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

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
```

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

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

ESEMPIO

```
var btnGeneraNumero = document.getElementById("btn_genera_numero");
var msgGeneraNumero = document.getElementById("msg_genera_numero");

btnGeneraNumero.onclick = function() {
    var casuale = Math.random() * 100;
    casuale = Math.round(casuale);
    var msg ="";
    if (casuale <= 20) {
        msg="Il numero generato cade nel primo intervallo.";
    } else if ((casuale > 20) && (casuale <= 50)) {
        msg ="Il numero generato cade nel secondo intervallo";
    } else if ((casuale > 50) && (casuale <= 70)) {
        msg = "Il numero generato cade nel terzo intervallo.";
    } else {
        msg = "Il numero generato cade nel quarto intervallo.";
    }
    msgGeneraNumero.innerHTML += msg + "<br>";
};
```

ESEMPIO

```
/**
 * Funzione che formatta ore minuti e secondi
*/
function zeroPrima(n)
{
   //converto n in stringa concatenandolo a str
   var str = "";
   str = n;
   // se la lunghezza della stringa n è minore di 2
   // aggiungo uno 0 in testa
   if (n < 10){
     str = "0" + str;
   }
   return str;
}
```

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.

while, do e for

- In **javascript** usano tre costrutti per ottenere l'iterazione:
  - while
  - do
  - for
- La funzionalità è la stessa con modalità leggermente diverse.

while

- L'istruzione while viene schematizzata come segue:

```
while ( condizione )
      blocco istruzioni;
```

- Con questa istruzione viene prima valutata l'espressione <**condizione**>, se l'espressione risulta vera viene eseguito <**blocco istruzioni**> e si ritorna a controllare la condizione while, altrimenti si esce dal ciclo e si procede con il resto del programma.

do

- L'istruzione do può essere considerato una variante dell'istruzione while ed è strutturata nella maniera seguente:

```
do
 blocco istruzioni
 while ( condizione )
```

- Prima di tutto viene eseguito il blocco di istruzioni racchiusa tra do e while (quindi si esegue almeno una volta), poi si verifica il risultato di <**condizione**>, se è vero si riesegue il blocco che segue il do, altrimenti si continua con l'esecuzione del resto del programma.

for

valore iniziale della variabile

controllo del valore della variabile

a utilizzato per generare max

nu eri ca

```
var i = 0;
while (i < max) {
    listaNumeri += i + ": " + Math.random() + "<br />";
    i++;
}
```

modifica del valore della variabile

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.

for

- (...) per generare max

valore iniziale della variabile

controllo della variabile

modifica del valore della variabile

```
for (i = 0; i < max; i++) {
    listaNumeri += i + ": " + Math.random() + "<br />";
}
```

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.";
}
```

