# LEZIONE 08

## Il Document Object Model

# 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)

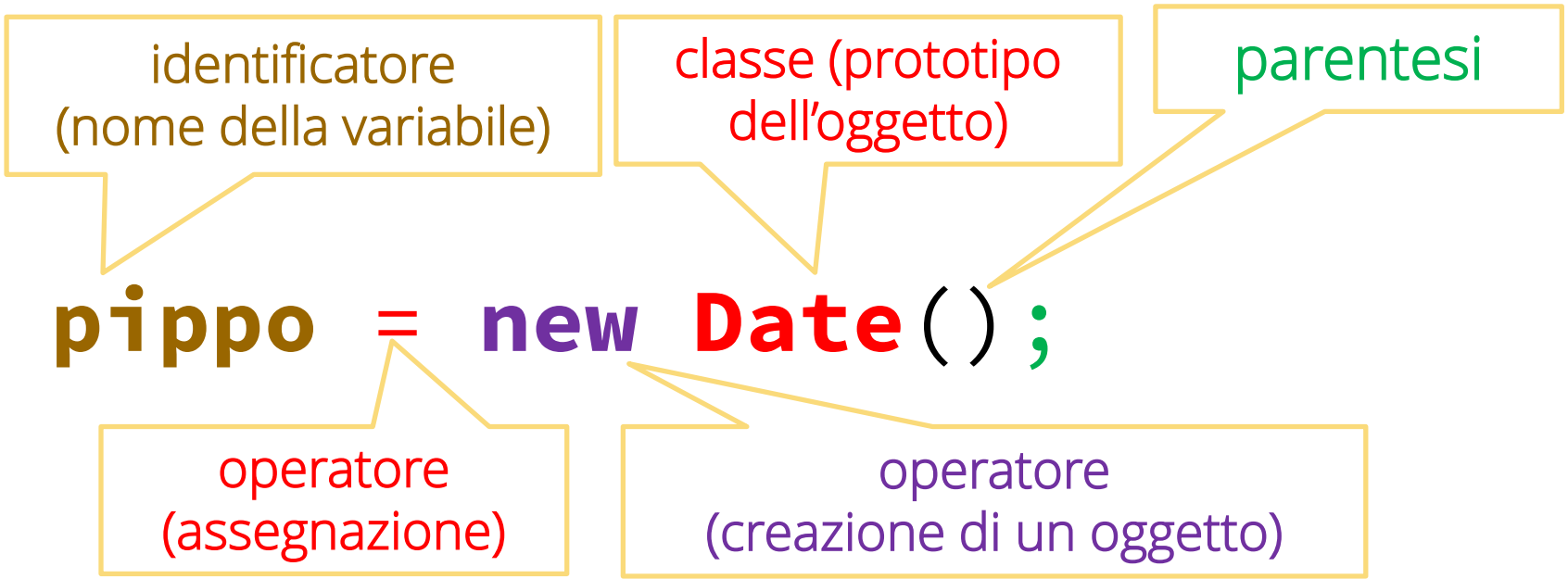
separatore

```
var pippo;
```

Identificatore (nome della variabile)

# ASSEGNARE UN VALORE

# 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 dato complesso detto oggetto che oltre al valore 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 **Object** che rappresenta un OGGETTO generico. 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 statiche.
  - **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 dato (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 contengono un valore (come le variabili) che rappresenta una caratteristica dell'oggetto a cui la proprietà è riferita.
- Inizializzando una varibile le assegno un tipo (o classe)

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

- E posso accedere alle sue proprietà.

# 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.

Mentre invece:

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

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

# METODI

- I metodi contengono codice eseguibile (come le funzioni), e ci consentono di eseguire azioni sull'oggetto.
- 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. Normalmente quando è previsto il passaggio di parametri i metodi modificano l'oggetto quando restituiscono valore restituiscono informazioni sull'oggetto.

# ACCESSO A UN METODO

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

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

il metodo **getDate** restituisce il giorno del mese della data contenuta in **pippo** che viene memorizzato nella variabile **giorno**.

- Quando un metodo prevede dei parametri normalmente serve a modificare l'oggetto:

```
pippo.setDate(3);
```

Modificherò la data contenuta nella variabile **pippo:** il giorno del mese sarà 3.

# 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**. **document** è una variabile riservata che contiene l'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 e *subiti* dagli elementi interattivi.
- Come per gli altri oggetti l'accesso alle proprietà e ai metodi avviene attraverso l'operatore di appartenenza (.)
- Se, per esempio, voglio recuperate l'elemento con l'attributo id = mio_id:

