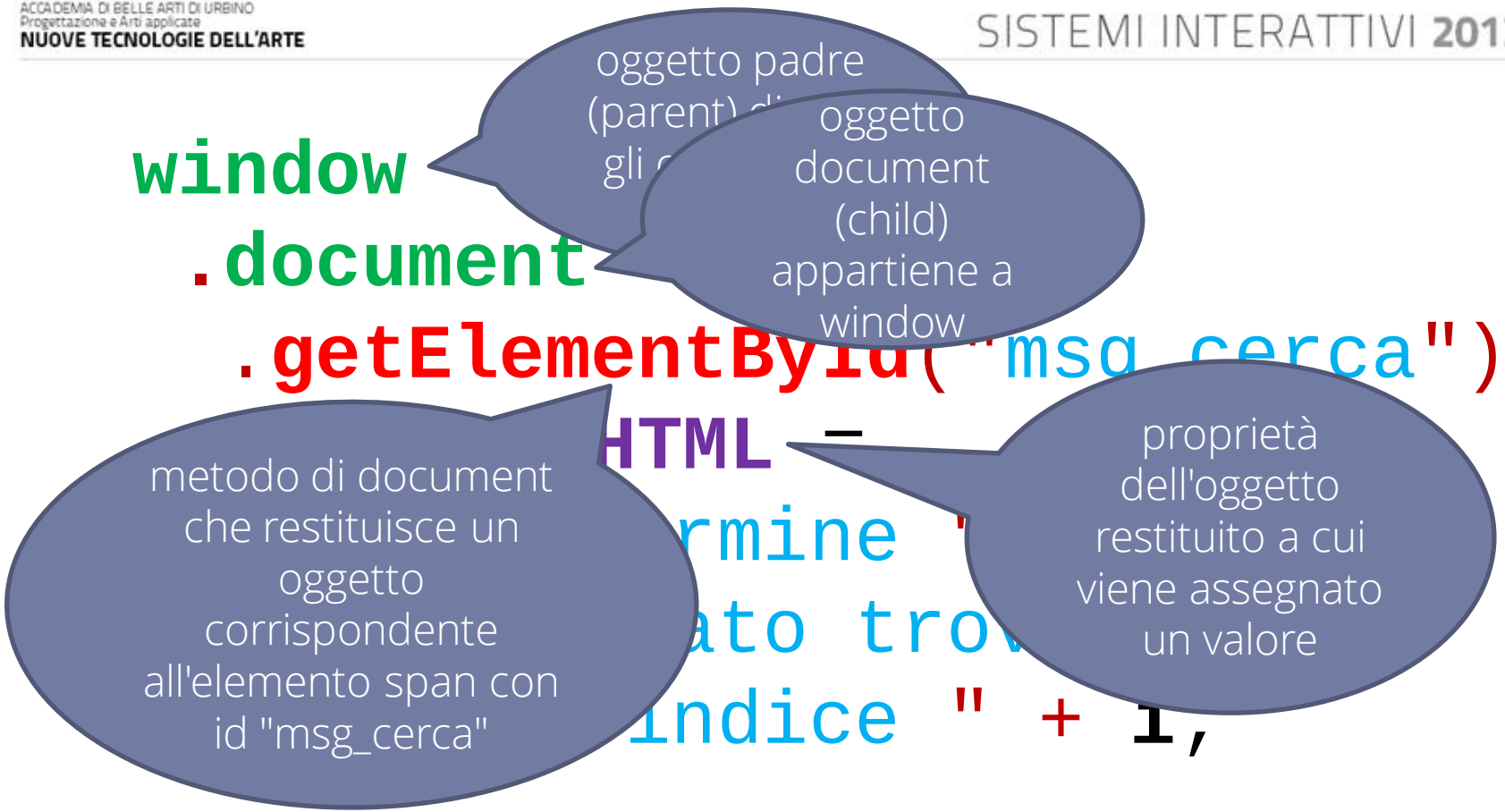
- Il punto e virgole è il segno che separa le istruzioni fra loro.
- Le istruzioni NON sono separate dalla fine riga.

A cosa serve il "."?

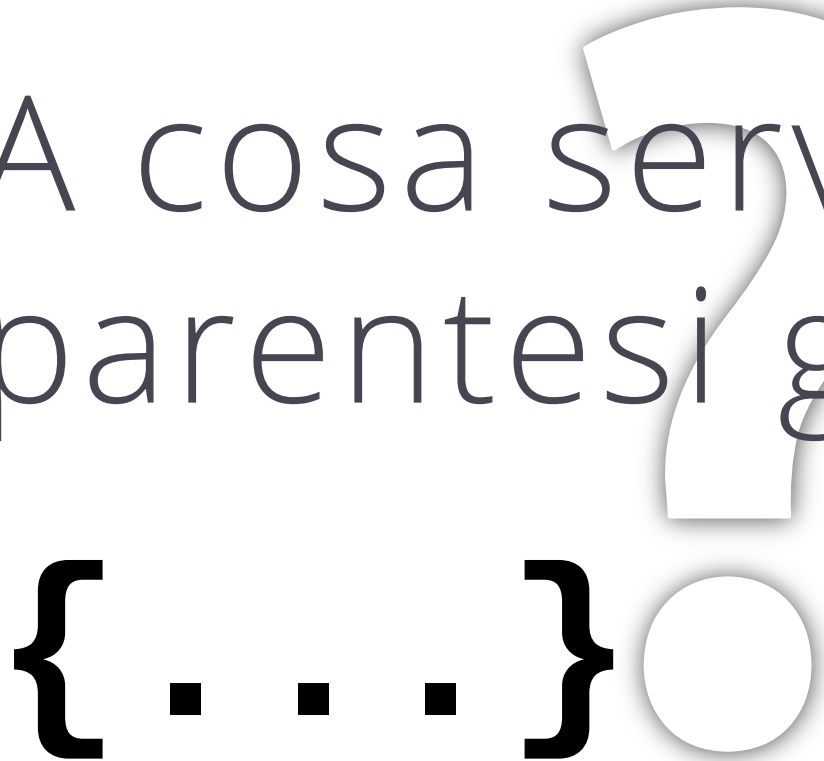
```
if (str.length < 2)
```



- Il punto è l'operatore di appartenenza indica che la proprietà o il metodo che si trova alla sua destra appartiene all'oggetto che si trova alla sua sinistra.
- Gli operatori punto possono essere usati in catena.