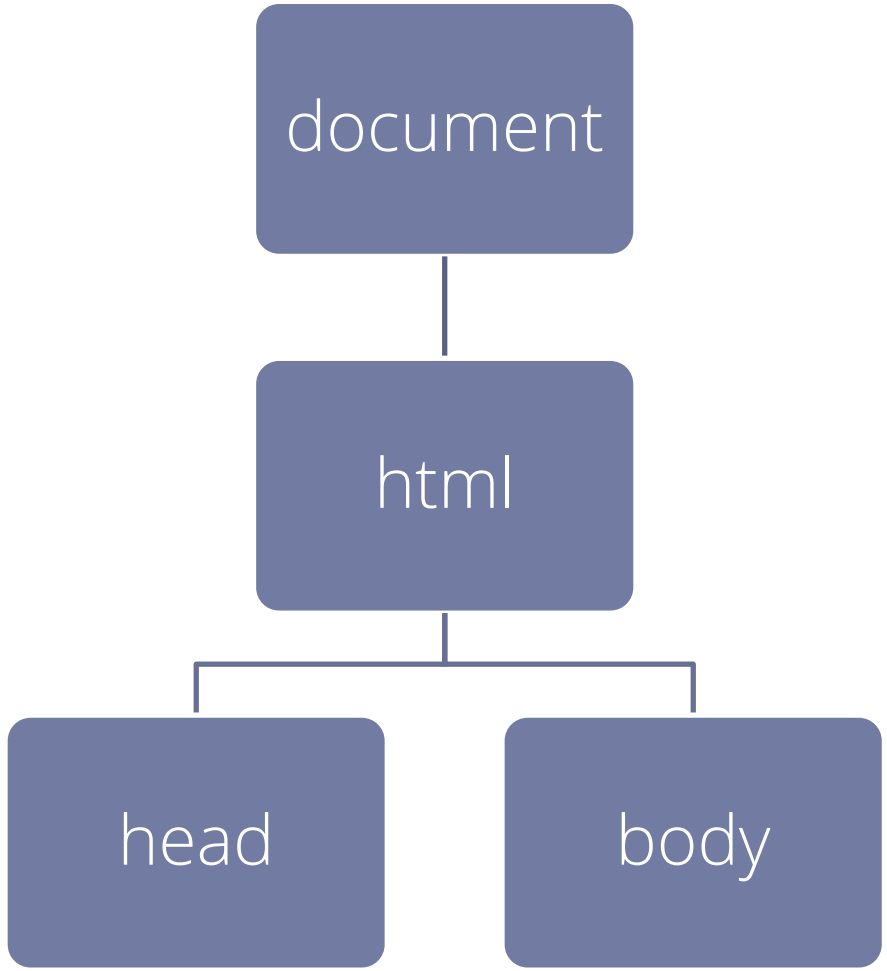
DOCUMENT OBJECT MODEL

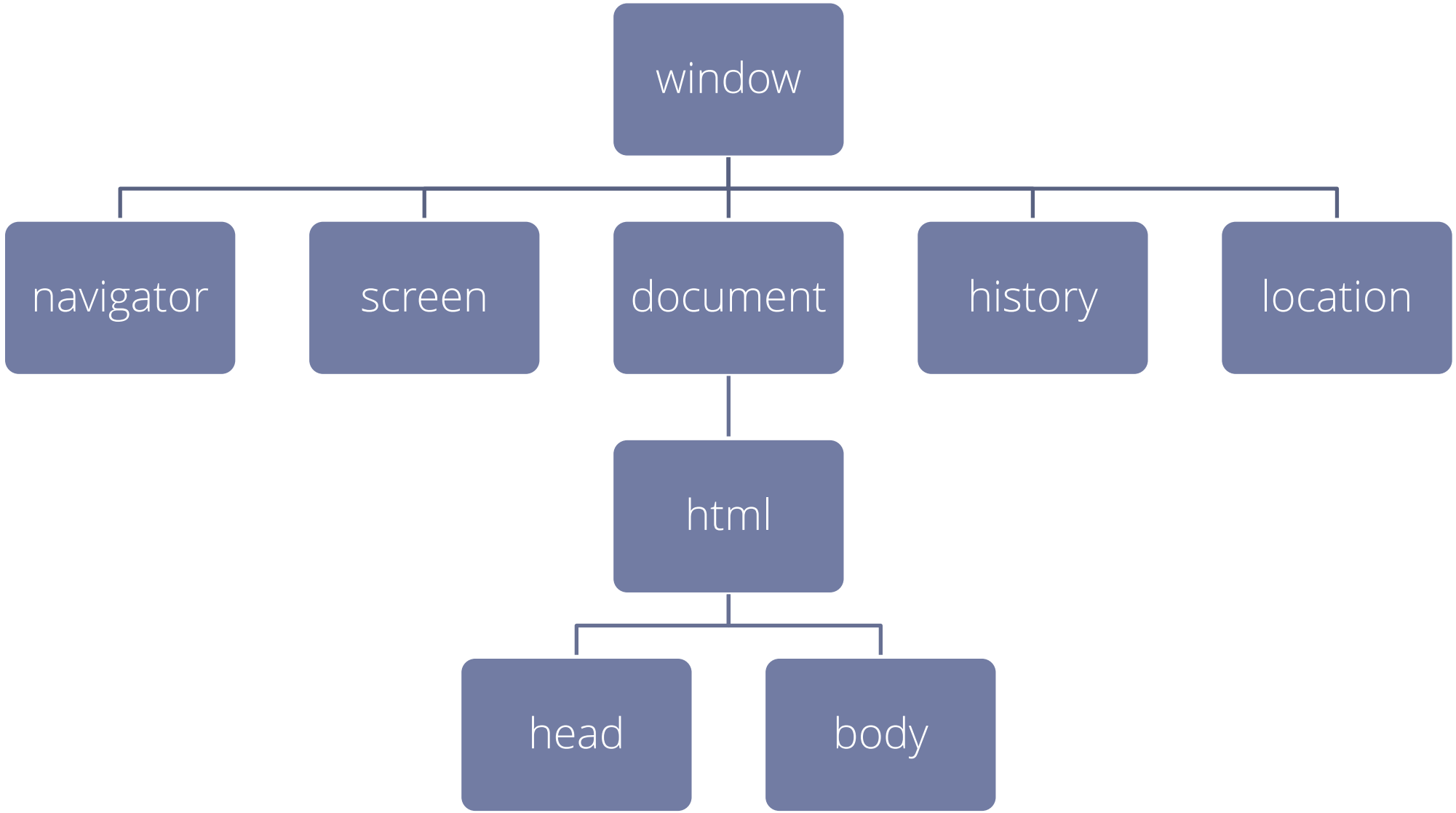
DOM

- HTML (e XHTML) hanno la funzione di strutturare in una rigida gerarchia i contenuti di una pagina WEB
- Quando i browser moderni caricano il contenuto di una pagina organizzano quindi questi contenuti in memoria in una struttura gerarchica ben definita
- Questa struttura gerarchica è il Document Object Model.
- Javascript consente di intervenire su questa struttura aggiungendo, togliendo o modificando gli elementi di cui è composta.

