LEZIONE 6

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;
}
```

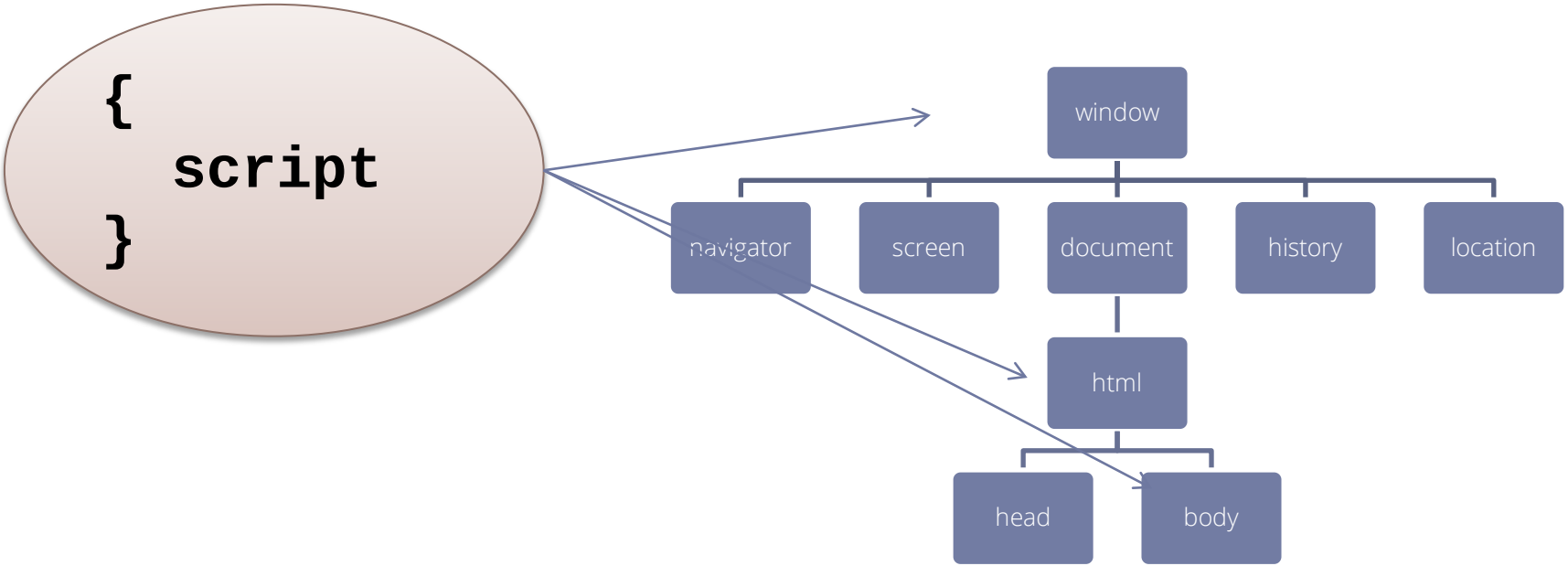
funzione anonima

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;
```

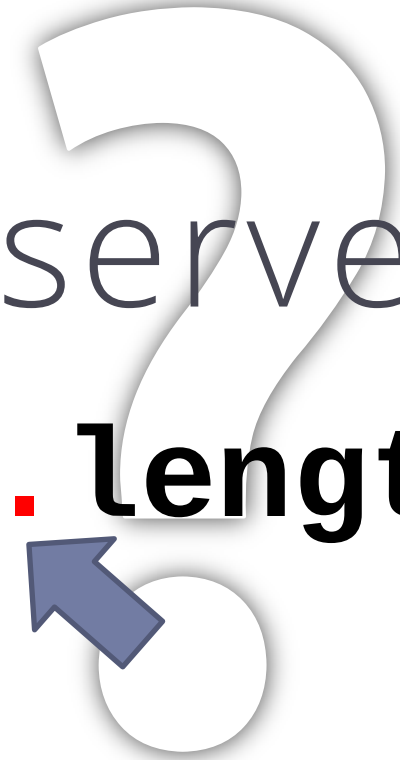
A cosa serve il ";"?

