LEZIONE 7

VARIABILI

VARIABILI

- Una variabile è composta da due elementi: il suo nome e il suo valore; in un programma posso usare i nomi delle variabili al posto dei valori che rappresentano.
- Usando le variabile posso generalizzare un processo:
  - Se ho bisogna di determinare se un numero è primo generalizzo il processo usando una variabile al posto del numero da controllare
  - Mi basterà assegnare alla variabile il valore del numero da verificare per raggiungere l'obiettivo.
- Se do alle variabili nomi significativi aumento la leggibilità del programma

VARIABILI

- Prima di usare una variabile la dichiaro usando l'istruzione var.
- Per assegnare alla variabile un valore utilizzo l'operatore di assegnazione ("=").

DEFINIRE UNA VARIABILE

parola chiave
(direttiva)

separatore

```
var pippo;
```

Identificatore
(nome della variabile)

ASSEGNARE UN VALORE

identificatore (nome della variabile)

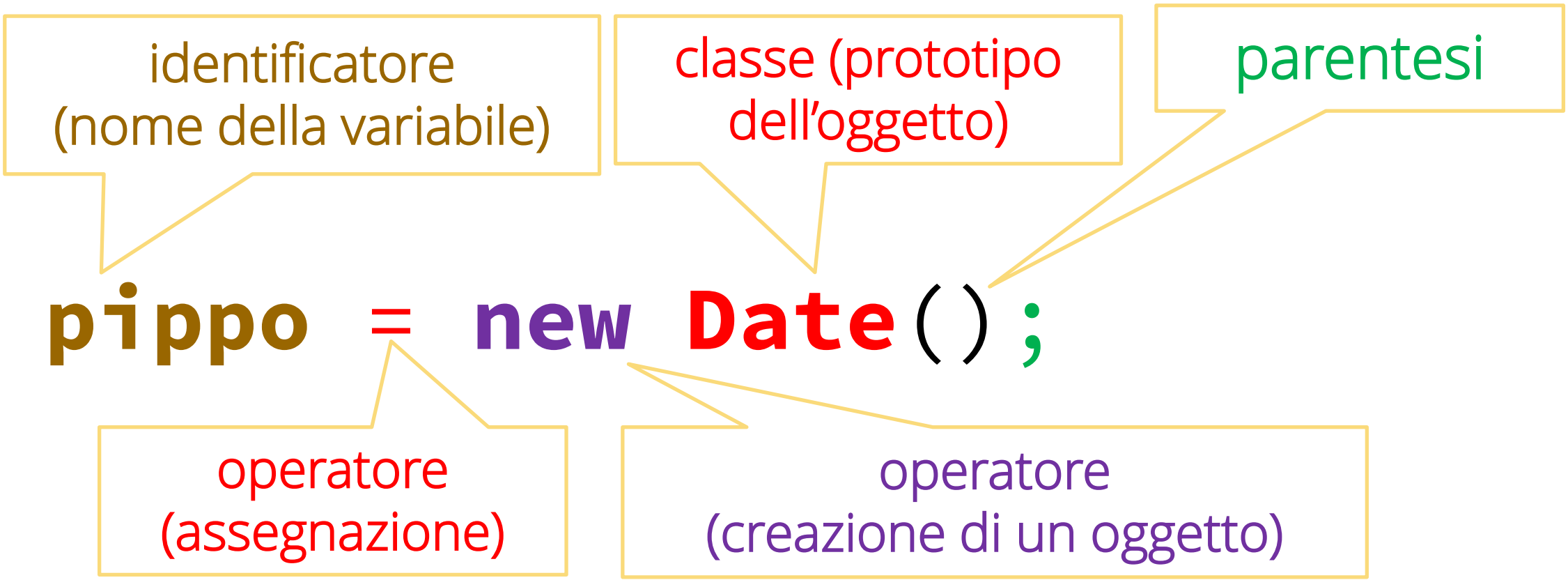
costante stringa

separatore

```
pippo = "Ciao gente!";
```

operatore (assegnazione)

ASSEGNARE UN VALORE



TIPI DI DATI

Tipi di dati, oggetti e classi

OGGETTO e CLASSE

- L'oggetto è una grandezza informatica in grado di rappresentare i dati.
- Nella programmazione **OOP** tutte i dati si rappresentano e si elaborano tramite oggetti.
- Quando scrivo:

```
pippo = new Date();
```

- o:

```
pippo = "Ciao gente!";
```

- non assegno semplicemente un valore a una variabile. Creo un oggetto di uno specifico tipo che oltre ai dati contiene tutti gli strumenti necessari ad elaborarlo.

OGGETTO e CLASSE

- Nella programmazione **OOP** i tipi di oggetto (il **tipo** è, in generale, determinato dal tipo di dati che l'oggetto è destinato ad elaborare) si chiamano **classi**.
- La classe, cioè, è l'insieme di regole che determinano come funziona un oggetto di una determinato tipo (o, i termini OOP, un *istanza di una determinata classe*).
- In Javascript non si fa menzione espressa del temine class (che per altro è una parola riservata che non può essere usata dall'utente). Per noi quindi **oggetto** (inteso come tipo di oggetto) e **classe** saranno sinonimi.
- Esistono oggetti (classi) predefinite dal linguaggio, oggetti (classi) create dalla comunità dei programmatori e che posso caricare e utilizzare nelle mie pagine Web e, infine, noi stessi possiamo creare le nostre classi per risolvere i nostri problemi.

OGGETTI (CLASSI) PREDEFINITI

- Tipi di base (o tipi semplici):
  - **Number**: numeri interi e numeri con parte decimale
  - **String**: Stringhe di caratteri (testo)
  - **Boolean**: Vero e falso
- Tipi complessi:
  - **Array**: Lista indicizzata di valori.
  - **Date**: Date.
  - **RegExp**: *Regular Expression.*
- E infine l'oggetto (classe) **Object** che rappresenta un classe senza regole. Con Object posso rappresentare una struttura dati qualsiasi e costruire dei tipi di dati utente, le mie classi.

OGGETTI (CLASSI) STATICI

- Oggetti (classi) predefiniti, già creati da javascript da cui non si possono creare istanze, ma che hanno solo metodi e proprietà stattiche.
  - **Math**: libreria di funzioni matematica
  - **JSON**: Trasformazione di oggetti nella loro rappresentazione stringa e viceversa
  - **localStorage**: Metodi per gestire la memorizzazione di informazioni sul dispositivo dell'utente.

PROPRIETÀ E METODI

- Ogni tipo di oggetto o classe è caratterizzato da:
- Proprietà che ci consentono di leggere o modificare determinate caratteristiche di un individuo della classe
- Metodi che ci mettono a disposizione determinate **azioni** che gli oggetti possono compiere o subire

PROPRIETÀ

- Le proprietà si usano come le variabili, solo che non sono globali ma riferite ad un elemento specifico: l'oggetto (l'istanza della classe)
- Una volta che una variabile contiene un oggetto

```
pippo = "Ciao gente!";
```

- Si può accedere alle sue proprietà con due sintassi diverse

ACCESSO ALLE PROPRIETÀ

- Usando l'operatore di appartenenza "."

```
var len = pippo.length;
```

- Usando le parentesi quadre che racchiudano il **nome della proprietà come stringa di caratteri**.

```
var len = pippo["length"];
```

- In entrambi i casi `len` conterrà il numero di caratteri di cui è composta la variabile stringa `pippo`

ACCESSO ALLE PROPRIETÀ

- Alcune proprietà sono a sola lettura

```
pippo.length = 3;
```

è un errore perché la proprietà **`length`** è a sola lettura.

```
elemento.innerHTML = "Ciao gente!";
```

modifica il contenuto HTML dell'elemento della pagina **`elemento`** (che un elemento HTML definito nella pagina).

METODI

- I metodi si usano come le funzioni, solo che non sono globali ma riferite ad un elemento specifico: l'oggetto (l'istanza della classe)
- Una volta che una variabile contiene un oggetto

```
pippo = new Date();
```

- Posso accedere ai suoi metodi.
- Come le funzioni i metodi possono ricevere parametri e ritornare valori.

ACCESSO A UN METODO

- Uso l'operatore di appartenenza "." e faccio seguire l'identificatore dall parentesi tonde

```
var giorno = pippo.getDate();
```

`giorno` contiene il giorno del mese della data corrente.

- Alcuni metodi accettano parametri:

```
pippo.setDate(1);
```

La variabile `pippo` conterrà il primo giorno del mese corrente

PROPRIETÀ E METODI STATICI

- Esistono metodi e proprietà particolari che non si applicano ai singoli oggetti (istanze de una classe) ma hanno un uso globale.
- Per usarli non li applico all'oggetto ma alla classe

```
var data = Date.parse("March 21, 2012");
```

- La variabile **data** contiene l'oggetto Date ottenuto convertendo in data la stringa passata come parametro al metodo statico **Date.parse**

L'OGGETTO document

- Quando il browser carica la pagina web crea automaticamente l'oggetto **document**.
- L'oggetto **document** è costituito dall'insieme di tutti gli elementi HTML definiti nella pagina.
- La maggior parte dell'attività di programmazione in javascript consiste nell'interagire con l'oggetto **document**: aggiungere elementi, estrarre elementi e modificarli, rispondere con azioni agli eventi provocati dall'utente ecc.
- Come epr gli altri oggetti l'accesso alle proprietà e ai metodi avviene attraverso l'operatore di appartenenza (.)
- Se, per esempio, voglio recuperate l'elemento con id mio_id:

```
var elemento = document.getElementById('mio_id');
```

COSTRUCTOR

- È la particolare funzione che consente di creare un oggetto di un particolare tipo (un'istanza di una classe):
  - Implicito:

```
var str = "Ciao!";
```

  - Esplicito:

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

  - Altra funzione o metodo:

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

FUNZIONI

COSA È UNA FUNZIONE

- Una funzione (o metodo) è una costrutto presente in tutti i linguaggi di programmazione che consente di associare un gruppo di comandi ad un identificatore.
- Quando nel programma scriverò l'identificatore saranno eseguiti tutti i comandi che compongona la funzione

DEFINIZIONE

- Una funzione deve essere **dichiarata** e **definita**;
  - cioè vanno specificati il nome e il numero di parametri che verranno utilizzati nel corpo della funzione
  - e successivamente dovremo scrivere il corpo della funzione vera e propria.
  - all'interno del corpo della funzione potrò definire un valore di ritorno.

SCRITTURA DI FUNZIONI CON NOME

```
function numefunzione (parametro1, parametro2, ….) {
    // Blocco di istruzioni
}
```

- nomefunzione è il nome univoco della funzione. Tutti i nomi di funzione in un documento devono essere univoci.
- parametro1, parametro2, ... uno o più parametri che vengono passati alla funzione. I parametri sono detti anche *argomenti*.
- Blocco di istruzioni contiene tutto il codice *JavascriptScript* relativo alla funzione. Questa parte contiene le istruzioni che eseguono le azioni, ovvero il codice che si desidera eseguire. Il commento *// Blocco di* istruzioni è un segnaposto che indica dove deve essere inserito il blocco della funzione.