STRUTTURA MINIMA DI UNA PAGINA HTML

```
<html>
  <head></head>
  <body></body>
</html>
```




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

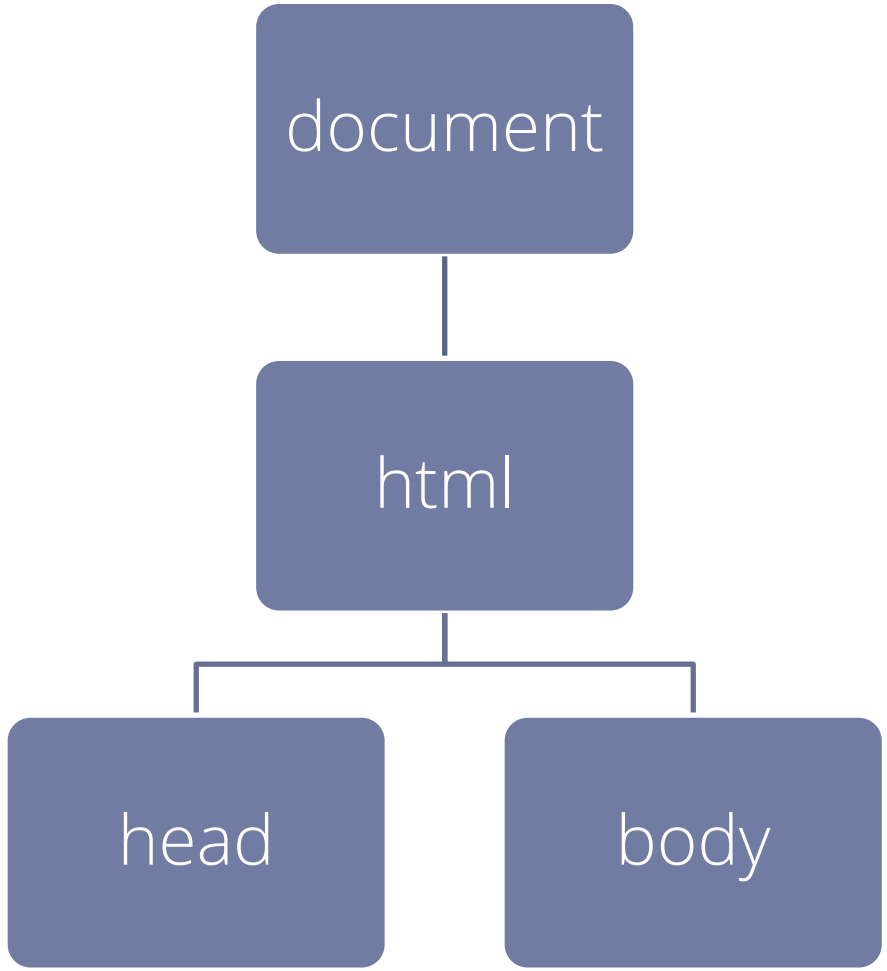
- L'oggetto location rappresenta l'url da cui è stata caricata la pagina
- La sua funzione principale è quella di caricare una pagina diversa nella corrente finestra o frame.
- Allo script è consentito di accedere ad informazioni solo sulla url da cui è stato caricato.