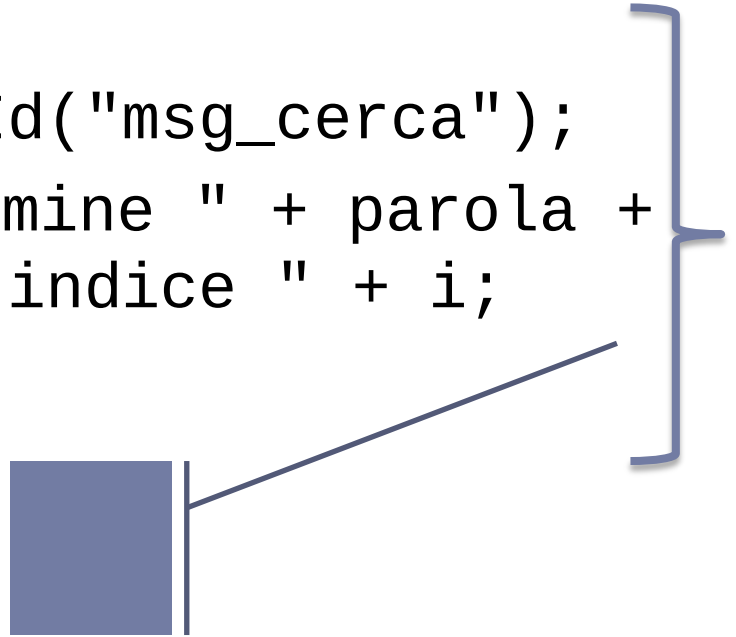

oggetto padre
(parent) di
gli
oggetto
document
(child)
appartiene a
window
window
.document
.getElementById("msg_cerca")
HTML =
rmine "
ato trov
ndice " + i;
metodo di document
che restituisce un
oggetto
corrispondente
all'elemento span con
id "msg_cerca"
proprietà
dell'oggetto
restituito a cui
viene assegnato
un valore

A cosa servono le parentesi graffe ?



- Una coppia di { } definisce un blocco di istruzioni che vengo eseguite insieme, prima di proseguire con l'esecuzione del programma.

```
if (mesi[i] == parola)
{
  var campo =
     document.getElementById("msg_cerca");
  campo.innerHTML = "Il termine " + parola +
     " è stato trovato all'indice " + i;
  return;
}
```



Uso delle parentesi tonde: Quale è la differenza tra

```
somma;
```

e

```
somma(5, 8);
```

- **`somma`** (che è il nome che ho dato alla funzione) è la variabile che contiene le istruzioni di cui la funzione è composta. Il valore rappresentato da **`somma`** è la funzione stessa, il blocco di script di cui è costituita.
- se faccio seguire a **`somma`** le parentesi tonde, tra le quali inserirò la lista dei parametri se previsti, la funzione viene eseguite e il valore di **`somma(…)`** sarà il valore restituito dell'istruzione **`return`** se presente altrimenti sarà undefined

Uso degli operatori logici: Quale è la differenza tra

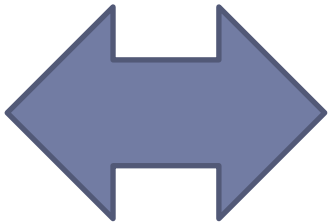
e

- = è l'operatore di assegnazione. Assegna il valore che si trova alla sua destra alla variabile o alla proprietà che si trova alla sua sinistra:

```
a = 10;
```

- il risultato di questa espressione è 10;

- == è un operatore di confronto. Confronta i due valori (costanti, variabili od espressioni) che unisce. Il risultato dell'espressione è true se i due valori sono uguali altriementi false.



a == b;

questa espressione dà come risultato true ?

```
a > b || b == 5
```

e questa ?

```
a > b && b == 5
```

PRENDERE DECISIONI

LE STRUTTURE DI CONTROLLO

- Le strutture di programmazione che mi consentono di prendere decisioni sono essenzialmente due:
  - **condizionale**: faccio una determinata cosa se una condizione risulta vera altrimenti ne faccio un'altra
  - **iterativa** (o loop): ripeto una determinata operazione finche una condizione risulta vera

GLI OPERATORI LOGICI

operazione	javascript	precedenza
uguaglianza	==	1
disuguaglianza	!=	1
maggiore	>	1
maggiore o uguale	>=	1
minore	<	1
minore o uguale	<=	1
and	&&	2
or	||	2
not	!	2

SINTASSI DELL'ISTRUZIONE IF

- L'istruzione if può avere due forme:

  if (espressione) blocco di istruzioni

  if (espressione) blocco di istruzioni else blocco di istruzioni
- L'espressione che compare dopo la parola chiave if deve essere di tipo logico, se la condizione risulta vera viene eseguita l'istruzione subito seguente; nel secondo caso, invece, se la condizione risulta vera si esegue l'istruzione seguente, altrimenti si esegue l'istruzione subito dopo la parola chiave **else**.
- Per più scelte invece si può usare l'**else if** che permette di porre una condizione anche per le alternative, lasciando ovviamente la possibilità di mettere l'else (senza condizioni) in posizione finale.