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

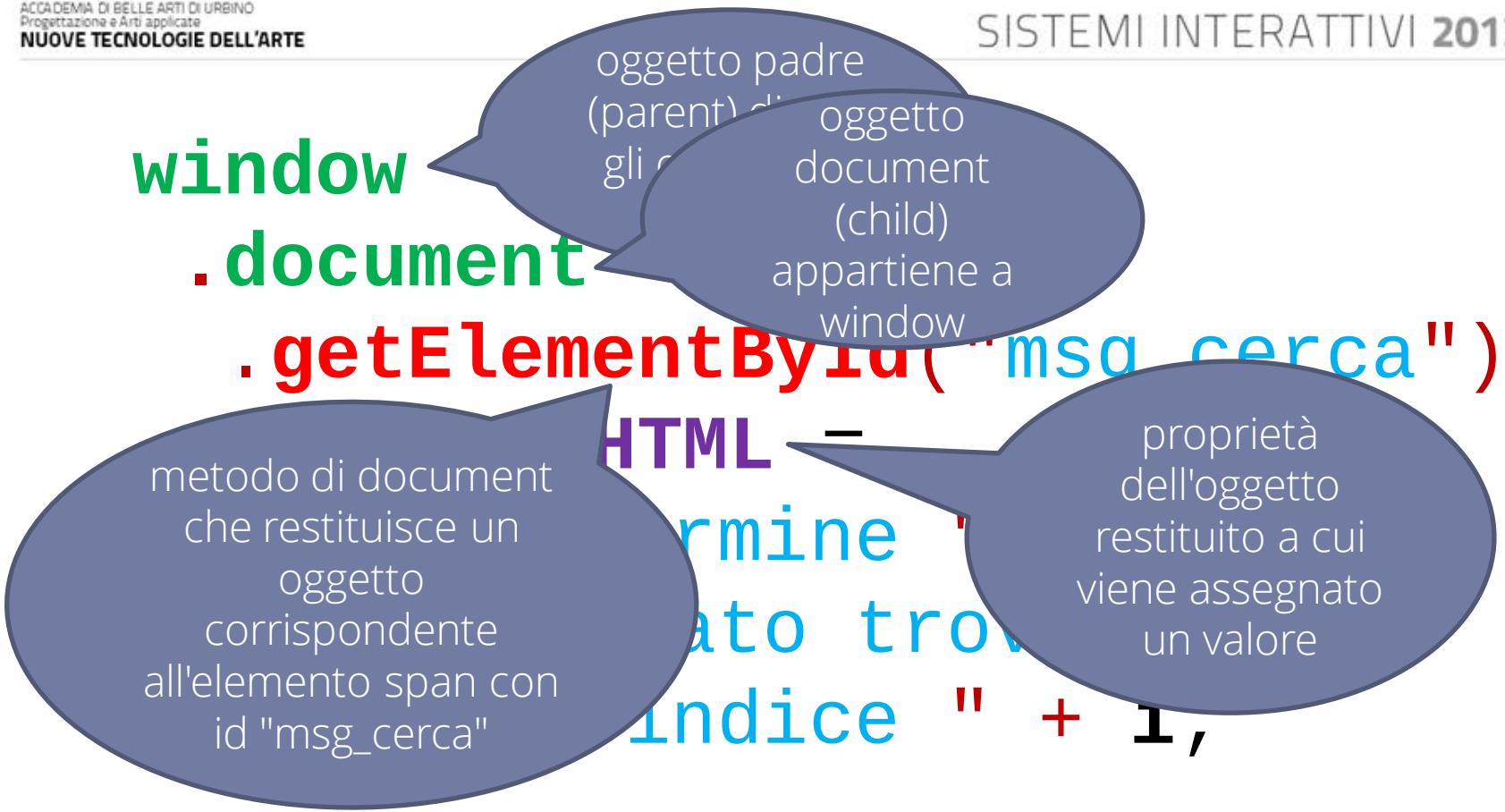
- Il punto e virgole è il segno che separa le istruzioni fra loro.
- Le istruzioni NON sono separate dalla fine riga.

A cosa serve il "."?