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.

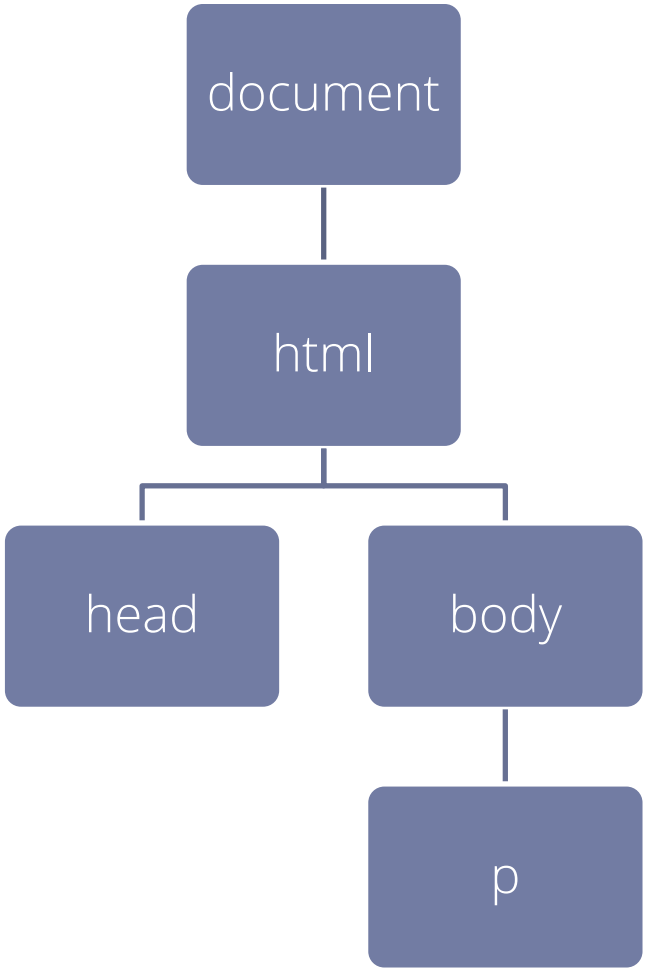
Documento vuoto

```
<html>
  <head></head>
  <body></body>
</html>
```



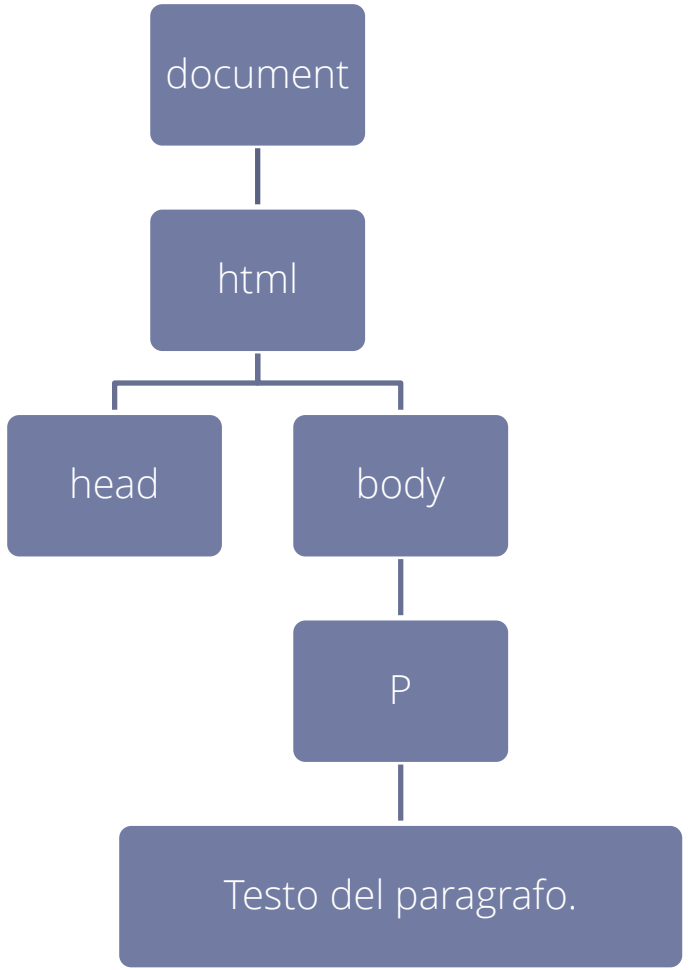
Aggiunta di un paragrafo vuoto

```
<html>
  <head></head>
  <body>
    <p></p>
  </body>
</html>
```