BLOCCO IF

```
If (condizione)
{
  //comandi se condizione è vera
}
// il programma continua qui
```

BLOCCO IF ELSE

```
If (condizione)
{
  comandi se condizione è vera
}
else
{
  comandi se condizione è falsa
}
// il programma continua qui
```

ESEMPIO 1

```
/**
 * Funzione che formatta ore minuti e secondi
*/
function zeroPrima(n)
{
   //converto n in stringa concatenandolo a str
   var str = "";
   str = str + n;
   // se la lunghezza della stringa n è minore di 2
   // aggiungo uno 0 in testa
   if (str.length < 2){
     str = "0" + str;
   }
   return str;
}
```

ESEMPIO 2

```
var confronta = function ()
{
   var n = parseFloat(document.getElementById("numero").value);
   var c = parseFloat(document.getElementById("confronta").value);
   var message = "";
   if (isNaN(c) || isNaN(n))
   {
      message = "Errore: almeno uno dei valori inseriti non è un numero."
   }
   else if (c > n)
   {
      message = "Il numero inserito (" + c +") è maggiore del numero di riferimento."
   }
   else if (c == n)
   {
      message = "Il numero inserito (" + c +") è uguale del numero di riferimento."
   }
   else
   {
      message = "Il numero inserito (" + c +") è minore del numero di riferimento."
   }
   document.getElementById("messaggio_confronto").innerHTML = message;
}
```

La programmazione Iterativa

- Flusso naturale del programma:
  - viene eseguita un'istruzione dopo l'altra fino a che non si incontra l'istruzione di fine programma.
- Programmazione iterativa:
  - un'istruzione (o una serie di istruzioni) vengo eseguite continuamente, fino a quando non sopraggiungono delle condizioni che fanno terminare il ciclo.

for

- Il for inizializza una variabile, pone una condizione e poi modifica (normalmente incrementa o decrementa) la variabile iniziale.

```
for (inzializzazione; condizione; modifica)
  blocco istruzioni;
```

- Il codice <blocco istruzioni> viene eseguito fino a che l'espressione <condizione> risulta vera, poi si passa la all'istruzione successiva al for.

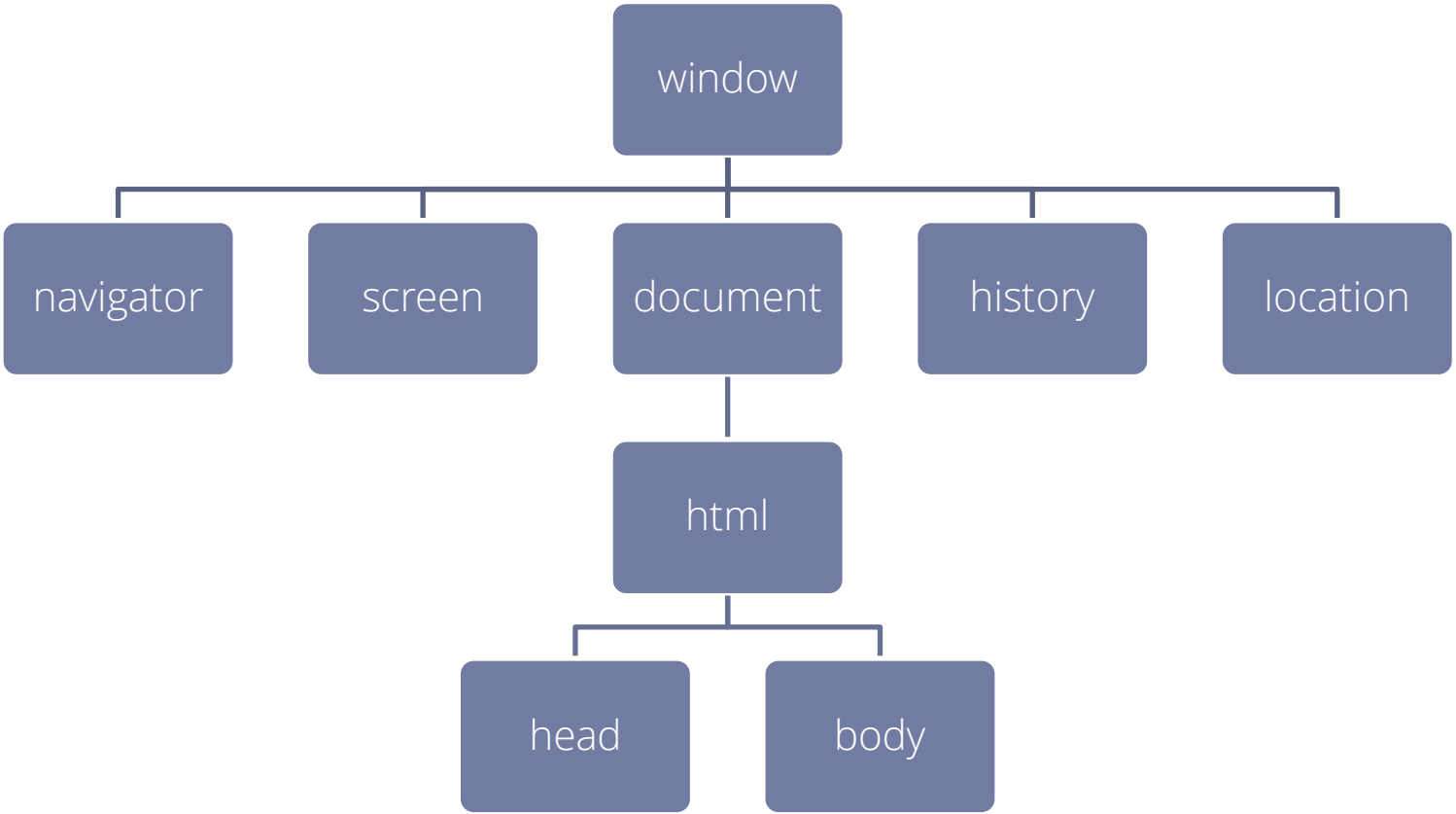
esempio

```
for (var i = 0; i < valoreMassimo; i++)
{
   // faccio qualcosa utilizzando in valore di
   // che incrementa ad ogni ciclo fino a che
   // non raggiunge il valore massimo
}
// quando i raggiunge il valore massimo il
// programma continua qui
```

esempio

```
var cerca = function()
{
  var str = document.getElementById("ricerca").value;
  for (var i = 0; i < mesi.length; i++)
  {
    if (mesi[i] == str)
    {
         document.getElementById("messaggio_ricerca").innerHTML =
            "La stringa " + str + " è stata trovata al posto " + i;
         return;
    }
  }
  document.getElementById("messaggio_ricerca").innerHTML = "La stringa " +
         str + " non è stata trovata.";
}
```