SCRITTURA DI FUNZIONI ANONIME

```
var nomevariabile = function (parametro1, parametro2, ….) {
        // Blocco di istruzioni
}
```

- nomevaribile è il nome di una variabile.
- parametro1, parametro2, … uno o più parametri che vengono passati alla funzione. I parametri sono detti anche *argomenti*.
- Blocco di istruzioni contiene tutto il codice *ActionScript* relativo alla funzione. Questa parte contiene le istruzioni che eseguono le azioni, ovvero il codice che si desidera eseguire.

PARAMETRI E VALORI DI RITORNO

- I parametri (un elenco di valori separati tra virgole contenuti tra le parentesi tonde che caratterizzano la funzione) servono per passare alla funzione i valori su cui compiere l'elaborazione .
- Una funzione può restituire un valore che di norma è il risultato dell'elaborazione compiuta. Per compiere questa operazione si utilizza l'istruzione **return** che specifica il valore che verrà restituito dalla funzione.
- L'istruzione **return** ha anche l'effetto di interrompere immediatamente il codice in esecuzione nel corpo della funzione e restituire immediatamente il controllo del flusso di programma al codice chiamante

SCOPO DELLE VARIABILI

- Lo scopo delle variabili definisce l'ambito della loro visibilità.
- Con l'introduzione delle funzione ho di fatto definito due livelli di script:
  - Uno script di primo livello che viene eseguito immediatamente quando il browser lo incontra.
  - Uno script di secondo livello che sta dentro il corpo delle funzioni e che viene eseguito solo quando le funzioni sono richiamate.
- Quando definisco una variabile nello script di primo livello il suo valore sarà disponibile in tutto il codice della pagina web.
- Quando definisco una variabile nello script di secondo livello, cioè all'interno del corpo di una funzione, il suo valore sarà disponibile solo all'interno della funzione. Così in funzioni diverse potrò avere senza problema variabili con lo stesso nome senza che i valori contenuti interferiscano l'uno con l'altro.

parola chiave (direttiva)

ESEMPIO

```
function cheData() {
        var adesso, dataOraStringa, mesi;
        mesi = ["Gennaio", "Febbraio", "Marzo", "Aprile", "Maggio",
                "Giugno", "Luglio", "Agosto", "Settembre", "Ottobre",
                "Novembre", "Dicembre"];
        adesso = new Date();
        dataOraStringa = adesso.getDate() + " " + mesi[adesso.getMonth()] + " " +
                         adesso.getFullYear() + " " + adesso.getHours() + ":" +
                         adesso.getMinutes();
        return dataOraStringa;
}
```

NUMBER

constructor

```
var n = 5;
var n = new Number(5);
var n = 10.6;
var n = new Number(10.6);
```

proprietà statiche

Proprietà	Descrizione
MAX_VALUE	Restituisce il massimo numero consentito in JavaScript
MIN_VALUE	Restituisce il minimo numero consentito in JavaScript
NEGATIVE_INFINITY	Rappresenta l'infinito negativo.
POSITIVE_INFINITY	Rappresenta l'infinito positivo.

metodi

Method	Description
toExponential(x)	Reastituisce una stringa, che rappresenta il numero come notazione esponenziale dove x (opzionale) indica il numero dei decimali da usare
toFixed(x)	Converte il numero in una stringa, con x numero di decimali. Se x non viene specificato nessun decimale.
toPrecision(x)	Converte il numero in una stringa di lunghezza x. Se necessario vengono aggiunti punto decimale e 0.
toString(base)	Converte il numero in una stringa secondo la base specificata da base. La base di default è 10 (numero decimale). Se base vale 2 si ottiene la rappresentazione binaria del numero, se 16 quella esadecimale, ecc.

STRING

CONSTRUCTOR

```
var str = "Ciao!";
var str = new String("Ciao!");
```

PROPRIETÀ

- Gli oggetti della classe String hanno una sola proprietà, la proprietà **length** che restituisce la lunghezza della stringa, cioè il numero di caratteri di cui è composta.

MANIPOLAZIONE

Method	Description
charAt(pos)	Restituisce il carattere alla posizione pos
charCodeAt(pos)	Restituisce il carattere (in formato Unicode) alla posizione pos
concat(s1, s2)	Concatena due stringhe (come s1 + s2)
fromCharCode(code)	Restituisce il carattere corrispondente al valore unicode code
indexOf(searchstring, start)	Restituisce la posizione della prima occorrenza della stringa searchstring in una stringa (-1 se non lo trova). Opzionalmente la ricerca può partire dalla posizione start
lastIndexOf(searchstring, start)	Restituisce la posizione dell'ultima occorrenza della stringa searchstring in una stringa (-1 se non lo trova). Opzionalmente la ricerca può partire dalla posizione start in vece che dall'ultimo carattere.
match(regexp)	Il metodo match cerca le corrispondenza e tra l'espressione regolare regexp e la stringa, e restituisce un array di corrispondenze. Se non viengono trovate corrispondenze viene restituito null.
replace(regexp/substr, newstring)	Replace() cerca una corrispondenza tra una stringa (o un'espressione regolare) e una stringa, e sostituisce la corrispondenze trovate con newstring
search(regexp)	Il metodo search cerca le corrispondenza e tra l'espressione regolare regexp e la stringa, e restituisce la posizione in cui è stata trovata oppure -1 se non viengono trovate corrispondenze.
slice(inizio, fine)	Estrae la parte di una stringa compresa tra inizio e fine e restituisce la parte estratta in una nuova stringa. In caso non sia passato un valore, fine sarà l'ultimo carattere della stringa.
split(char)	Converte la stringa in un array usando char come carattere di separazione.
substr(start, length)	Estrae lentgth caratteri dalla stringa, a partire da start e li restituisce in una nuova stringa. Se length non è specificati vengono restituiti i caratteri da start fino alla fine della stringa.
substring(from, to)	Estrae i caratteri delle stringa tra from e to non compreso. Se to è omesso fino alla fine della stringa.
toLowerCase()	Converte in minuscolo
toUpperCase()	Converte in maiuscolo

ARRAY

CONSTRUCTOR

```
var a = [1,6,78,23];
var a = new Array(1,6,78,23);
```

PROPRIETÀ

- Gli oggetti della classe Array hanno una sola proprietà, la proprietà **length** che restituisce la lunghezza dell'array, cioè il numero di elementi di cui è composto.

METODI

Method	Description
concat(array2,array3, ..., arrayX)	Unisce uno o più array all'array a cui il metodo è applicato, e restituisce una copia degli array così uniti.
indexOf(elemento, start)	Cerca elemento in un array partendo da start (o dall'inizio se start è omesso) e ne restituisce la posizione. Se start è negativo indica la posizione reativa alla fine dell'array.
join(separatore)	Unisce gli elementi di un array in una stringa, e restituisce la stringa. Gli elementi sono separati da separatore. Il separatore di default è la virgola .
lastIndexOf(elemento, start)	Cerca l'ultima ricorrenza di elemento in un array partendo da start (o dall'iniziose start è omesso) e ne restituisce la posizione o -1 se elemento non viene trovato.
pop()	Rimuove l'ultimo elemento di un array, e restituisce l'elemento rimosso.
push(elemento)	Aggiunge elemento alla fine dell'array e restituische la nuova lunghezza.
reverse()	Inverte l'ordine degli elementi dell''array.
shift()	Rimuove il primo elemento di un array, e restituisce l'elemento rimosso.
slice(inizio, fine)	Estrae gli elementi a partire da inizio, fino a fine, non incluso e li restituisce in un nuovo array. L'array originale non viene modificato.
sort(sortfunct)	Ordina gli elementi di un array (alfabetico ascendente) o usa sortfunct per stabilire l'ordine
splice(indice, quanti, item1,....., itemX)	Rimuove quanti elementi dall'array a partire dalla posizine indice e inserisce gli elementi item1, ..., itemX (se forniti) a partire dalla posizine indice. Restituisce gli elementi rimossi.
toString()	Restituisce l'array convertito in stringa.
unshift(elemento)	Aggiunge elemento all'inizio dell'array e restituische la nuova lunghezza

sort

```
var rubrica = [
              {nome:"Mario", cognome:"Rossi" },
              {nome:"Luigi", cognome:"Neri" },
              {nome:"Piero", cognome:"Verdi" },
              {nome:"Mario", cognome:"Bianchi" }
            ];
var sortCognome = function (a,b){
                        if (a.cognome > b.cognome){
                          return 1;
                        } else if (a.cognome == b.cognome){
                          return 0;
                        } else {
                          return 1;
                        }
                    };
rubrica.sort(sortCognome);
```

DATE

CONSTRUCTOR

```
var d = new Date();
var d = new Date(milliseconds);
var d = new Date(dateString);
var d = new Date(year, month, day,
                 hours, minutes,
                 seconds,
                 milliseconds);
```

Metodo statico

Date.parse(str)	Analizza una data in formato stringa e restituisce il numero di millisecondi dalla mezzanotte del 1 Gennaio 1970.

Metodi	Descrizione
getDate()	Restituisce il giorno del mese (1-31)
getDay()	Restituisce il giorno della settimana (0-6, 0 = domenica)
getFullYear()	Restituisce l'anno (quattro cifre)
getHours()	Restituisce l'ora (da 0-23)
getMilliseconds()	Restituisce i millisecondi (0-999)
getMinutes()	Restituisce i minuti (0-59)
getMonth()	Restituisce il mese (0-11)
getSeconds()	Restituisce i secondi (0-59)
getTime()	Restituisce il numero di millisecondi trascorsi dalla mezzanotte del 1 gennaio 1970
getTimezoneOffset()	Restituisce la differenza di tempo tra il GMT e l'ora locale, in pochi minuti
getUTCDate()	Restituisce il giorno del mese, in base all'ora universale (da 1-31)
getUTCDay()	Restituisce il giorno della settimana, in base all'ora universale (da0-6)
getUTCFullYear()	Restituisce l'anno, in base all'ora universale (quattro cifre)
getUTCHours()	Restituisce l'ora, in base all'ora universale (da 0-23)
getUTCMilliseconds()	Restituisce i millisecondi, in base all'ora universale (0-999)
getUTCMinutes()	Restituisce i minuti, in base all'ora universale (da 0-59)
getUTCMonth()	Restituisce il mese, in base all'ora universale (da 0-11)
getUTCSeconds()	Restituisce i secondi, in base all'ora universale (da 0-59)

Metodi	Descrizione
setDate()	Imposta il giorno del mese di un oggetto data
setFullYear()	Imposta l'anno (quattro cifre) di un oggetto data
setHours()	Imposta l'ora di un oggetto data
setMilliseconds()	Imposta i millisecondi di un oggetto data
setMinutes()	Impostare i minuti di un oggetto data
setMonth()	Imposta il mese di un oggetto data
setSeconds()	Imposta i secondi di un oggetto data
setTime()	Consente di impostare una data e un'ora aggiungendo o sottraendo un determinato numero di millisecondi per/da mezzanotte del primo gennaio 1970
setUTCDate()	Imposta il giorno del mese di un oggetto data, in base all'ora universale
setUTCFullYear()	Imposta l'anno di un oggetto data, in base all'ora universale (quattro cifre)
setUTCHours()	Imposta l'ora di un oggetto data, in base all'ora universale
setUTCMilliseconds()	Imposta i millisecondi di un oggetto data, in base all'ora universale
setUTCMinutes()	Impostare i minuti di un oggetto data, in base all'ora universale
setUTCMonth()	Imposta il mese di un oggetto data, in base all'ora universale
setUTCSeconds()	Impostare i secondi di un oggetto data, in base all'ora universale
setDate()	Imposta il giorno del mese di un oggetto data
setFullYear()	Imposta l'anno (quattro cifre) di un oggetto data
setHours()	Imposta l'ora di un oggetto data