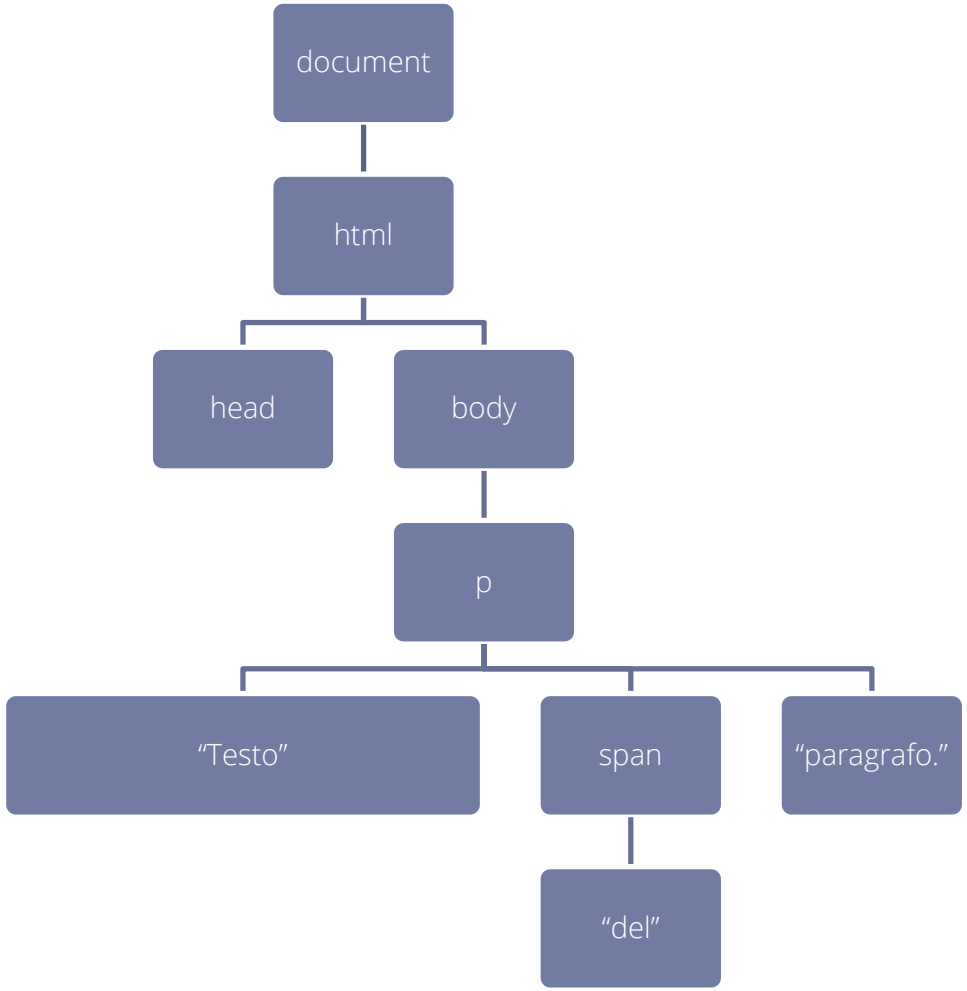
Aggiunta di testo al paragrafo

```
<html>
  <head></head>
  <body>
    <p>Testo del
    paragrafo.</p>
  </body>
</html>
```



Aggiunta di un elemento

```
<html>
  <head></head>
  <body>
    <p>Testo
    <span>del</span>
    paragrafo.</p>
  </body>
</html>
```