DOCUMENT OBJECT MODEL


window
navigator
screen
document
history
location
html
head
body

LA STRTTURA AD ALBERO

- Dopo che un documento viene caricato nel browser, gli oggetti vengono organizzati in memoria nella struttura gerarchica specificato dal **DOM**.
- Ogni elemento di questa struttura ad albero viene chiamato nodo.
- Ogni nodo può essere:
  - un nuovo ramo dell'albero (cioè avere o non avere altri nodi figli)
  - una foglia (non avere nodi figli)
- Nel DOM avremo:
  - elementi
  - nodi di testo

OBJECT REFERENCE

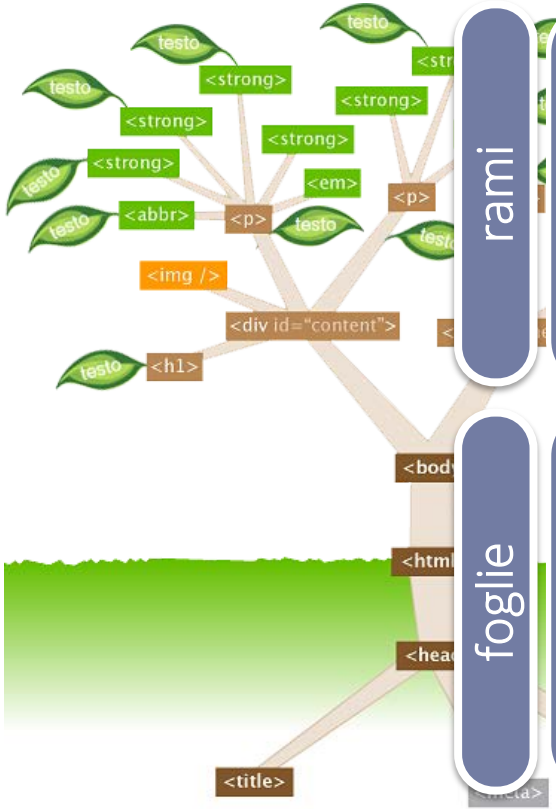
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo su cui agire
- Ad ogni nodo posso dare un nome unico utilizzando l'attributo id.
  - <p id="primoParagrafo" >
  - <img id="logo" src="images/logo.jpg" alt="Logo Azienda">
  -

DARE UN NOME AD UN NODO

- Per poter essere utilizzato facilmente in uno script l'ID di un oggetto deve seguire alcune regole:
  - non può contenere spazi
  - non devono contenere segni di punteggiatura tranne che per il carattere di sottolineatura (es.: primo_paragrafo)
  - deve essere racchiuso tra virgolette quando viene assegnato all'attributo id
  - non deve iniziare con un carattere numerico
  - Deve essere unico all'interno dello stesso documento

L'OGGETTO DOCUMENT

LA METAFORA DELL'ALBERO



element
sono i nodi che corrispondono ai tag HTML

- Tutti gli **element** possono avere attributi
- La maggior parte degli **element** può contenere altri nodi

textNode
sono i nodi che corrispondono al testo all'interno dei tag HTML

- i **textNode** non hanno attributi
- i **textNode** non contengono altri nodi
- i **textNode** hanno una proprietà che restituisce il testo che contengono

RECUPERARE GLI ELEMENTI

- getElementById(id)
  Questo metodo permette di recuperare l'elemento caratterizzato univocamente dal valore del proprio attributo ID e restituisce il riferimento all'elemento in questione.
- La sintassi è:

```
element = document.getElementById(ID_elemento);
```