Metodi	Descrizione
toDateString()	Converte la parte relativa alla data di un oggetto Date in una stringa leggibile
toISOString()	Restituisce la data come una stringa, utilizzando lo standard ISO
toJSON()	Restituisce la data come una stringa, formattato come una dataJSON
toLocaleDateString()	Restituisce la parte relativa alla data di un oggetto Date come una stringa, utilizzando le convenzioni di localizzazione
toLocaleTimeString()	Restituisce la parte di ora di un oggetto Date come una stringa, utilizzando le convenzioni di localizzazione
toLocaleString()	Converte un oggetto Date in una stringa, utilizzando le convenzioni di localizzazione
toString()	Converte un oggetto Date in una stringa
toTimeString()	Converte la parte ora di un oggetto Date in una stringa
toUTCString()	Converte un oggetto Date in una stringa, in base all'ora universale
UTC()	Restituisce il numero di millisecondi in una stringa data a partire dalla mezzanotte del 1 gennaio 1970, in base all'ora universale

MATH

PROPRIETÀ STATICHE

Propietà	Descrizione
E	Restituisce il numero di Eulero (circa 2,718)
LN2	Restituisce il logaritmo naturale di 2 (circa 0,693)
LN10	Restituisce il logaritmo naturale di 10 (circa 2,302)
LOG2E	Restituisce il logaritmo in base 2 di E (circa 1,442)
LOG10E	Restituisce il logaritmo in base 10 di E (circa 0,434)
PI	Restituisce PI (circa 3.14)
SQRT1_2	Restituisce la radice quadrata di 1/2 (circa 0,707)
SQRT2	Restituisce la radice quadrata di 2 (circa 1,414)

metodi statici

Method	Description
abs(x)	Restituisce il valore assoluto di x
acos(x)	Restituisce l'arcocoseno di x, in radianti
asin(x)	Restituisce l'arcoseno di x, in radianti
atan(x)	Restituisce l'arcotangente di x come un valore numerico compreso tra-PI / 2 e PI 2 radianti
atan2(y,x)	Restituisce l'arcotangente del quoziente dei suoi argomenti
ceil(x)	Restituisce X arrotondato per eccesso al numero intero più vicino
cos(x)	Restituisce il coseno di x (x è in radianti)
exp(x)	Restituisce il valore di E elevato alla x
floor(x)	Restituisce X arrotondato per difetto al numero intero più vicino
log(x)	Restituisce il logaritmo naturale (base e) di x
max(x,y,z,...,n)	Restituisce il numero con il valore più alto
min(x,y,z,...,n)	Restituisce il numero con il valore più basso
pow(x,y)	Restituisce il valore di x alla potenza y
random()	Restituisce un numero casuale compreso tra 0 e 1
round(x)	Arrotonda x al numero intero più vicino
sin(x)	Restituisce il seno di x (x è in radianti)
sqrt(x)	Restituisce la radice quadrata di x
tan(x)	Restituisce la tangente di un angolo

REGEXP

constructor

```
var re = new RegExp(pattern,
            mod);
var re = /pattern/modificatori;
var re = /0-9/g;
```

MODIFICATORI

Modificatore	Descrizione
i	Eseguire case-insensitive di corrispondenza
g	Eseguire una partita globale (trovate tutte le partite, piuttosto che fermarsi dopo la prima partita)
m	Effettuare ricerche su righe multiple

parentesi quadre

Espressione	Descrizione
[abc]	Trova qualsiasi carattere tra le parentesi
[^abc]	Trova qualsiasi carattere non tra le parentesi
[0-9]	Trova qualsiasi cifra 0-9
[A-Z]	Trova un carattere tra A maiuscola e Z maiuscola
[a-z]	Trova un carattere tra a minuscola a z minuscola
[A-z]	Trova un carattere da maiuscolo a minuscolo A z
[adgk]	Trova qualsiasi carattere nell'elenco
[^adgk]	Trova un carattere non compreso nell'elenco
(red|blue|green)	Trova una delle alternative indicate

metacaratteri

Metacarattere	Descrizione
..	Trova un singolo carattere, eccetto newline o terminatore di linea
\w	Trova un carattere alfanumerico
\W	Trova un carattere non alfanumerico
\d	Trova una cifra
\D	Trova un carattere non numerico
\s	Trova uno spazio bianco
\S	Trova un carattere non-spazio
\b	Trova un match ad inizio / fine di una parola
\B	Trovare non è una partita ad inizio / fine di una parola
\0	Trova un carattere NUL
\n	Trova un carattere di nuova riga
\f	Trova un carattere di avanzamento modulo
\r	Trova un carattere di ritorno
\t	Trova un carattere di tabulazione
\v	Trova un carattere di tabulazione verticale
\xdd	Trova il carattere specificato da un numero esadecimale dd
\uxxxx	Trova il carattere Unicode specificato da un numero esadecimale xxxx

Quantificatori

Quantificatore	Descrizione
n+	Corrisponde a qualsiasi stringa che contiene almeno un n
n*	Corrisponde a qualsiasi stringa che contiene zero o più occorrenze di n
n?	Corrisponde a qualsiasi stringa che contiene zero o una occorrenze di n
n {X}	Corrisponde a qualsiasi stringa che contiene una sequenza di X n
n {X, Y}	Corrisponde a qualsiasi stringa che contiene una sequenza di n da X a Y
n {X,}	Corrisponde a qualsiasi stringa che contiene una sequenza di almeno X n.
n$	Corrisponde a qualsiasi stringa con n alla fine.
^n	Corrisponde a qualsiasi stringa con n all'inizio.
?=n	Corrisponde a qualsiasi stringa che viene seguita dalla stringa specifica n
?!n	Corrisponde a qualsiasi stringa che non è seguita dalla stringa specifica n

prorietà e metodi

Proprietà	Descrizione
global	Specifica se il modificatore "g" è impostato
ignoreCase	Specifica se il modificatore "i" è impostato
lastIndex	L'indice da cui iniziare la prossima ricerca
multiline	Specifica se il modificatore "m" è impostato
source	Il testo del pattern RegExp

Metodo	Descrizione
compile()	Compila un espressione regolare
exec ()	Cerca la prima occorrenza e la restituisce
test ()	Cerca la prima occorrenza . Restituisce vero o falso

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 1

```
/**
 * 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;
}
```