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

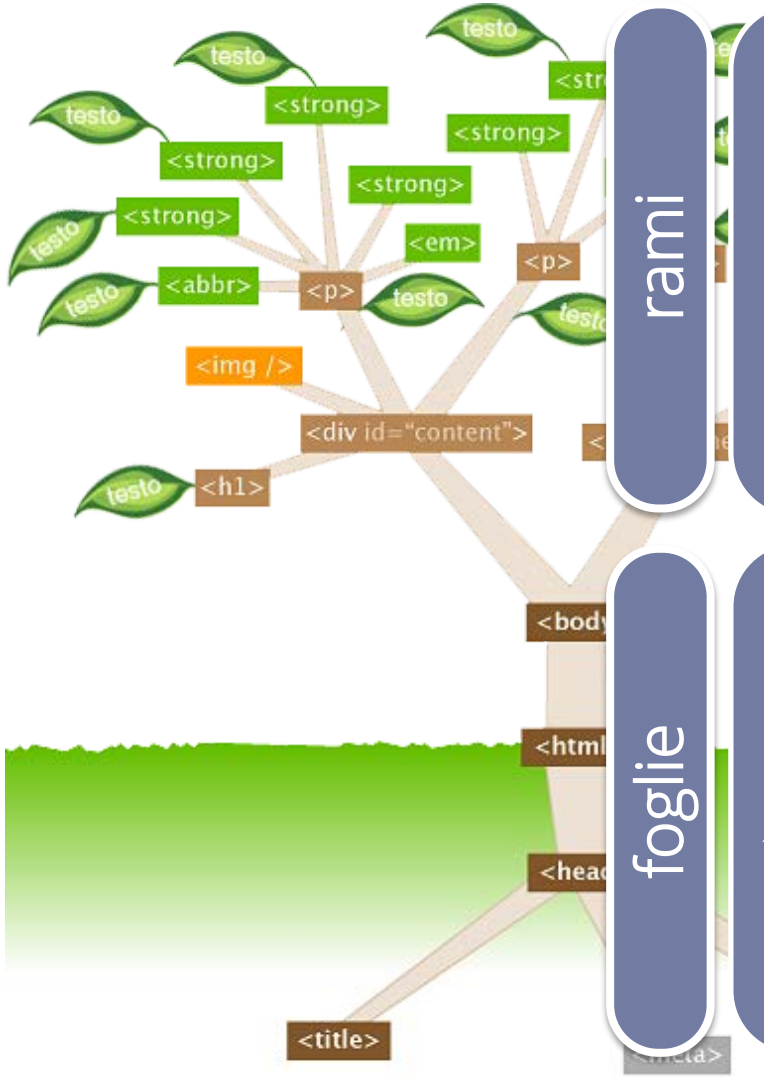
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo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



rami

element sono i nodi che corrispondono ai tag HTML

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

foglie

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);
```

RECUPERARE GLI ELEMENTI

- getElementsByClassName(nomeClasse) 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.getElementsByClassName(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");
```

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;
```

ELABORARE GLI ELEMENTI

- 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;
```