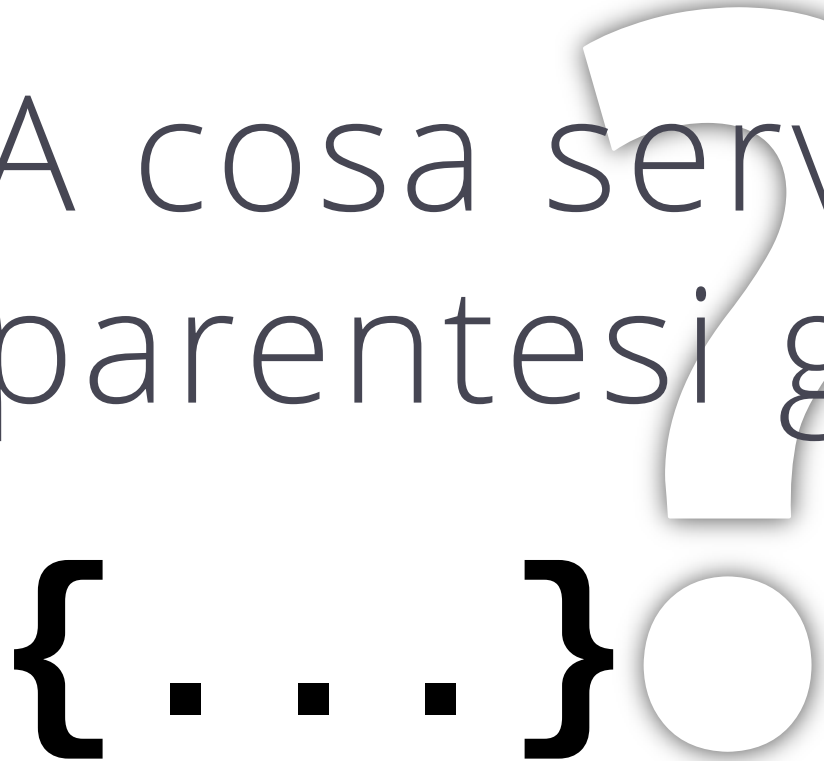
```
if (str.length < 2)
```



- 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.

OBJECT

- Object è una grandezza informatica in grado di rappresentare elementi complessi.
- In Javascript tutte le grandezze si rappresentano tramite oggetti. Esistono oggetti di base definiti dal linguaggio:
  - Number
  - String
  - Date
  - Array
  - Boolean
  - Math
  - RegExp
- E oggetti che servono a rappresentare I dati del mondo reale.

ESEMPIO

```
var user:Object = new Object();
user.name = "Irving";
user.age = 32;
user.phone = "555-1234";
```

Viene creato un nuovo oggetto denominato user e tre proprietà: name, age e phone che sono tipi di dati String e Numeric.
Lo stesso oggetto può essere creato anche assegnando alla variabile il letterale di tipo *Object* corrispondente.

```
var user:Object;
user = {name:”Irving”,age:32,phone:”555-1234”};
```

Quando si assegna ad una variabile un valore in formato letterale non è necessario richiamare il costruttore della classe con l'operatore *new*. Queato vale sia per *Object* che per *Array.*

PROPRIETÀ E METODI

- Ognuno degli oggetti cha abbiamo visto ha:
- Proprietà che ci consentono di leggere o modificare determinate caratteristiche di un elemento
- Metodi che ci mettono a disposizione determinate **azioni** che gli oggetti possono compiere

Rappresentazione del DOM

- Ogni elemento del DOM è rappresentato come Object
- L'accesso alle proprietà e ai metodi avviene attraverso l'oiperatore di appartenenza (.)
- Se, per esempio, voglio recuperate il riferimento ad un oggetto scrivo:

```
window.document.getElementById('id')
```

PROPRIETÀ COMUNI

- Tutti le classi hanno in comune due proprietà:
  - **constructor**: contiene la funzione utilizzata quando si crea una nuova istanza della classe.
  - **prototype**: oggetto che contiene tutte le proprietà e i metodi che avrà la nuova istanza creata.

EVENTI

- Grazie agli eventi possiamo "impacchettare" il codice scritto attraverso JavaScript e farlo eseguire non appena l'utente esegue una data azione:
  - quando clicca su un bottone di un form possiamo controllare che i dati siano nel formato giusto;
  - quando passa su un determinato link possiamo
  - Quanto è completato il caricamento di una immagine
  - eccetera....

EVENTI

evento	si applica agli elementi di tipo...	esempio
onload	<body>, <img>	document.getElementById("imageId").onload = ...
onunload	<body>	document.onunload = ...
onmouseover	tutti gli elementi che occupano uno spazio	document.getElementById("Id"). onmouseover = ...
onmouseout	tutti gli elementi che occupano uno spazio	document.getElementById("Id"). onmouseout = ...
onclick	tutti gli elementi che occupano uno spazio	document.getElementById("Id"). onclick = ...
onkeypress	<a>, <area>, <input>,	document.getElementById("Id"). onkeypress = ...
onchange	<select>	document.getElementById("selectId"). onchange = ...
onsubmit	<form>	document.getElementById("formId"). onsubmit = ...
onfocus	<a>, <input>, <body>, <textarea>, <button>	document.getElementById("Id"). onfocus = ...
onblur	<a>, <input>, <body>, <textarea>, <button>	<document.getElementById("Id"). onblur = ...