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.";
}
```

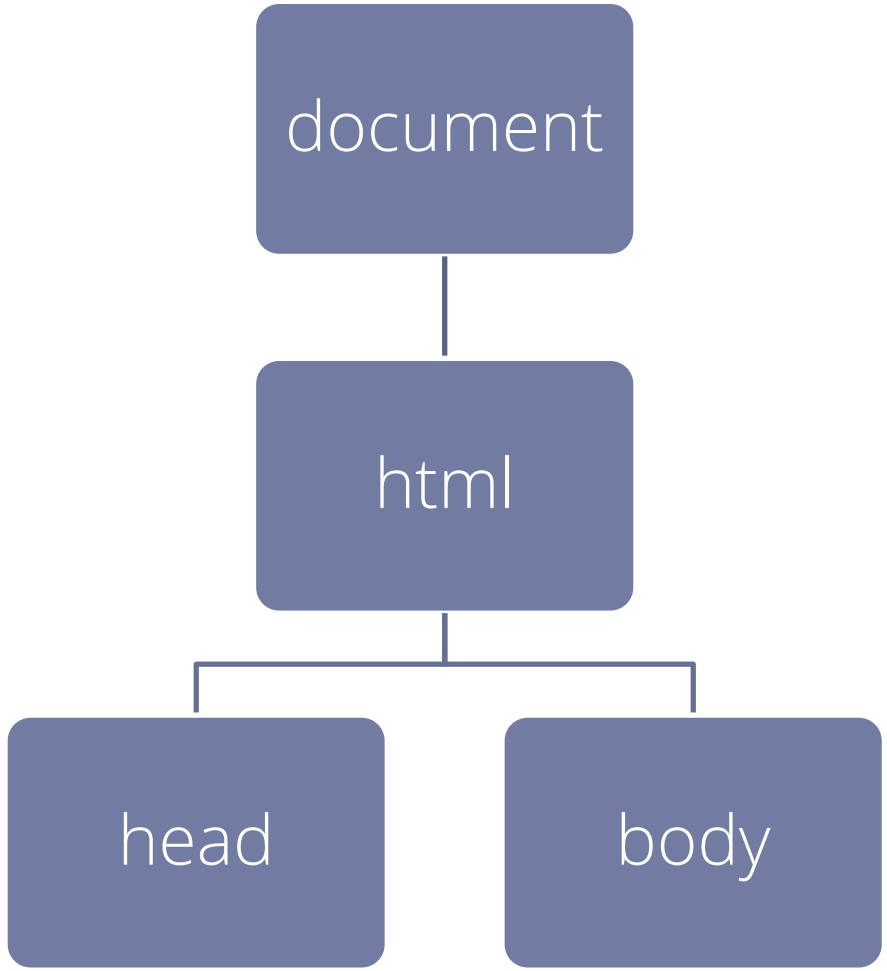
DOCUMENT OBJECT MODEL

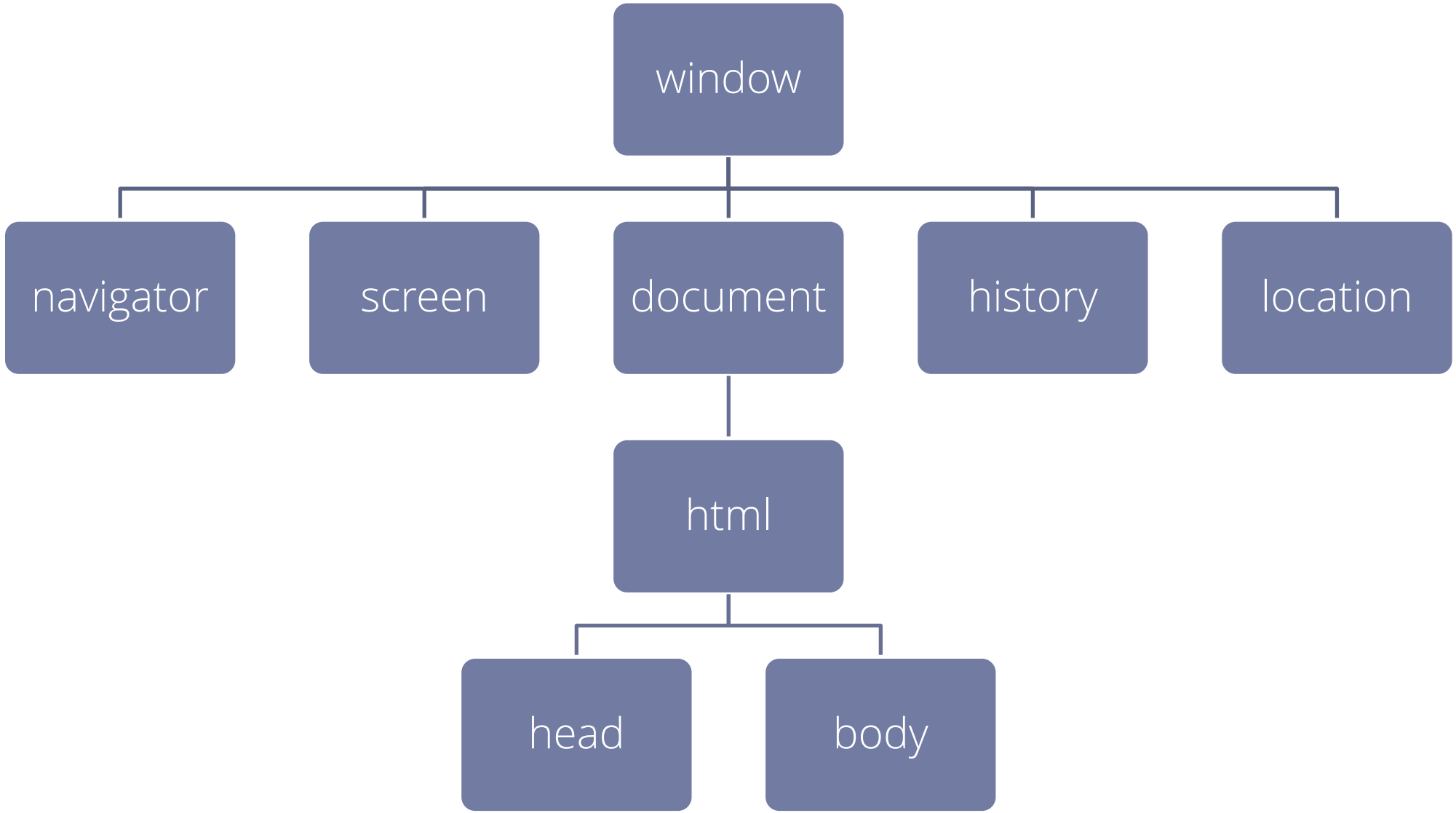
DOM

- 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

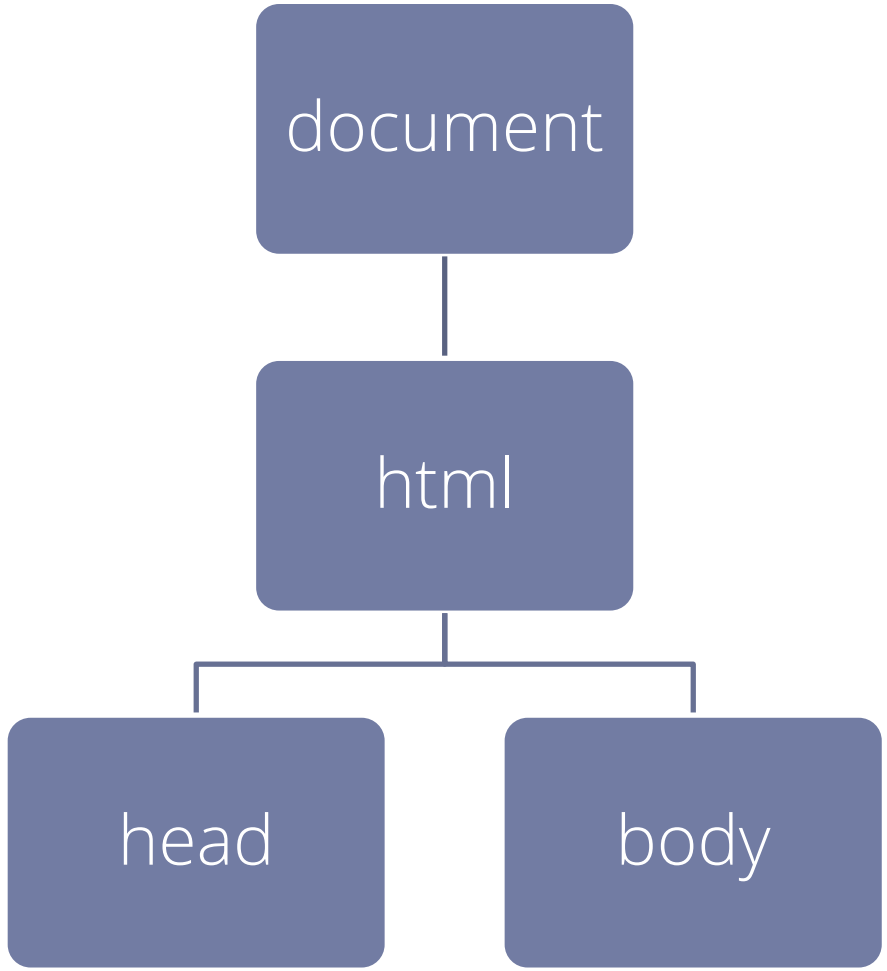
- L'oggetto location rappresenta l'url da cui è stata caricata la pagina
- La sua funzione principale è quella di caricare una pagina diversa nella corrente finestra o frame.
- Allo script è consentito di accedere ad informazioni solo sulla url da cui è stato caricato.

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.

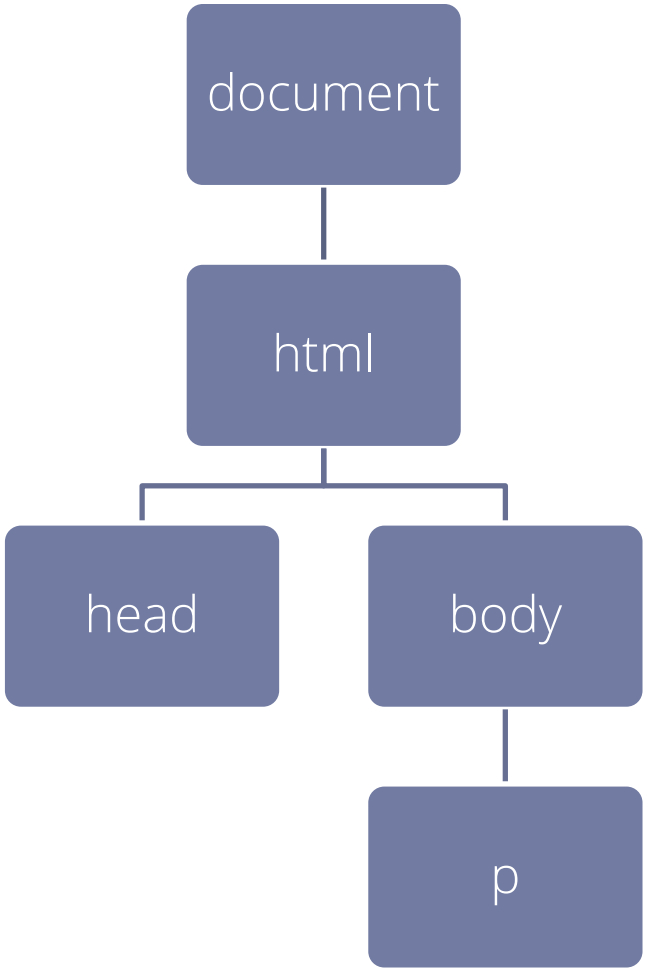
Documento vuoto

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



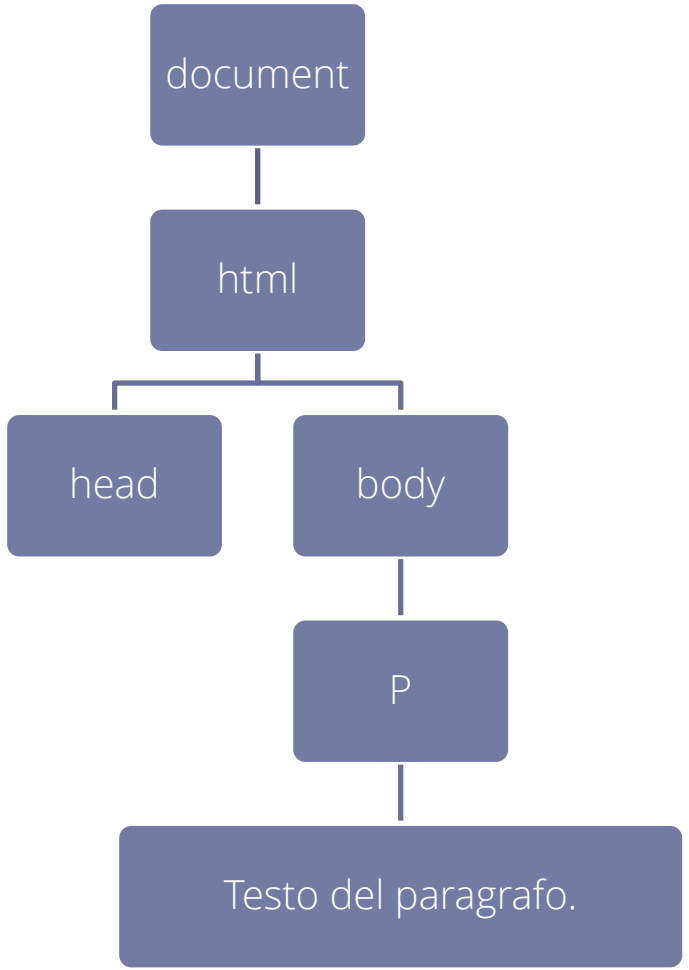
Aggiunta di un paragrafo vuoto

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



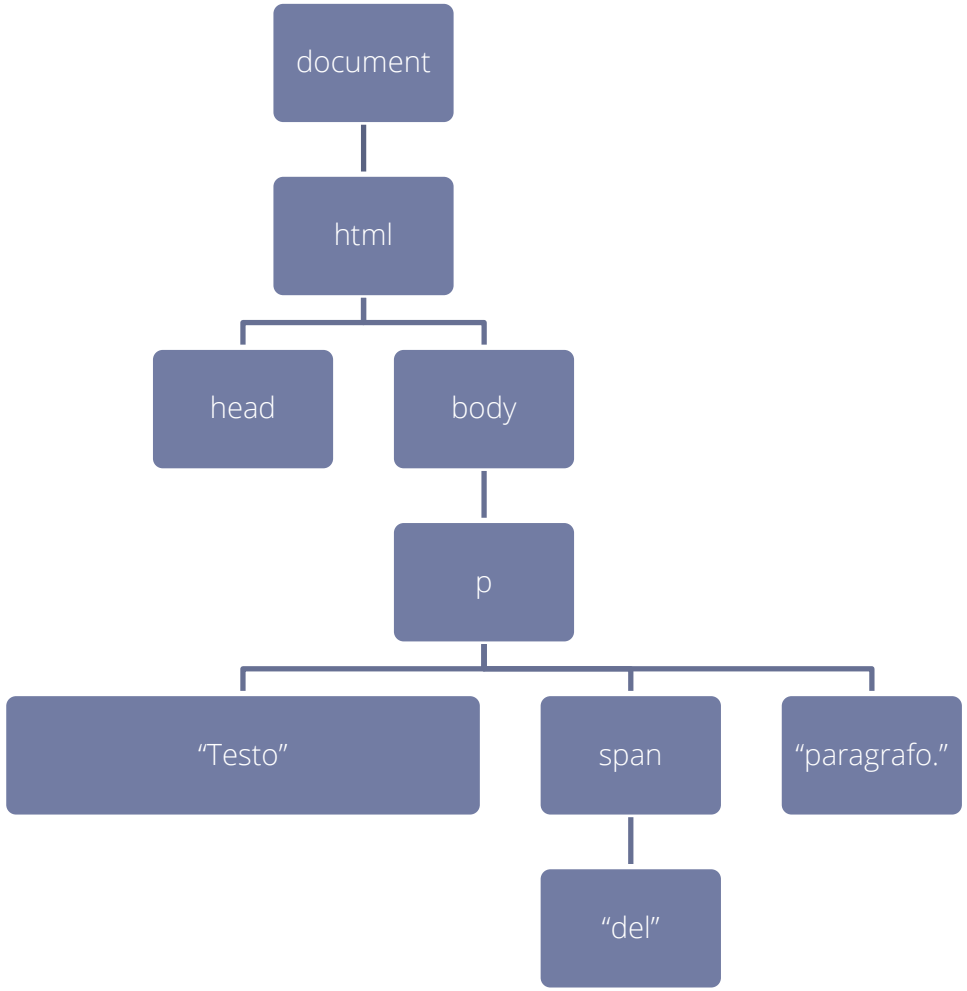
Aggiunta di testo al paragrafo