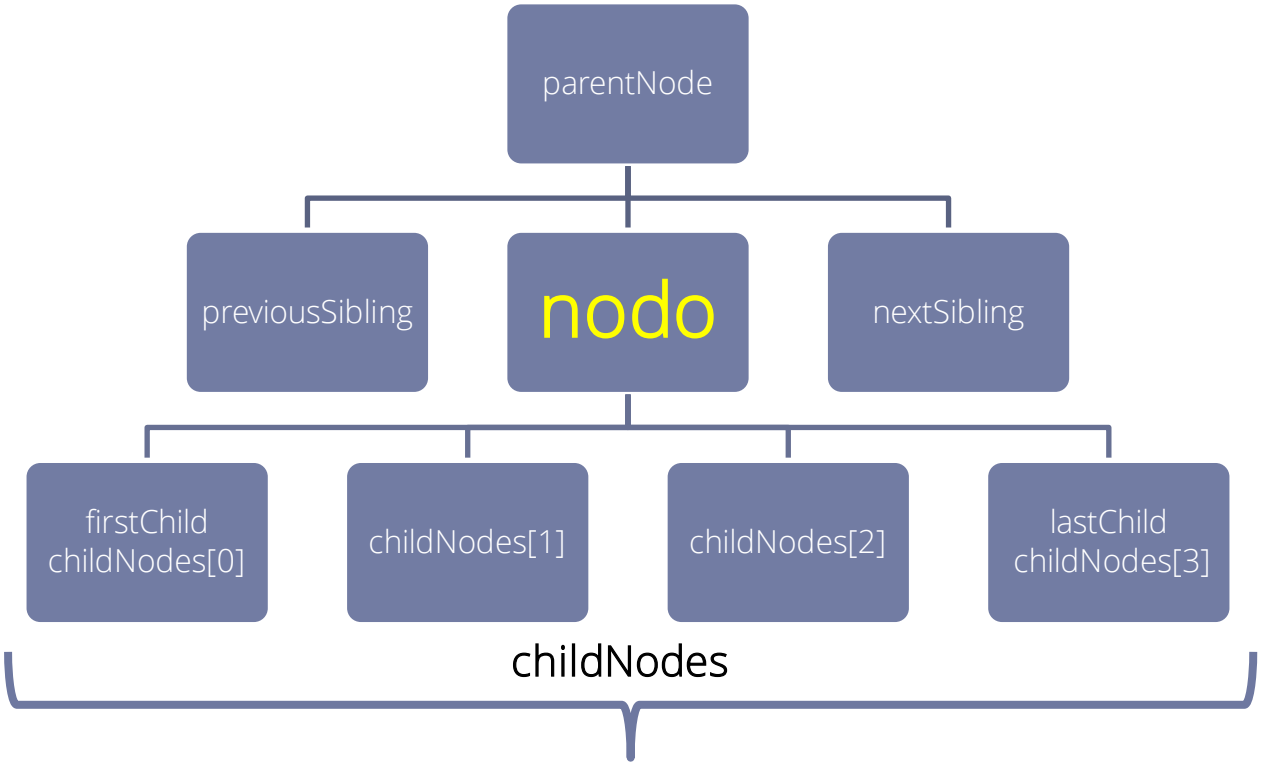
PROPRIETÀ DEI NODI

CONTENUTO DEL NODO

- innerHTML
  È una proprietà non standard introdotta originariamente da Internet Explorer , ma oggi supportata da tutti i maggiori browser. La proprietà restituisce il codice HTML compreso tra il tag di apertura e il tag di chiusura che definiscono l'elemento a cui è applicata.
- Sintassi:

```
elemento.innerHTML = "<p>Hello world! </p>";
testo = elemento.innerHTML;
```

RELAZIONE TRA I NODI



VALORI E RIFERIMENTI

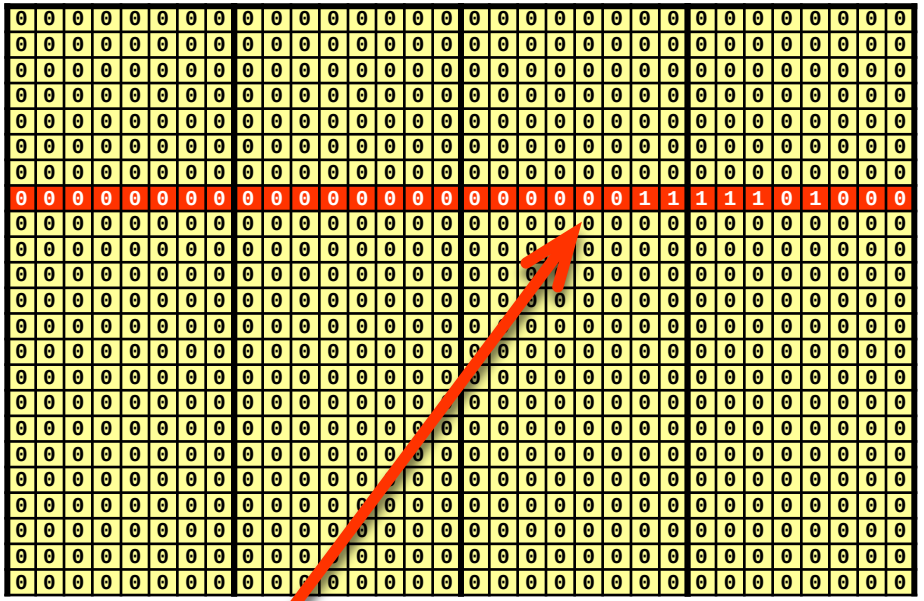
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso espresso in bit.
- Quando scrivo:

```
var a = 1000;
```

- Dico che a corrisponde ad una ben determinata cella di memoria composta da 32 bit in cui è scritto il formato binario il numero 1000.

VALORI E RIFERIMENTI

- La variabile a è associata a una cella di memoria.
- La cella contiene il valore di a in formato binario.

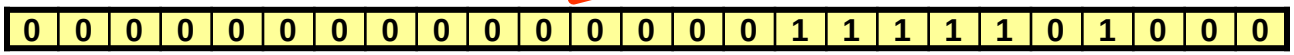


```
var a = 1000;
```

VALORI E RIFERIMENTI

- Se assegno ad **a** un numero intero stabilisco due cose
  - Che ad a vengono riservati 32 bit in memoria
  - Che il valore contenuto nella cella viene interpretato come numero intero

```
a = 1000;
a = -1;
```



VALORI E RIFERIMENTI

- Quando la casella che la variabile rappresenta contiene direttamente il dato si dice che la varibile contiene un valore.
- Se scrivo

```
var a = 10;
var b = a;
```

il valore di a viene copiato nella casella di memoria rappresentata da b e i due valori rimangono indipendenti.

VALORI E RIFERIMENTI

- Quando il valore assegnato a una variabile è un oggetto l'interprete javascript fa un operazione un po' più complessa. Lo spazio di 32 bit riservato alla variabile viene usato per memorizzare l'indirizzo di memoria in cui è collocato l'oggetto.
- In questo caso la variabile contiene il **riferimento** all'oggetto..
- Se scrivo:

```
var elemento = document.createElement("div");
```

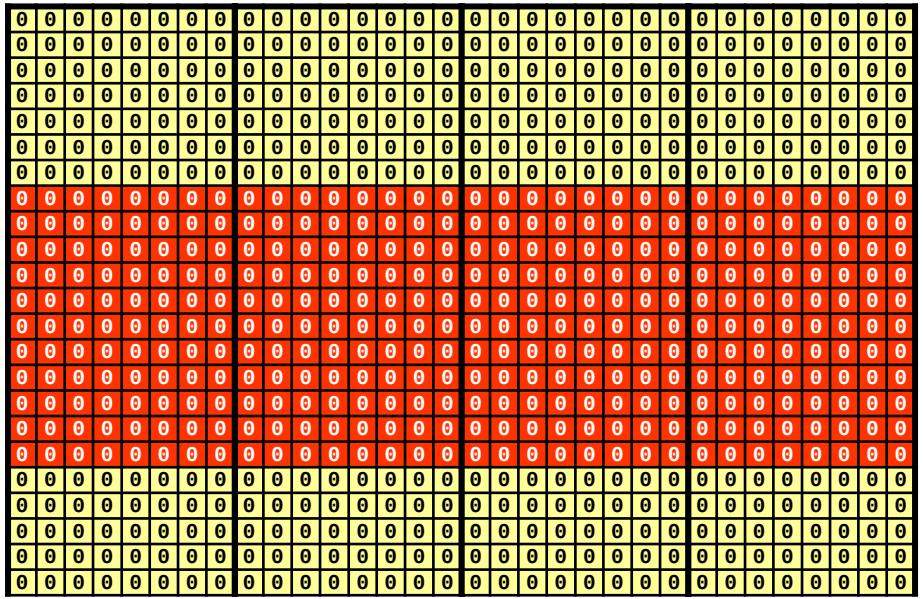
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

VALORI E PUNTATORI

```
var elemento = document.createElement("div");
```

Elemento contiene l'indirizzo alla zona di memoria in cui l'oggetto è memorizzato.

che punta a...