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

LE TABELLE DI VERITÀ

- Prendiamo questi enunciati:
  - esco se il tempo è bello ed è caldo
  - esco se il tempo è bello o è caldo
  - non esco se il tempo non è bello e non è caldo
  - non esco se il tempo non è bello o non è caldo

LE TABELLE DI VERITÀ

– esco se il tempo è bello ed è caldo

enunciato 1	congiunzione	enunciato 2	risultato
tempo è bello	ed	temperatura è caldo	esco
true	and	true	true
false	and	true	false
true	and	false	false
false	and	fasle	false

LE TABELLE DI VERITÀ

– esco se il tempo è bello o è caldo

enunciato 1	congiunzione	enunciato 2	risultato
tempo è bello	o	temperatura è caldo	esco
true	or	true	true
false	or	true	true
true	or	false	true
false	or	fasle	false

LE TABELLE DI VERITÀ

– non esco se il tempo non è bello e non è caldo

	enunciato 1	congiunzione	enunciato 2	risultato
non	tempo è bello	e	temperatura è caldo	non esco
not	true	and	true	false
not	true	and	false	false
not	false	and	true	false
not	false	and	false	true

le tabelle di verità

- non esco se il tempo non è bello o non è caldo

	enunciato 1	congiunzione	enunciato 2	risultato
non	tempo è bello	o	temperatura è caldo	non esco
not	true	or	true	false
not	true	or	false	true
not	false	or	true	true
not	false	or	false	true

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
```

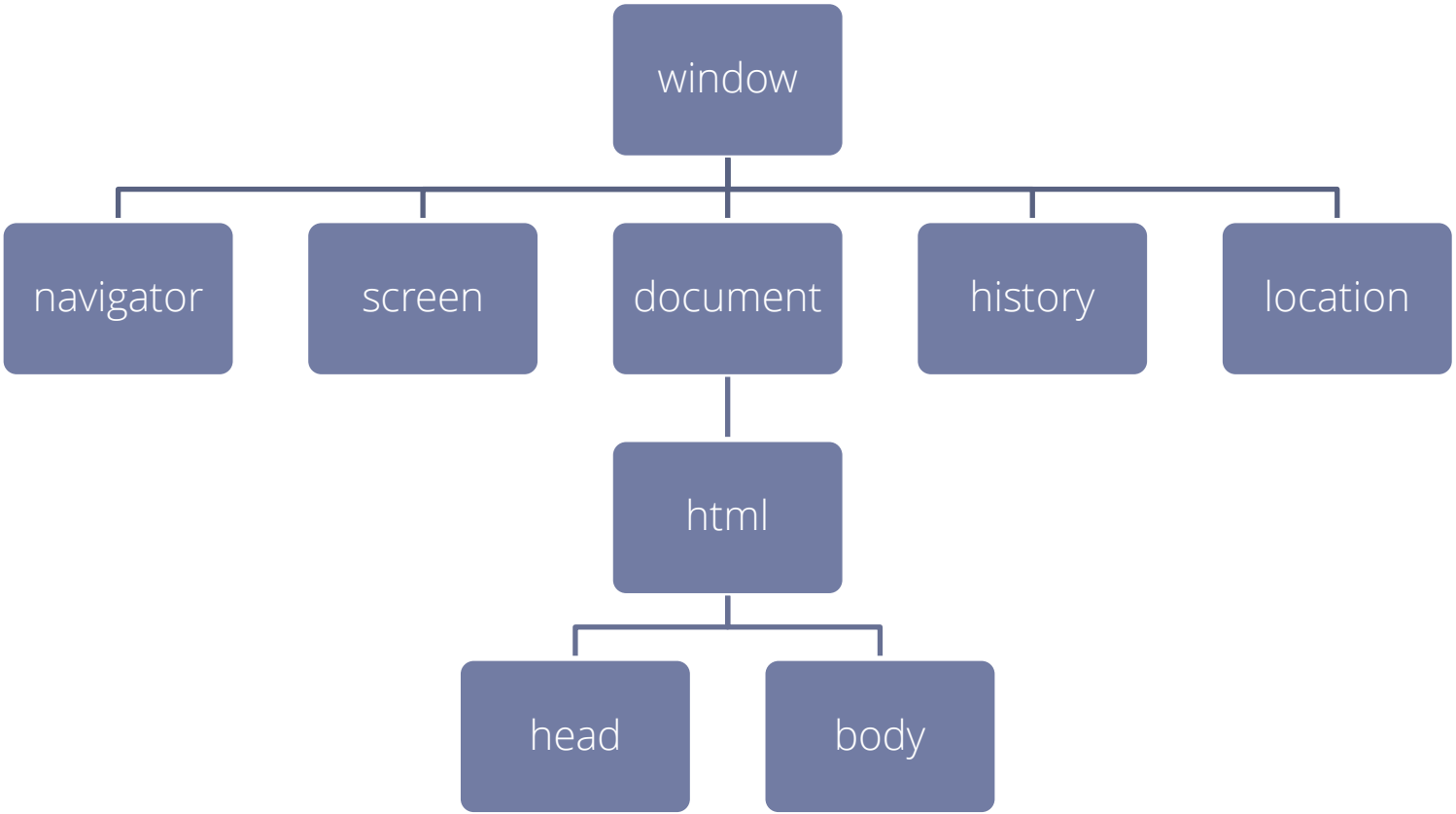
esempio