```
<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

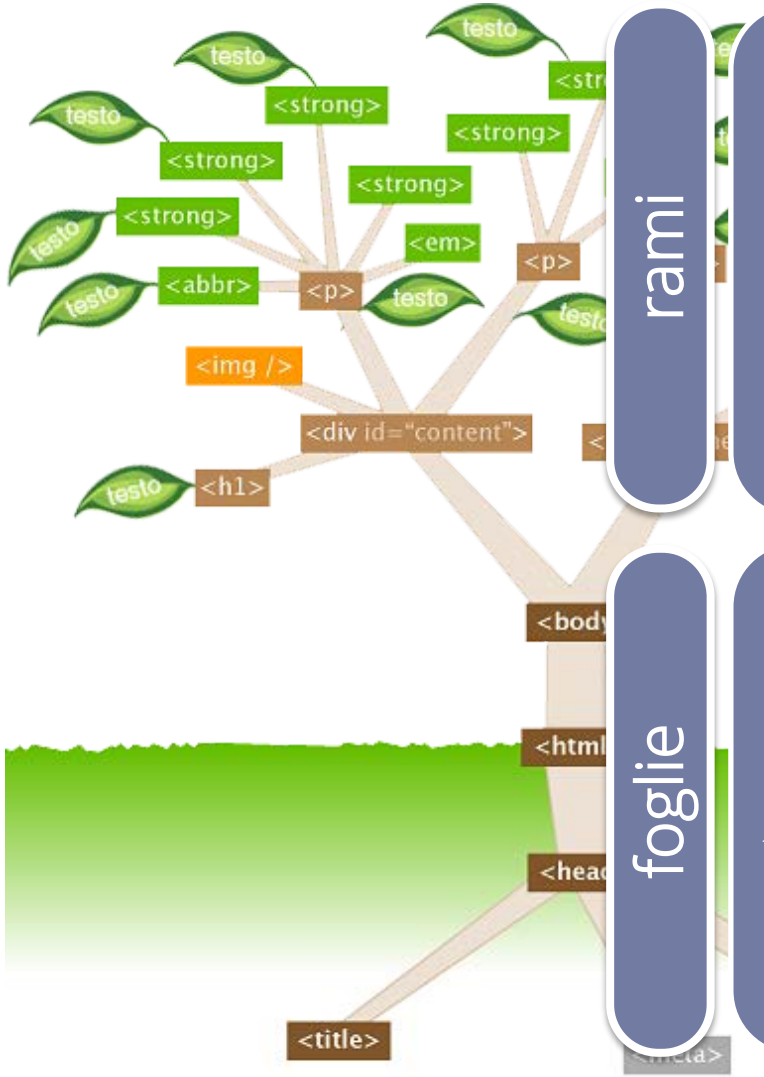
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo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

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;
```

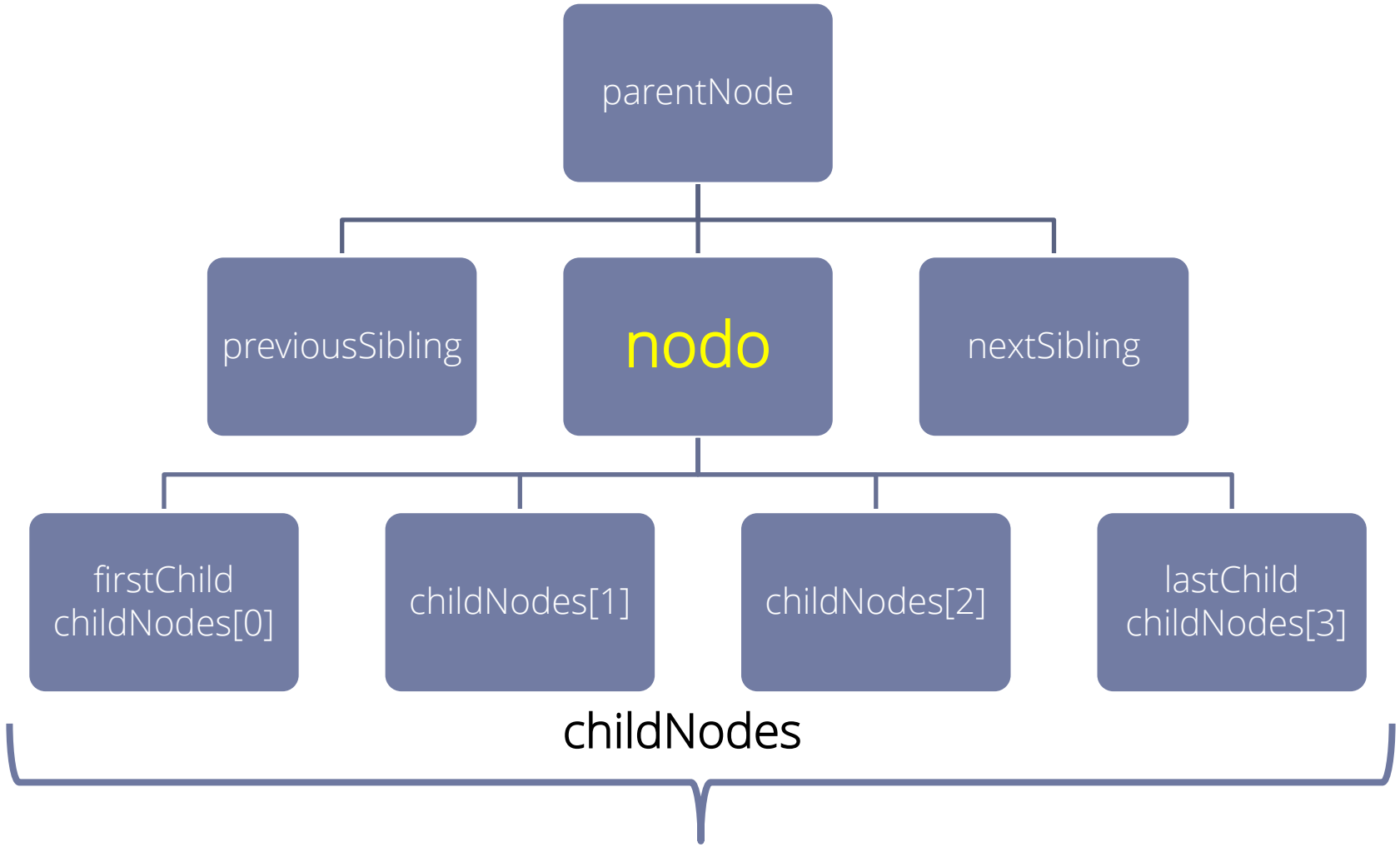
ATTRIBUTI

- 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

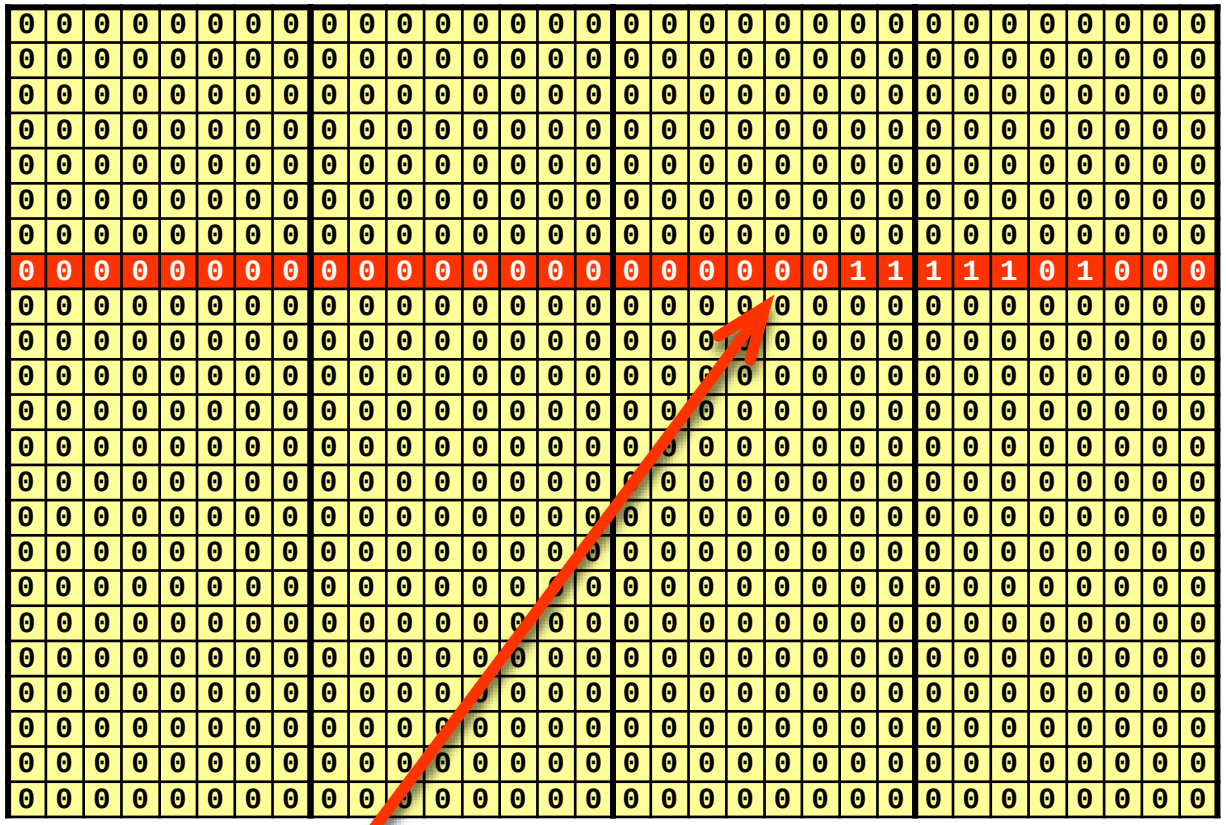
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

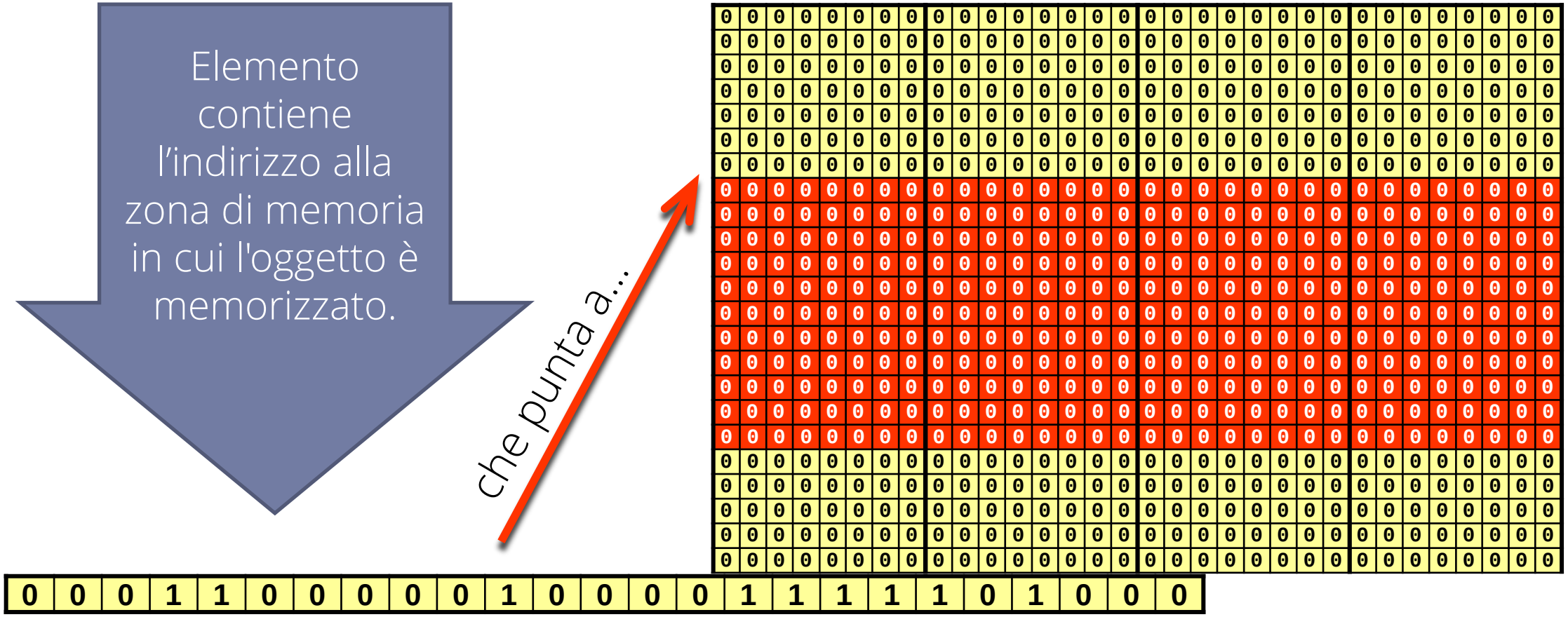
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

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.

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')
```