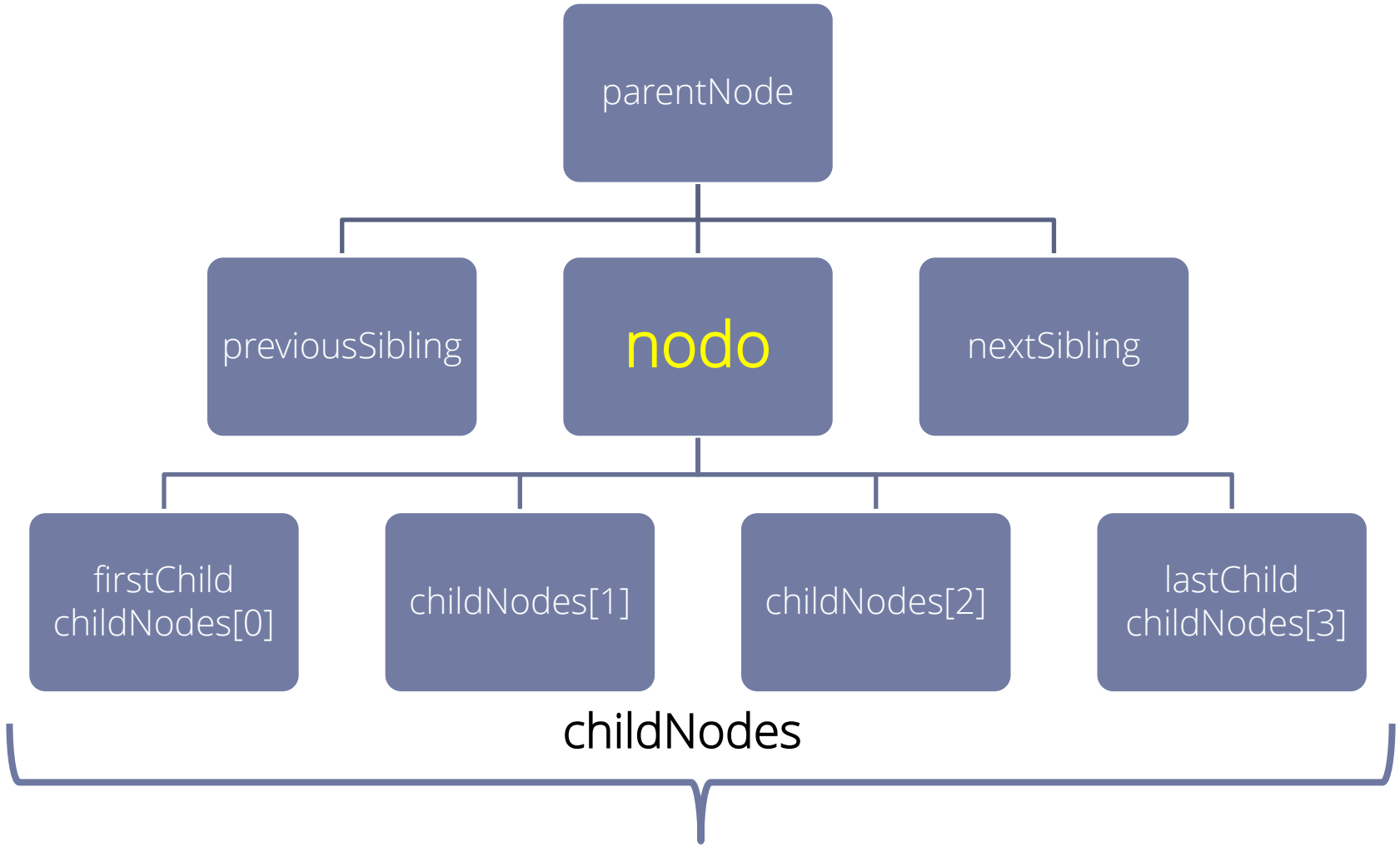
ATTRIBUTI

- 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);
```

PROPRIETÀ DEI NODI

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);
```

VALORI E RIFERIMENTI

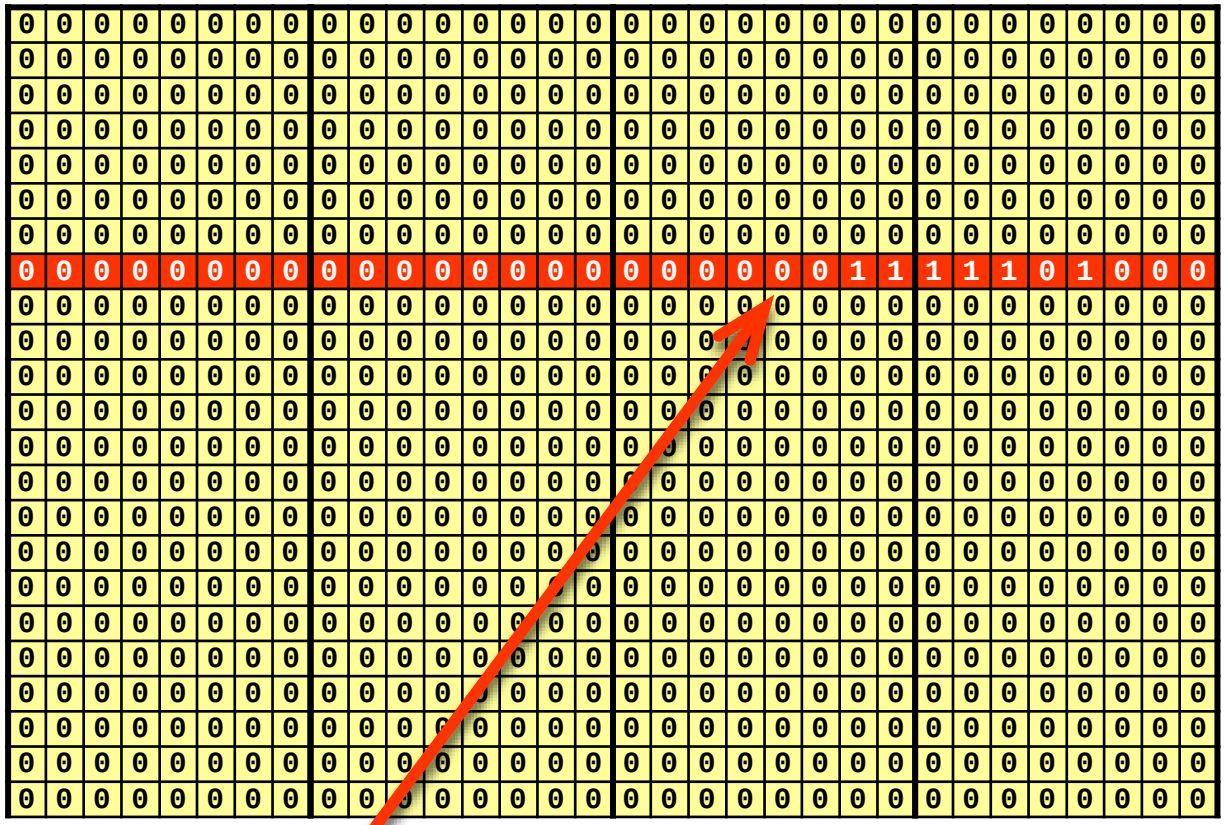
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