```
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.";
}
```

JAVASCRIPT

- Javascript serve per programmare il browser.
- Lo studio di Javascript è strettamente legato allo studio del Document Object Model (DOM)

DOCUMENT OBJECT MODEL


window
navigator
screen
document
history
location
html
head
body

WINDOW

- L'oggetto window è al vertice della gerarchia degli oggetti.
- Rappresenta il la finestra del browser in cui appaiono i documenti HTML. In un ambiente multiframe, anche ogni frame è un oggetto window.
- Dato che ogni azione sul documento si svolge all'interno della finestra, la finestra è il contenitore più esterno della gerarchia di oggetti. I suoi confini fisici contengono il documento.

NAVIGATOR

- L'oggetto navigator rappresente il browser.
- Utilizzando questo oggetto gli script posso accedere alle informazioni sul browser che sta eseguendo il vostro script (marca, versione sistemo operativo).
- E' un oggetto a sola lettura, e il suo uso è limitato per ragioni di sicurezza.

SCREEN

- L'oggetto screen rappresente lo schermo del computer su cui il browser è in esecuzione.
- E' un oggetto a sola lettura che consente allo script conoscere l'ambiente fisico in cui il browser è in esecuzione.
- Ad esempio, questo oggetto fornisce informazioni sulla risoluzione del monitor.

HISTORY

- L'oggetto history rappresenta l'oggetto che in memoria tiene traccia della navigazione e presiede al funzionamento dei bottoni back e forward e alla cronologia del browser.
- Per ragioni di sicurezza e di privacy gli script non hanno accesso a informazioni dettagliate sulla history e l'oggetto di fatto consente solo di simulare i bottoni back e forward.

LOCATION

- L'oggetto location rappresenta l'url da cui è stata caricata la pagina
- La sua funzione principale è quella di caricare una pagina diversa nella corrente finestra o frame.
- Allo script è consentito di accedere ad informazioni solo sulla url da cui è stato caricato.

DOCUMENT

- Ogni documento HTML che viene caricato in una finestra diventa un oggetto document.
- L'oggetto document contiene il contenuto strutturato della pagina web.
- Tranne che per gli html, head e body, oggetti che si trovano in ogni documento HTML, la precisa struttura gerarchica dell'oggetto document dipende dal contenuto del documento.

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

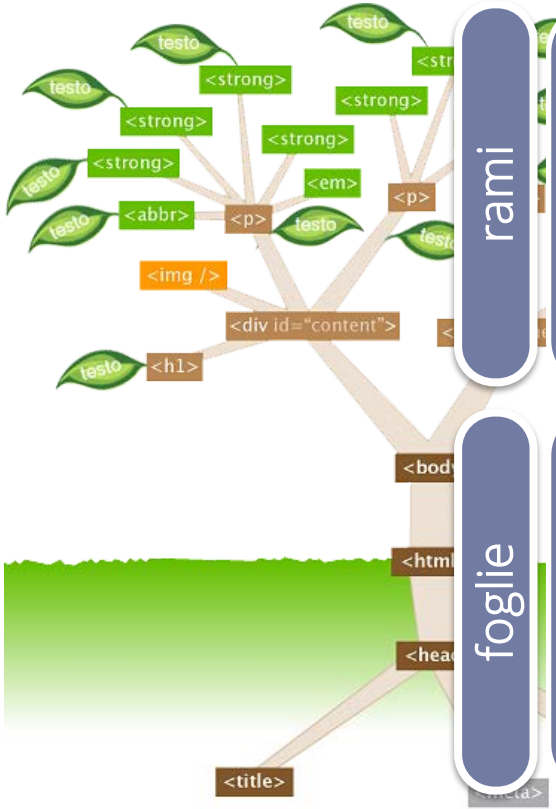
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo 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);
```