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....

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.

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:

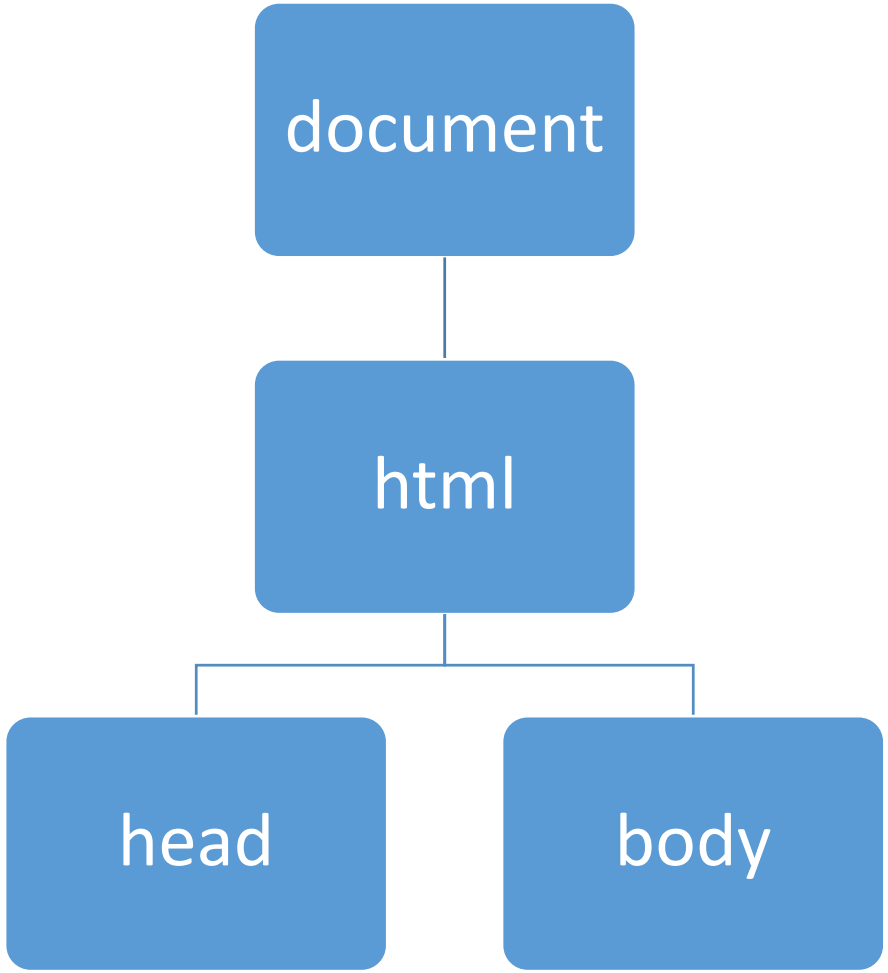
Metodo	Spiegazione	Esempio
alert	Presenta un messaggio all'utente e mostra il bottone Ok	window.alert("messaggio');
confirm	Richiede una conferma all'utente. Mostra i bottoni Ok e Annulla	var risposta; risposta = window.confirm("Vuoi continuare?');
prompt	Richiede all'utente di inserire un valore. Mostra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