VALORI E RIFERIMENTI

- Quando la casella che la variabile rappresenta contiene l'indirizzo di memoria a partire dal quale e memorizzato l'oggetto si dice che la variabile, contiene il riferimento all'oggetto.
- L'interprete si occuperà automaticamente di risolvere il riferimento.

```
var elemento = document.createElement("div");
elemento.setAttribute("class", "articolo");
```

- Se però scrivo

```
var e = elemento;
```

quello che viene copiato in **e** è il riferimento all'oggetto ed entrambe le variabili si riferiranno allo stesso elemento.

LA LEGGIBILITÀ DEL CODICE

Leggibilità

- Scrivere programmi *sensati* e *leggibili* è difficile, ma molto importante
- È essenziale per lavorare in gruppo
- Aiuto il debugging
- Aiuta a riutilizzare il codice e quindi ci risparmia fatica

Leggibilità significa:

- Progettare con chiarezza
- Scrivere codice con chiarezza

Progettare con chiarezza

- Dedicare il tempo necessario alla progettazione della nostra applicazione non è tempo perso.
- Ci aiuterà a chiarire la logica e la sintassi del nostro lavoro.
- Più avremo sviluppato l'algoritmo che sta alla base della nostra applicazione più il nostro programma sarà comprensibile

Scrivere con chiarezza

- La chiarezza della scrittura si ottiene attraverso due *tecniche :*
- L'*indentazione*: inserire spazi o tabulazioni per mettere subito in evidenza le gerarchie sintattiche del codice.
- I *commenti*: inserire note e spiegazione nel corpo del codice.

Identazione: un esempio

- Prendiamo in esamo questo brano di codice HTML :

```
<table> <tr> <td>a</td> <td>b</td> <td>c</td>
</tr> <tr> <td> <table> <tr> <td>a1</td> </tr>
<tr> <td>a2</td> </tr> </table> </td> <td>b1</td>

<td>c1</td> </tr> </table>
```

Identazione: un esempio

- E confrontiamolo con questo:

```
<table>
      <tr>
        <td>a</td>
        <td>b</td>
        <td>c</td>
      </tr>
      <tr>
        <td>
        <table>
          <tr>
            <td>a1</td>
          </tr>
          <tr>
            <td>a2</td>
          </tr>
        </table>
        </td>
        <td>b1</td>
        <td>c1</td>
      </tr>
</table>
```

Identazione

- Si tratta delle stessa tabella, ma nel primo caso ci risulta molto difficile capire come è organizzata. Nel secondo la gerarchia degli elementi risulta molto più chiara.

Identazione

- L'identazione non ha nessun effetto sulla compilazione del programma
- Serve solo a rendere il nostro lavoro più leggibile.

Inserire commenti

- Rende il codice leggibile anche ad altri
- Quando decidiamo di apportare modifiche a cose che abbiamo scritto ci rende la vita più facile.

Delimitatori

- Delimitatori di riga: tutto cio che segue il contrassegno di commento fino alla fine della riga non viene compilato. Esempi:

  //

- Delimitatori di inizio e fine: tutto ciò compreso tra il contrassegno di inizio e il contrassegno di fine non viene compilato.

  /* .... */ <!-- .... -->

Commenti

JavaScript ha due tipi di commenti:

tag di apertura	tag di chiusura	descrizione
//	non si chiude	è un commento "veloce", che deve essere espresso in una sola riga senza andare a capo
/*	*/	si usa per scrivere commenti su più righe

```
<script type="text/javascript">
     // questo è un commento su una sola riga
    /*
    questo è un commento che sta su più righe,  serve
    nel caso in cui ci siano commenti particolarmente
    lunghi
    */
    alert("ciao");
</script>
```

I FRAMEWORK

I FRAMEWORK JAVASCRIPT

- Nella produzione del software, il framework è una struttura di supporto su cui un software può essere organizzato e progettato.
- Lo scopo di un framework è di risparmiare allo sviluppatore la riscrittura di codice già steso in precedenza per compiti simili.
- In altre parole utilizzando un framework lo sviluppatore può dedicare meno tempo alla scrittura del codice e più tempo alla progettazione e al raggiungimento degli obiettivi.

FRAMEWORK PIÙ DIFFUSI

Dojo
- dojotoolkit.org

ExtJs
- www.sencha.com

jQuery
- jquery.com

jQuery UI
- jqueryui.com

MooTools
- mootools.net

Prototype
- prototypejs.org

script.aculo.us
- script.aculo.us
- dipende da prototypr

Progettato per farti cambiare il modo in cui scrivi Javascript

VANTAGGI

- jQuery ha due vantaggi principali:
  - superare uno dei problemi che maggiormente complica la vita agli sviluppatori: la compatibilità tra le varie versioni dei browser
  - rendere lo script più compatto: scrivi di meno, fai di più.

INSERIRE JQUERY IN UNA PAGINA

- jQuery viene rilasciata in due versioni:
  - Compressa (che permette di avere file di dimensioni notevolmente più piccole)
  - Non compresso (versione leggibile e con commenti adatta per il debug, per fini didattici e per lo sviluppo).
- La versione compressa (minified) è contraddistista dal suffisso .min.