RECUPERARE GLI ELEMENTI

- getElementsByTagName(tagName)
  l'insieme degli elementi caratterizzati dallo stesso tag viene restituito in **un array di elementi.** L'array conserva lo stesso ordine con cui i tag corrispondenti compaiono nel codice della pagina.
- La sintassi è:

```
elem_array= document.getElementsByTagName(nomeTag);
```

CREARE NODI ED ELEMENTI

- createElement(tagName)
  Il metodo crea un nuovo elemento di qualunque tipo. Restituisce un riferimento al nuovo elemento creato.
- La sintassi è:

```
nuovo_elemento = document.createElement(nomeTag);
```

CREARE NODI ED ELEMENTI

- createTextNode(text)
  Il metodo crea un nuovo nodo di testo e restituisce il riferimento al nuovo nodo creato.
- La sintassi è:

```
nuovo_testo = document.createTextNode(testo);
nuovo_testo = document.createTextNode("Ciao");
```

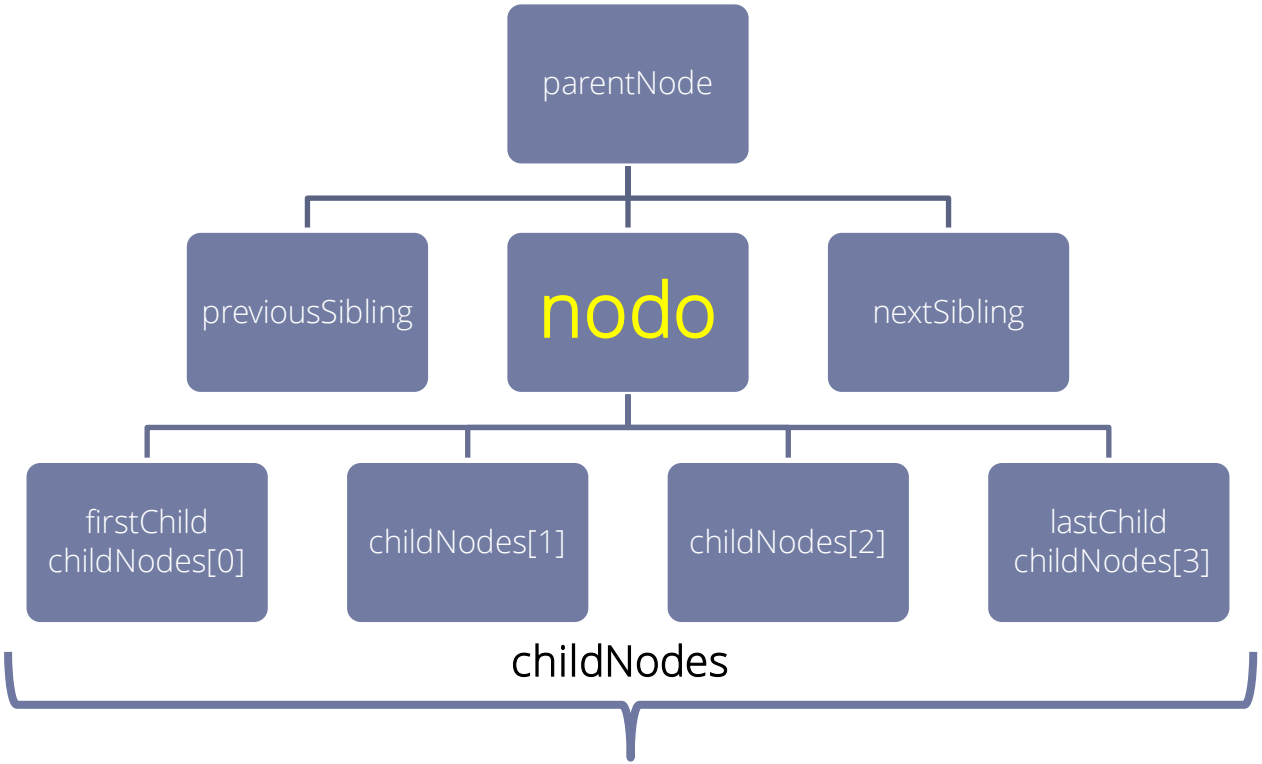
PROPRIETÀ DEI NODI

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



RELAZIONE TRA NODI

- parentNode
  proprietà che restituisce il riferimento al nodo che contiene il nodo corrente. Ogni nodo ha un solo **parentNode**. Quando il nodo non ha padre la proprietà restituisce null.

```
nodoPadre = nodo.parentNode;
```