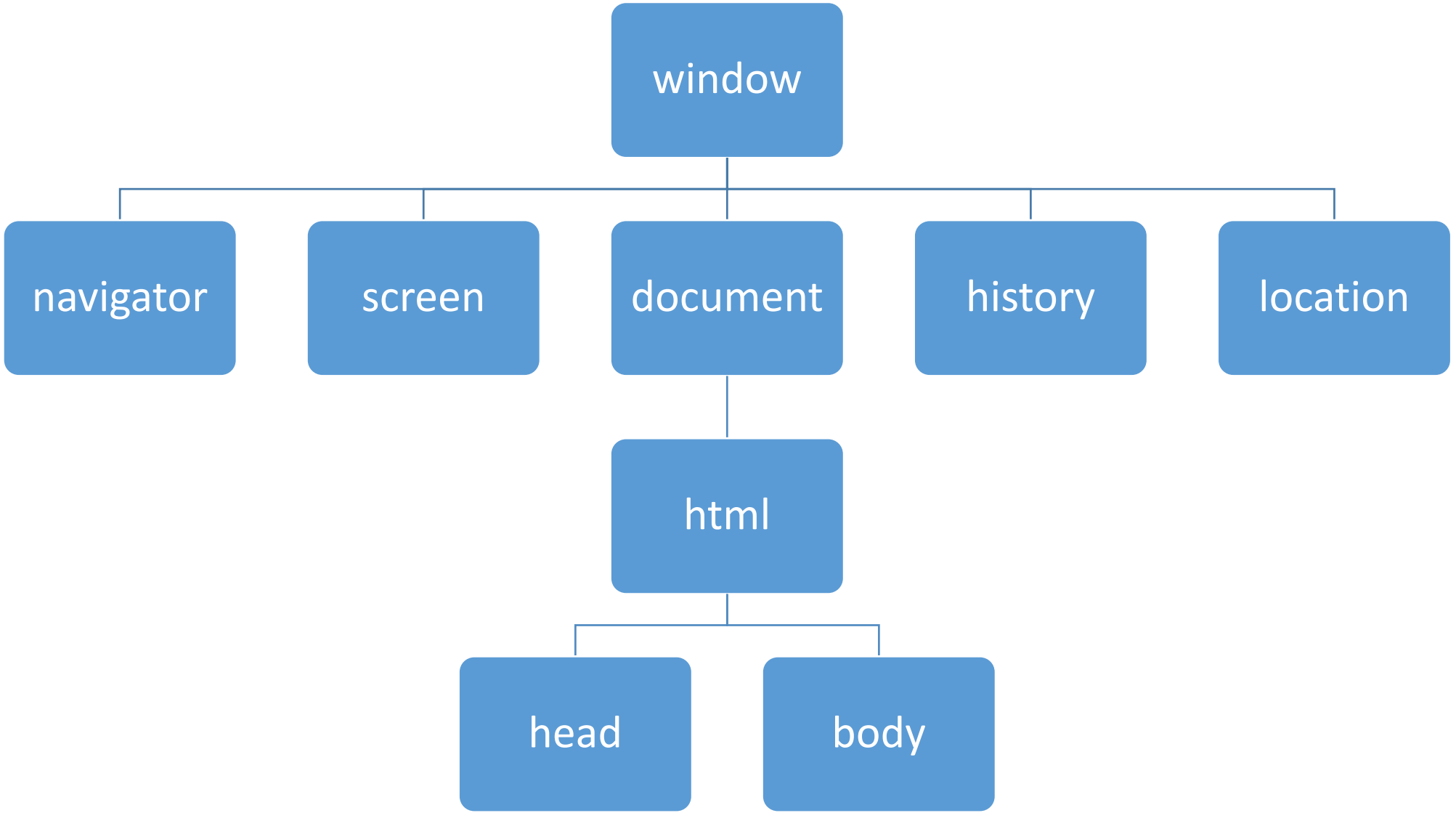
DOM

- 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

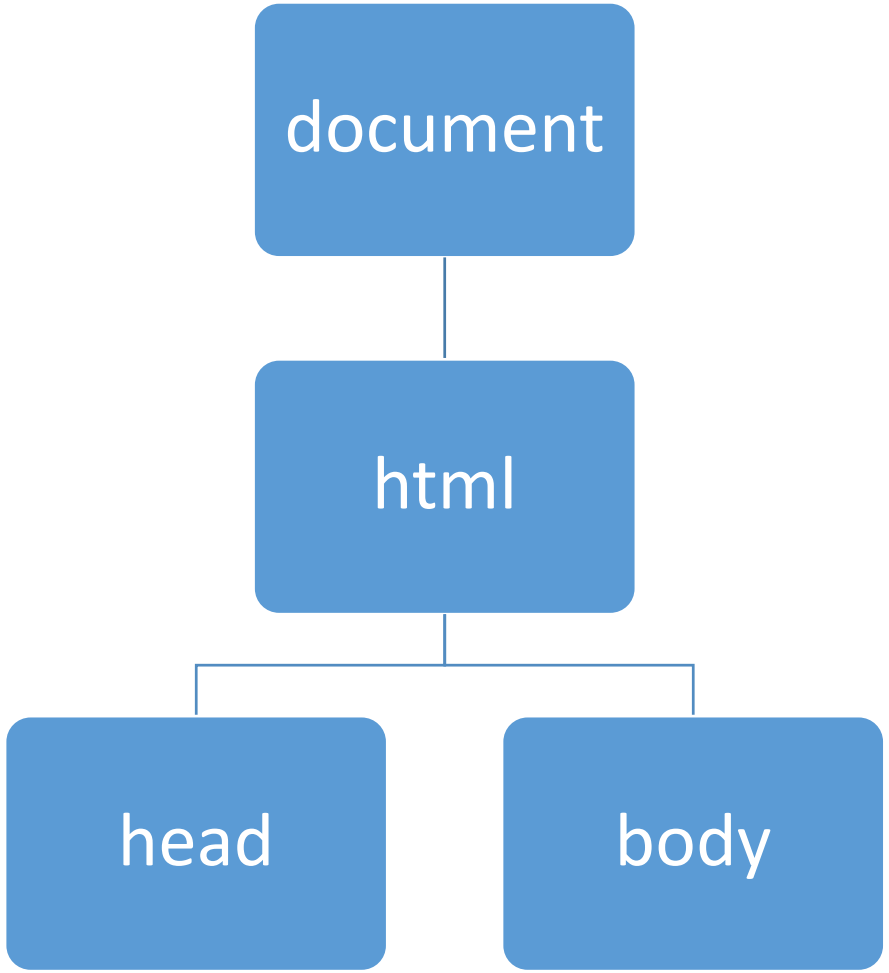
- L’oggetto location rappresenta l’url da cui è stata caricata la pagina
- La sua funzione principale è quella di caricare una pagina diversa nella corrente finestra o frame.
- Allo script è consentito di accedere ad informazioni solo sulla url da cui è stato caricato.

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.

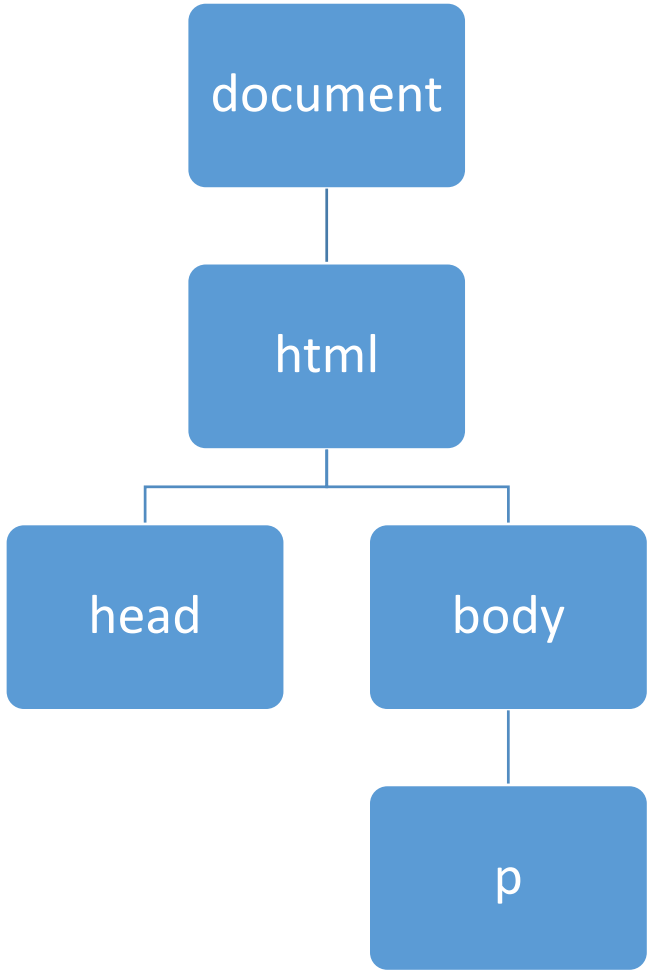
Documento vuoto

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



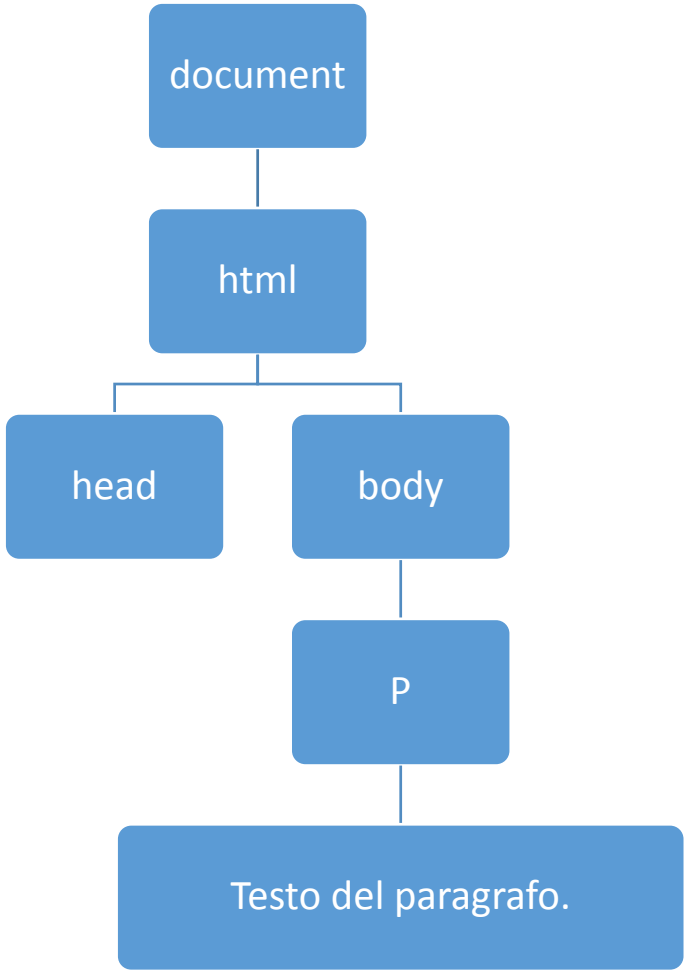
Aggiunta di un paragrafo vuoto

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



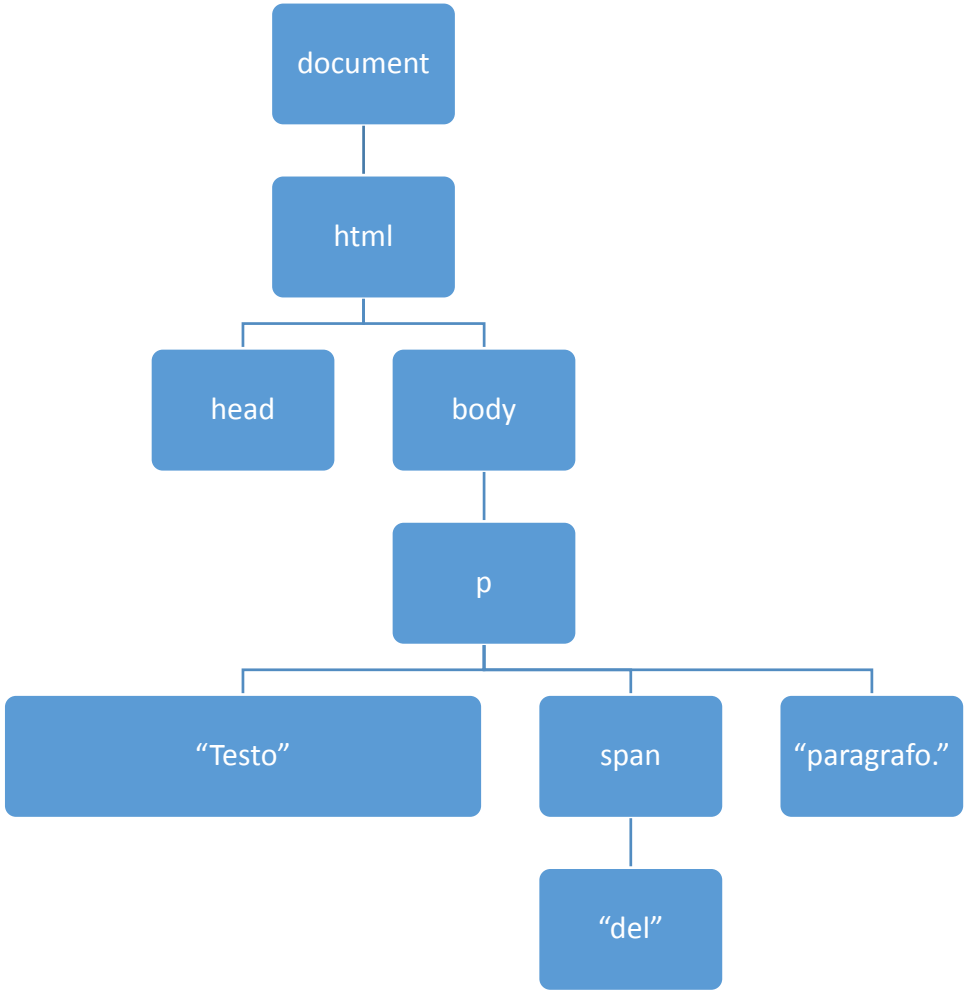
Aggiunta di testo al paragrafo

```
<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

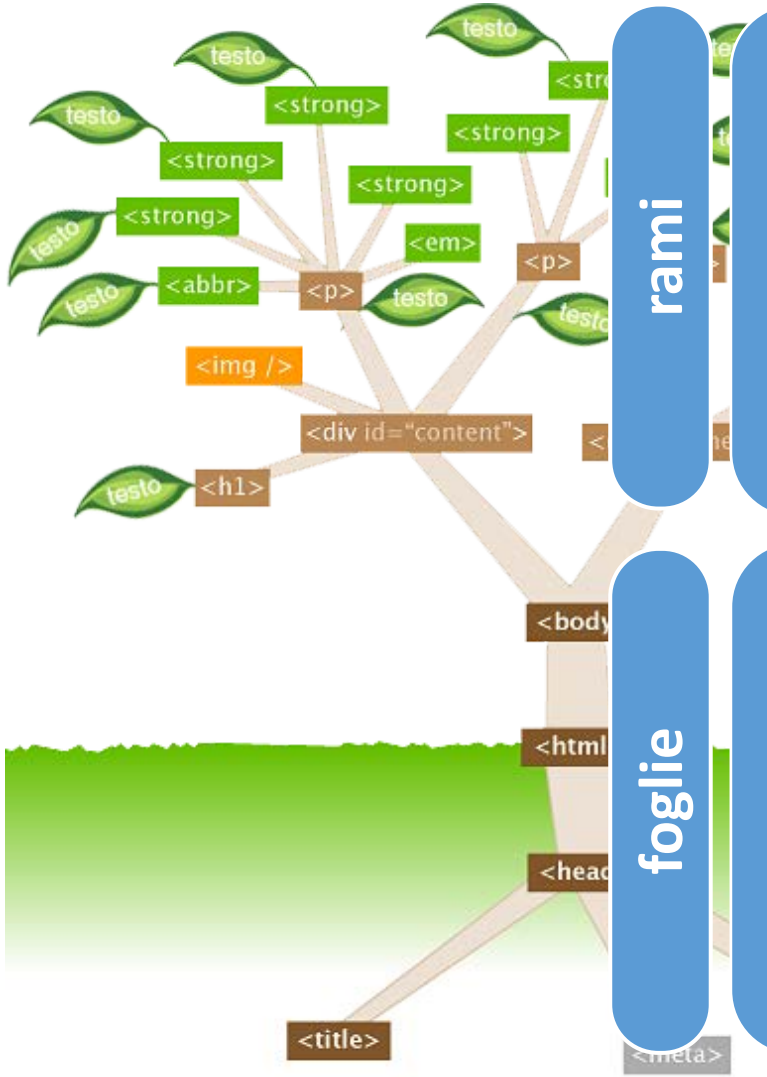
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo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

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);
```