JQUERY SU CDN

- Un certo numero di grandi imprese mettono a disposizioni copie di jQuery su CDN (Content Deployment Network) pubblici:
  - Google Ajax API CDN (Disponibile anche download sicuro SSL via HTTPS)
    - http://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js
  - Microsoft CDN (Disponibile anche download sicuro SSL via HTTPS)
    - http://ajax.aspnetcdn.com/ajax/jQuery/jquery-1.11.1.min.js
  - jQuery CDN
    - http://code.jquery.com/jquery-1.11.1.min.js (Minified version)
    - http://code.jquery.com/jquery-1.11.1.js (Development version)

JQUERY SU CDN

- Per caricare jQuery (come qualsiasi altra libreria) si usa il tag script:

```
<script type="text/javascript"
   src="http://ajax.googleapis.com/ajax/libs/jquery/1.7.2/jquery.min.js">
</script>
```

- Se si usa un CDN per caricare jQuery (soluzione consigliata) può essere un buona idea preparare un'alternativa di caricamento così:

```
<script type="text/javascript">
   var scriptpath = //inserisci l'url di jQuery sul tuo sito
   if (!jQuery){
        document.write('<script type="text/javascript" src="' +
                scriptpath + '" </script>');
   }
</script>
```

L'INIZIO

- La libreria jQuery è costituita da codice che viene eseguito non appena caricata e che:
  - Crea un oggetto jQuery che è il namespace in cui ci muoveremo usando jQuery
  - Crea una funzione globale jQuery che è il cuore della libreria
  - Crea un alias per entrambe che è il segno del dollaro

jQuery = $

jQuery PROCESSO IN JAVASCRIPT

DIFFERENZE

- In javascript questo processo non è obbligatorio in certi casi posso eseguire il codice prima che la pagina sia caricata o assegnare direttamente agli tag HTML gli eventi
- In jQuery tutti i comandi devono essere eseguiti quando il DOM è completamente caricato e jQuery è correttamente inizializzato.

$('document').ready()

- Per ottenere questo ogni comando di jQuery va inserito in questo blocco:

```
$(document).ready( function() {


});
```

$('document').ready()

- O, in versione compatta, semplicemente:

```
$(function() {

});
```

LO STILE DI SCRITTURA

- Per capire meglio. Ho una funzione globale:

```
$(); //jQuery();
```

- e un metodo della classe jQuery:

```
.ready();
```

- Che sono sinonimi. In entrambi casi passo come parametro una funzione anonima. Cioè l'intero blocco di codice di cui è composta la funzione preceduta da **function():**

```
$(function(){//blocco comandi});
```

LO STILE DI SCRITTURA

- Ma posso usare anche una funzione con nome:

```
function documentoPronto() {
  //corpo funzione
}
```

- E passare come parametro il nome della funzione:

```
$(document).ready(documentoPronto);
```

jQuery
write less, do more.

DIFFERENZE

- L'evento **windows.onload** viene sparato quando l'intero documento html è stato caricato, comprese le immagini.
- L'evento **.ready()** viene sparato quando il DOM è caricato (la sola struttura del documento).

jQuery
write less, do more.

DIFFERENZE

- **windows.onload** è una proprietà:

```
windows.onload = function() { //comandi}
```

- **.ready()** è un metodo:

```
$(document).ready(function() { //comandi});
```

jQuery
write less, do more.

DIFFERENZE

- Con l'evento **windows.onload** **assegno un'unica** funzione all'evento.
- Con l'evento **.ready()** **aggiungo** una funzione all'evento.

IL DOLLARO $

- la funzione $() (che sostituisce per concisione jQuery()) è la funzione principale. Può avere varie combinazioni di parametri:
  - **`$(funzione);`**
  - **`$(elemento);`**
  - **`$(selettore css);`**
  - **`$(selettore css, contesto);`**
  - **`$(codice html);`**

RICERCA DI ELEMENTI

- Il modo più classo di procedere di jQuery è quello di "selezionare alcuni elementi ed eseguire azioni su di essi."
- La selezione avviene passando alla funzione $():
  - Un stringa che rappresenta un selettore CSS
  - Un elemento del DOM (ad esempio document o window)

jQuery
write less, do more.

ESEMPI

- Selezione degli elementi in base alla loro ID
  - `$('#myid') / / L'ID deve essere univoco`
- Selezione degli elementi in base al nome della classe
  - `$('div.myClass')`
- Selezione degli elementi in base a un attributo
  - `$ ( 'input [name = first_name]' )`
- Selezionare gli elementi in base a un selettore CSS
  - `$ ( '# contenuti ul.people li' );`

PSEUDO-SELETTORI

- `$('a.external:first');` *// primo elemento <a>*
  // con classe 'external'
- `$('tr:odd');` *// elementi <tr>* dispari *in*
  //una tabella
- `$('#myForm :text');` *// tutti gli elementi input di*
  // tipo text in #myForm
- `$('div:visible');` *// tutte le div visibli*
- `$('div:gt(2)');` *// selecciona tutte le div eccetto*
  // le prime tre
- `$('div:animated');` *// tutte le div animate*

RISULTATO

- La funzione $() restituisce un oggetto di tipo jQuery.
- L'oggetto jQuery può rappresentare il set degli elementi trovati o un unico elemento.
- Per controllare se la ricerca ha prodotti risultato deve controllare la proprietà length dell'oggetto jQuery restituito.

```
if ($('div.foo').length > 0) { ... }
```

jQuery PSEUDO SELETTORI FORM

- :button Seleziona elementi <input> con l'attributo type='button'
- :checkbox Seleziona elementi <input> con l'attributo type='checkbox'
- :checked Seleziona elementi <input> selezionati
- :disabled Seleziona elementi disabilitati
- :enabled Seleziona elementi abilitati
- :file Seleziona elementi <input> con type='file'
- :image Seleziona elementi <input> con type='image'
- :input Seleziona elementi <input>, <textarea> y <select>
- :password Seleziona elementi <input> con type='password'
- :radio Seleziona elementi <input> con type='radio'
- :reset Seleziona elementi <input> con type='reset'
- :selected Seleziona elementi <options> selezionati
- :submit Seleziona elementi <input> con type='submit'
- :text Seleziona elementi <input> con type='text'

LAVORARE CON LE SELEZIONI

- Una volta ottenuto un set di componenti in base alla selezione, si possono utilizzare i metodi dell'oggetto jQuery.
- Gli oggetti jQuery non hanno proprietà direttamente accessibili escluso length
- I metodi si dividono in due categorie: getter e setter.
  - i metodi getter restituiscono una proprietà dell'elemento selezionato,
  - i metodi setter di impostano una proprietà di tutti gli elementi del set restituito.