RELAZIONE TRA NODI

- childNodes
  proprietà che restituisce una nodeList di riferimenti ai nodi che discendono direttamente dal nodo corrente. I nodi sono nello stesso ordine in cui appaiono nella pagina.

```
nodiFigli = nodo.childNodes;
```

RELAZIONE TRA NODI

- firstChild
  proprietà che restituisce il riferimento al primo dei figli che discendono direttamente dal nodo corrente. Corrisponde a childNodes[0].

```
primoFiglio = nodo.firstChild;
```

RELAZIONE TRA NODI

- lastChild
  proprietà che restituisce il riferimento all'ultimo dei figli che discendono dal nodo corrente. Corrisponde a childNodes[childNodes.length - 1].

```
ultimoFiglio = nodo.lastChild;
```

RELAZIONE TRA NODI

- previousSibling
  proprietà che restituisce il riferimento al nodo "fratello" precedente a quello al quale è applicato. Se il nodo non ha "fratelli maggiori", la proprietà restituisce null.

```
nodoFratello = nodo.previousSibling;
```

RELAZIONE TRA NODI

- nextSibling
  proprietà che restituisce il riferimento al nodo "fratello" successivo a quello al quale è applicato. Se il nodo non ha "fratelli minori", la proprietà restituisce null.

```
nodoFratello = nodo.nextSibling;
```

VALORE

- nodeValue
  proprietà che, se applicata ad un **element** (tag) restituisce null, mentre se applicata ad un **textNode** restituisce il testo che contengono. È una proprietà **read/write**.

```
testo = nodoDiTesto.nodeValue;
nodoDiTesto.nodeValue = "Ciao!";
```

METODI APPLICABILI AI NODI

ESISTONO FIGLI?

- hasChildNodes()
  Questo metodo se il nodo contiene altri nodi restituisce true altrimenti false.
- La sintassi è:
  `nodo.hasChildNodes();`

AGGIUNGERE O ELIMINARE FIGLI

- appendChild()
  Il metodo inserisce un nuovo nodo alla fine della lista dei figli del nodo al quale è applicato.
- La sintassi è:
  `nodo.appendChild(nuovoFiglio);`

AGGIUNGERE O ELIMINARE FIGLI

- insertBefore()
  Questo metodo consente di inserire un nuovo nodo nella lista dei figli del nodo al quale è applicato, appena prima di un nodo specificato.
- La sintassi è:
  `nodo.insertBefore(nuovoFiglio);`

AGGIUNGERE O ELIMINARE FIGLI

- replaceChild
  questo metodo consente di inserire un nuovo nodo al posto di un altro nella struttura della pagina.
- La sintassi è:

```
nodo.replaceChild(nuovoFiglio, vecchioFiglio);
```

aggiungere o eliminare figli

- removeChild
  il metodo elimina e restituisce il nodo specificato dalla lista dei figli del nodo al quale è applicato.
- La sintassi è:

```
figlioRimosso = nodo.removeChild(figlioDaRimuovere);
```

Copiare un nodo

- cloneNode
  il metodo restituisce una copia del nodo a cui è applicato, offrendo la possibilità di scegliere se duplicare il singolo nodo, o anche tutti i suoi figli.
- La sintassi è:

```
copia = nodo.cloneNode(copiaFigli);
```

ELEMENTS

ELABORARE GLI ELEMENTI

- tagName
  È la proprietà che restituisce il nome del tag dell'elemento a cui è applicata.
- Sintassi:

```
nome_tag = elemento.tagName;
```

ELABORARE GLI ELEMENTI

- attributes
  È la proprietà che restituisce l'elenco degli attributi diun determinato elemento. La lista è un oggetto di tipo NamedNodeMap che è una collezione di oggetti Attr.
- Esempi:

```
attributi = elemento.attributes;
classeElemento = attributi["class"].value;
```

ATTRIBUTI

- setAttribute, getAttribute e removeAttribute
  Questi tre metodi se applicati a un elemento rispettivamente creano o impostano, leggono ed eliminano un attributo dell'elemento stesso.
- Se elemento è una varibile che contiene il riferimento ad un elemento avrò:

```
elemento.setAttribute(nome_attributo, valore_attributo);
valore_attributo = elemento.getAttribute(nome_attributo);
elemento.removeAttribute(nome_attributo);
```