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;
```

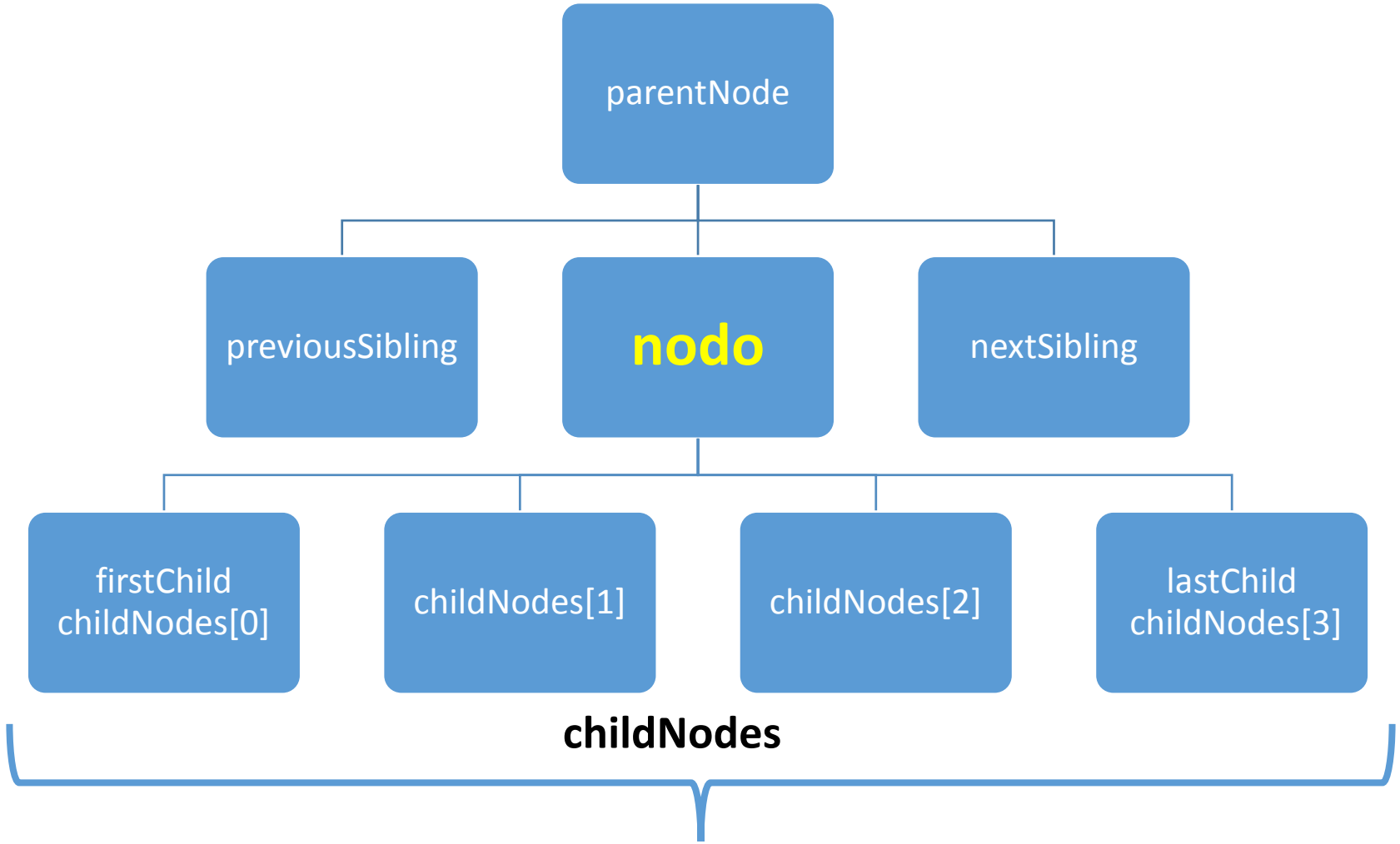
ATTRIBUTI

- **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

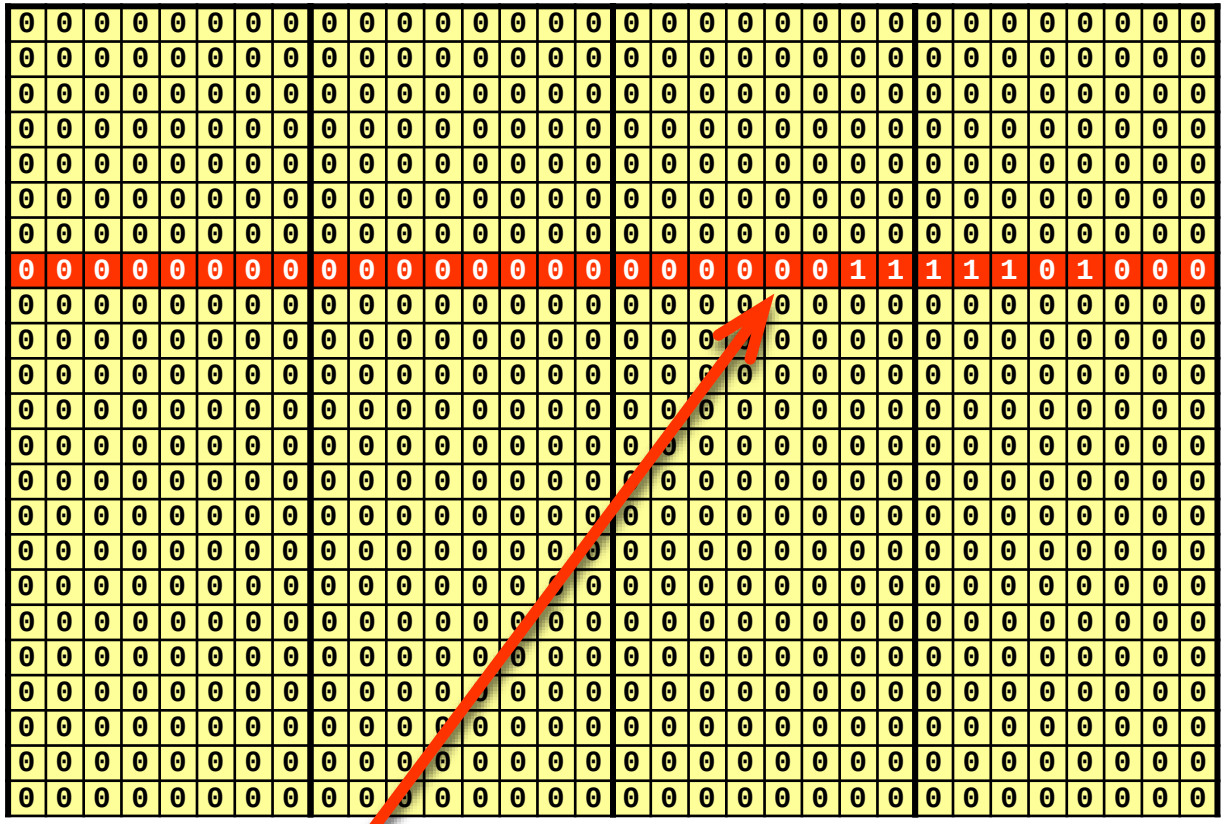
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

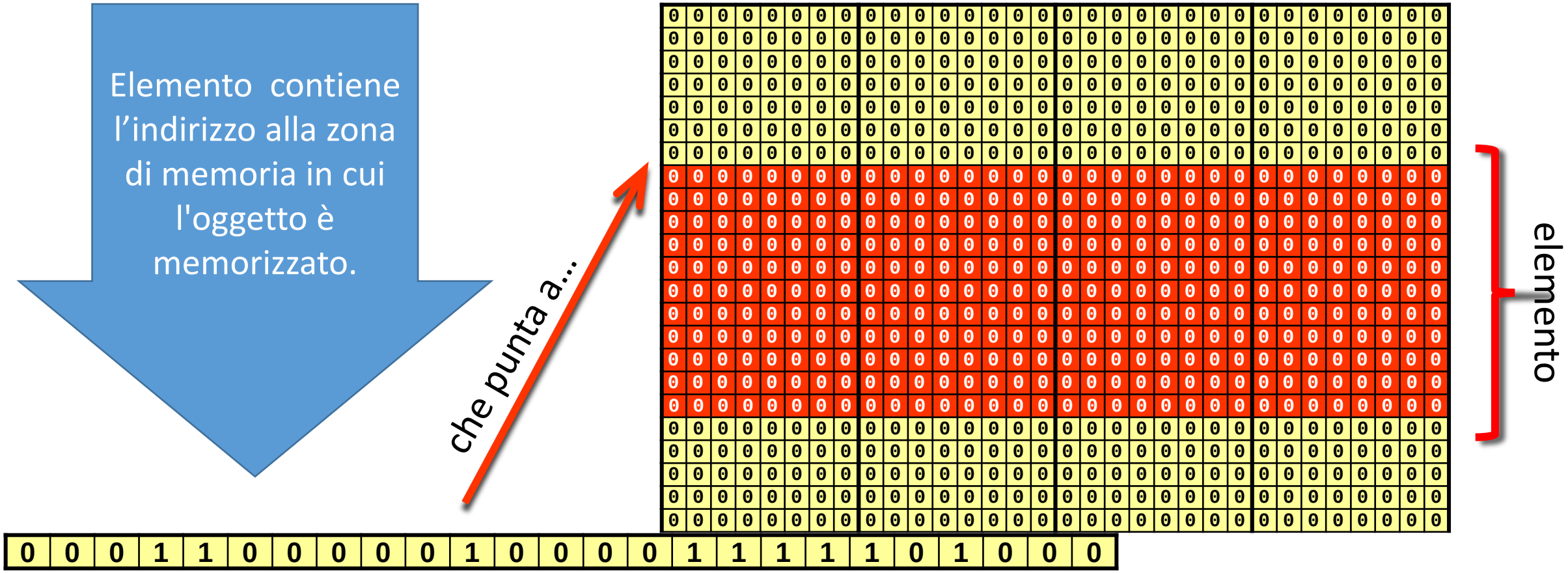
- Quando il valore assegnato a una variabile è un oggetto l'interprete javascript fa un operazione un po' più complessa. Lo spazio di 32 bit riservato alla variabile viene usato per memorizzare l'indirizzo di memoria in cui è collocato l'oggetto.
- In questo caso la variabile contiene il **riferimento**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



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.

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')
```

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….

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.

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');