CHAINING

- Ogni metodo jQuery restituisce l'oggetto jQuery su cui il metodo ha operato. Questo rende possibilie il concatenamento tipico della scrittura javascript di jQuery:

```
$('#content')
  .find('h3')
  .eq(2)
  .html('nuovo testo per il terzo h3');
```

GETTERS e SETTERS

- I metodi per impostare un valore hanno lo stesso nome dei metodi per ottenere un valore.
- Ciò che differenzia il metodo setter dal corrispondente getter è il parametro in più costituito dal valore a da impostare:

```
$ ('H1').html( 'ciao mondo' );
$ ('H1').html();
```

STILI CSS

- Per ottenere o modificare lo stile css di un elemento (o di un set di elementi) ho il metodo .css:
- Getter;

```
$ ( 'H1' ). css ( 'fontSize' )        / / restituisce"19px"
$ ( 'H1' ). css ( 'font-size' )       / / funziona
```

- Setter

```
$('h1').css('fontSize', '100px');
$('h1').css({ 'fontSize' : '100px', 'color' : 'red' });
```

CLASSI CSS

- Anche se molto utile, il metodo .css non dovrebbe essere usato per applicare direttamente stili agli elementi (si può fare direttamente da CSS). È meglio usare CSS per definire classi e applicare queste agli elementi a secondo delle nostre necessità.

```
var $h1 = $('h1');
$h1.addClass('big');
$h1.removeClass('big');
$h1.toggleClass('big');
if ($h1.hasClass('big')) { ... };
```

DIMENSIONI

- jQuery offre una varietà di metodi per ottenere e impostare le dimensioni e la posizione di un elemento.

```
$('h1').width('50px');    // imposta la larghezza di tutti gli elementi H1
$('h1').width();          // ottiene la larghezza di tutti gli elementi H1
$('h1').height('50px');   // imposta l'altezza di tutti gli elementi H1
$('h1').height();          // ottienel'altezza di tutti gli elementi H1
$('h1').position();   // restituisce un oggetto contenente
                      // informazioni sulla posizione
                      // del primo elemento H1 relativo all'offsett
                      // del suo elemento padre
```

ATTRIBUTI

- Il metodo attr() ha una sintassi simile a css ma ottiene e imposta gli attributi di un elemento anziché lo stile:
- Setter

```
$('a').attr('href', 'hrefTuttiUguali.html');
$('a').attr({
                    'title' : 'tutti lo stesso title',
                    'href' : 'laStessaUrl.html'
              });
```

- Getter

```
$('a').attr('href'); //restituisce href del primo elemento
```

ALTRI GETTER-SETTER

- .html()
  Ottiene o imposta il contenuto di un elemento HTML.
- .text()
  Ottiene o imposta il contenuto testuale dell'elemento, nel caso un testo con tag HTML, questi viene rimossi.
- .val()
  Ottiene o imposta il valore (*value*) di elementi di un form.

jQuery CREARE NUOVI ELEMENTI

- Si può utilizzare $() anche per creare un nuovo oggetto jQuery (quindi un nuovo elemento)
  - Stringa HTML

```
$('<p>Un nuovo paragrafo</p>');
$('<li class="new">nuovo elemento della lista</li>');
```

  - Stringa HTML + oggetto che rappresenta gli attributi

```
$('<a/>', {
  html : 'Un <strong>nuovo</strong> enlace',
  'class' : 'new',
  href : 'foo.html'
});
```

INSERIRE, SPOSTARE...

- Ci sono diversi modi per collocare gli elementi nel DOM:
  - Posizionare nuovi elementi o selezioni rispetto ad un altro elemento
  - Inserire nuovi elementi o selezioni come figli di un altro elemento

```
var $myNewElement = $('<p>Nuovo elemento</p>');
$myNewElement.appendTo('#content');
$myNewElement.insertAfter('ul:last');
        // eliminerá l'elemento <p>
        // esistente in #content
$('ul').last().after($myNewElement.clone());
        // copia l'elemento <p>
        // e lo duplica
```

EVENTI

- Contrariamente a javascript gli eventi si gestiscono con metodi non con proprietà

```
$('p').click(function() {
    alert('click');
});
```

ON

- *$(selector)*.on("evento1 evento2", funzione)
  1° parametro: Uno o più tipi di eventi separati da uno spazio.

  funzione: Una funzione che viene associata all'evento per ciascuno elemento del set

Evento	Descrizione
blur	Si verifica quando l'elemento perde il focus
change	Si verifica quando l'elemento cambia valore
click	Si verifica quando fai clic colmouse
dblclick	Si verifica quando un doppio clic del mouse
error	Si verifica quando c'è un errore
focus	Si verifica quando l'elemento riceve il focus
keydown	Si verifica quando si preme un tasto
keypress	Si verifica quando il tasto viene premuto e rilasciato
keyup	Si verifica quando il tasto viene rilasciato
load	Si verifica quando il documento viene caricato
mousedown	Si verifica quando si preme il pulsante del mouse
mouseenter	Si verifica quando il puntatore del mouse entra nell'area dell'elemento
mouseleave	Si verifica quando il puntatore del mouse esce dall'area dell'elemento
mousemove	Si verifica quando il puntatore del mouse si sposta
mouseout	Si verifica quando il puntatore del mouse viene spostato all'esterno di un elemento
mouseover	Si verifica quando il puntatore del mouse viene spostato sopra un elemento
mouseup	Si verifica quando il pulsante del mouse viene rilasciato
resize	Si verifica quando la finestra viene ridimensionata
scroll	Si verifica quando un elemento viene fatto scorrere con la barra di scorrimento
select	Si verifica quando un testo viene selezionato
submit	Si verifica quando un form viene inviato
unload	Si verifica quando il documenti viene abbandonato