VALORI E RIFERIMENTI

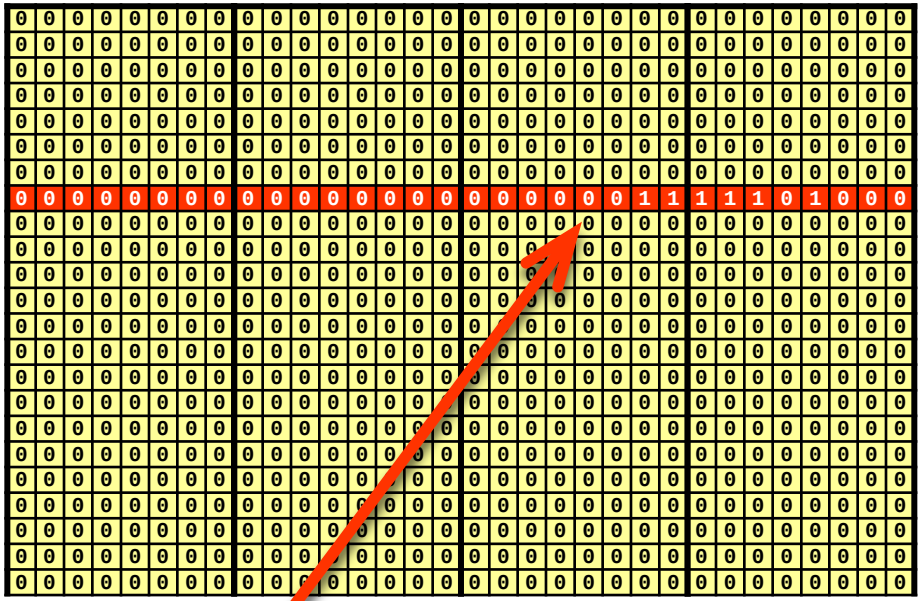
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

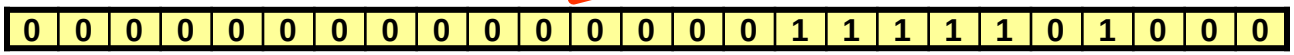


```
var a = 1000;
```

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");
```

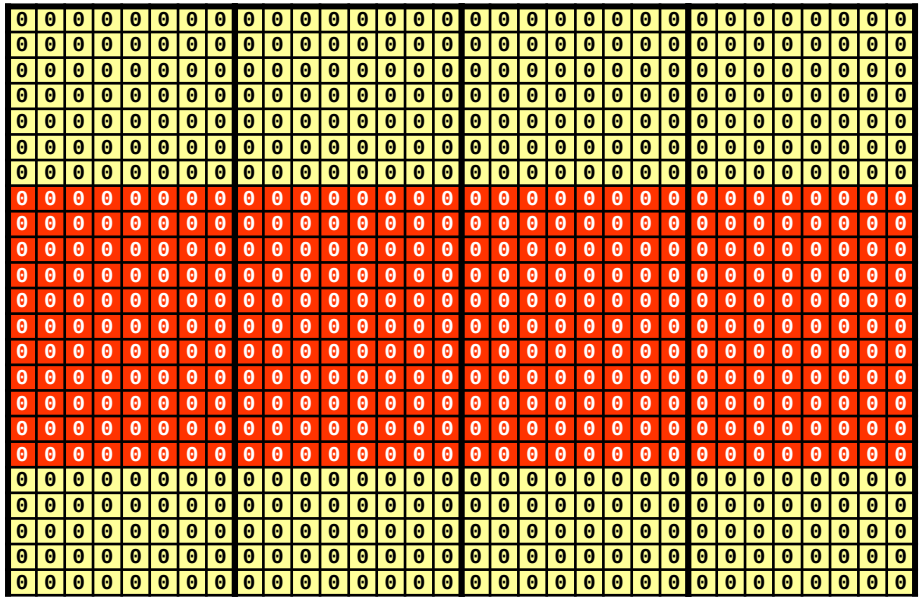
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

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>
```

Finestre di dialogo

- L'oggetto window ci fornisce, tre metodi che ci consentono di fornire o di chiedere informazioni all'utente utilizzando delle finestre di dialogo:

Metodo	Spiegazione	Esempio
alert	Presenta un messaggio all'utente e mostra il bottone Ok	window.alert("messaggio");
confirm	Richiede una conferma all'utente. Mostra i bottoni Ok e Annulla	var risposta; risposta = window.confirm("Vuoi continuare?");
prompt	Richiede all'utente di inserire un valore. Mostra un campo di testo e il bottone Ok	var nome; nome = window.prompt('Come ti chiami?', 'Inserisci qui il tuo nome');