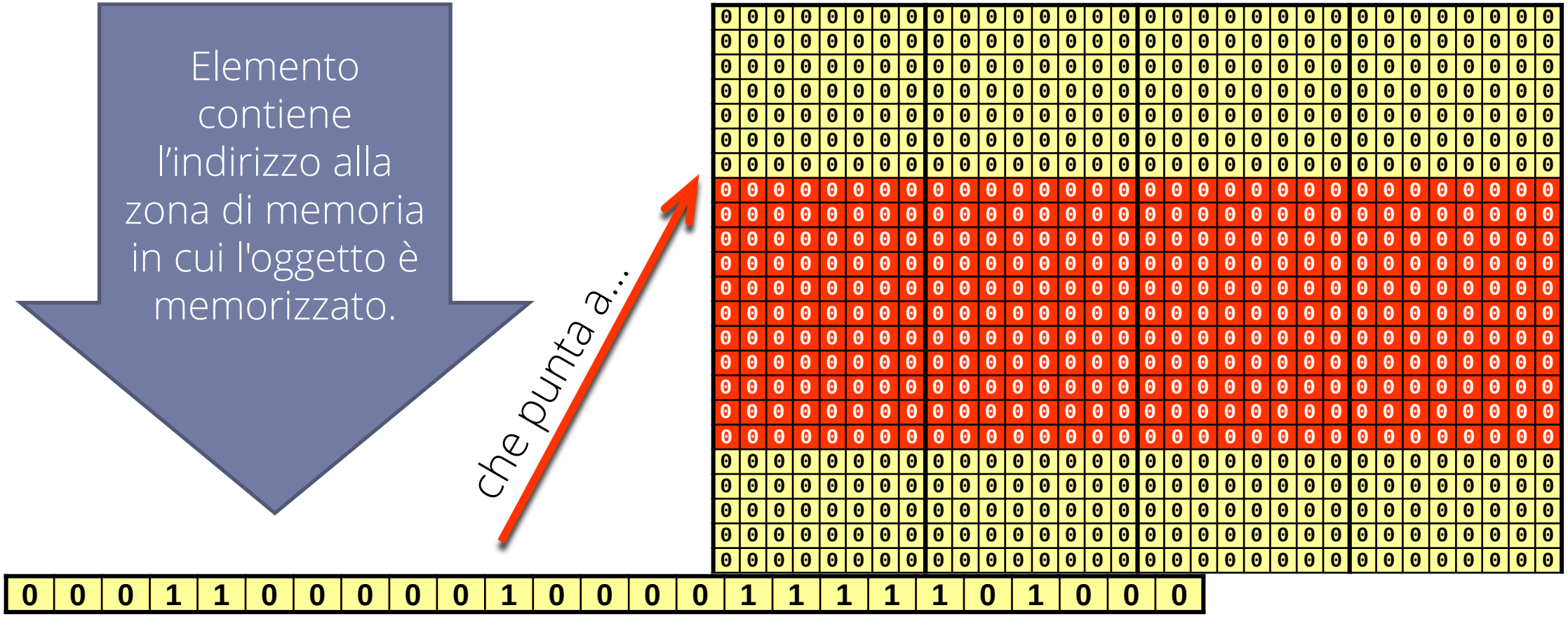
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

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.