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

# COSTRUCTOR

- È la particolare metodo 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");
```

# NUMBER

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

constructor

# 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 |

# replace()

- Il metodo replace () cerca in una stringa i gruppi di caratteri che corrispondono a una substringa o a una *regular expression*, e restituisce una nuova stringa in cui le occorrenze trovate vengono sostituite da un valore specificato,
- Nota: Se si usa una substring e non una regular expression per la ricerca, verrà sostituita solo la prima occorrenza trovata. Per sostituire tutte le occorrenze di una ricerca è necessario usare una regular expression con il modificatore g,
- Questo metodo non cambia la stringa originale.

# split()

- Il metodo split () viene utilizzato per dividere una stringa in un array di stringhe utilizzanda uno o più caratteri come separatore, e restituisce il nuovo array.
- Se si utilizza una stringa vuota ("") come separatore, la stringa è diviso nei singoli caratteri che la compongono.
- Nota: Il metodo split () non modifica la stringa originale.

# 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 ricerca globale (trova tutte le ricorrenze (match), altrimenti trova solo la prima. |
| 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 |

# 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 |

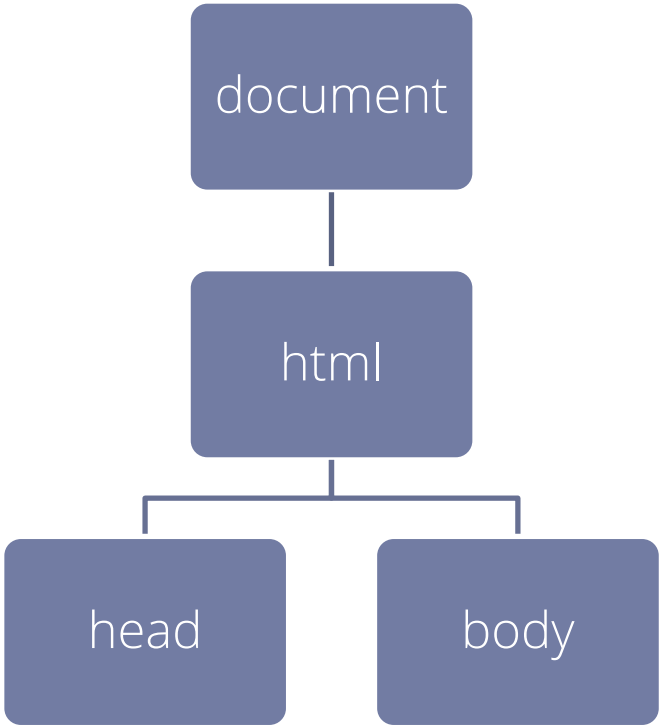
# DOCUMENT OBJECT MODEL

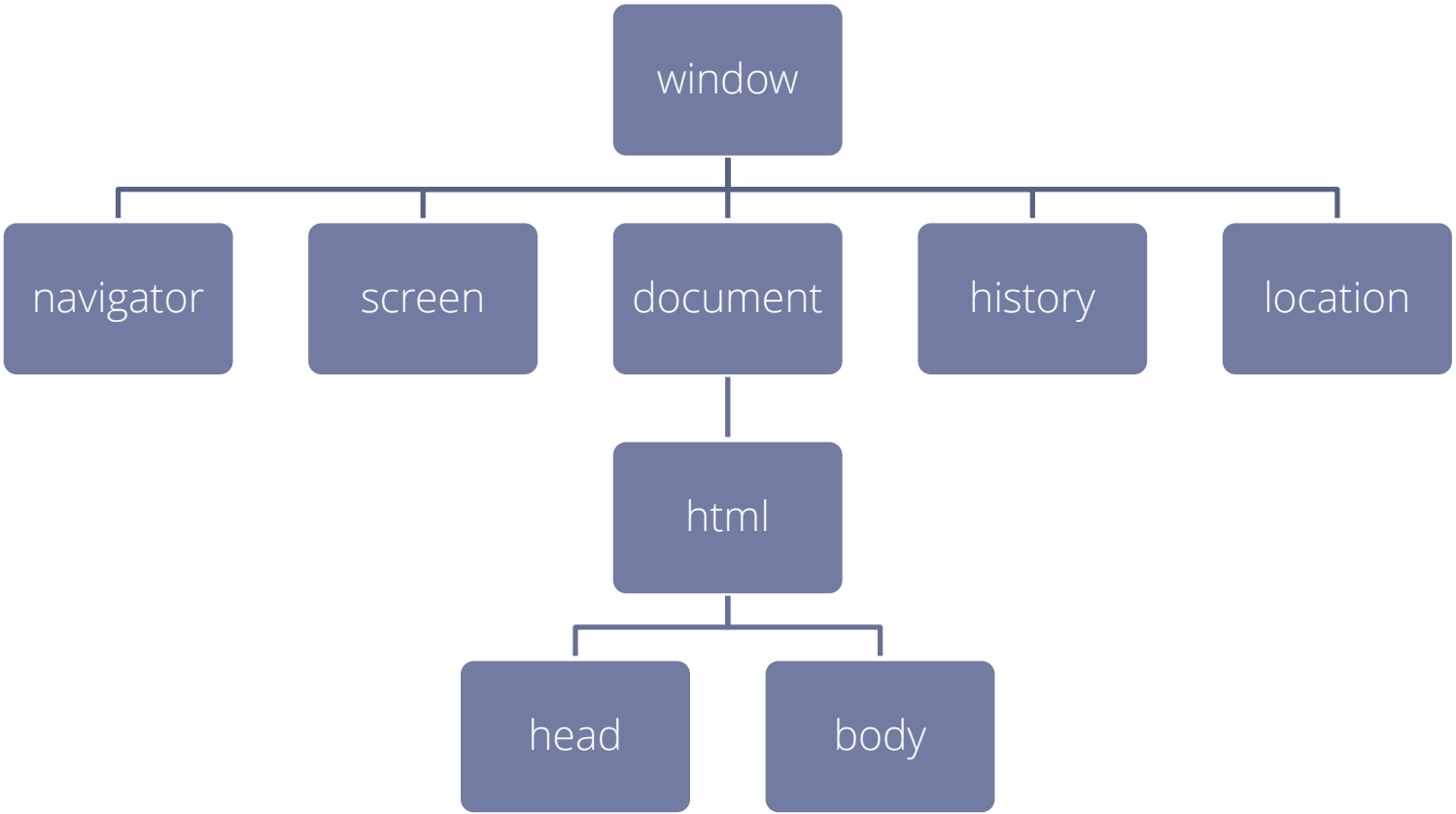
# DOM

- HTML (e XHTML) hanno la funzione di strutturare in una rigida gerarchia i contenuti di una pagina WEB
- Quando i browser caricano il contenuto di una pagina organizzano quindi questi contenuti in memoria in una struttura gerarchica ben definita utilizzando una architettura ad oggetti.
- 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

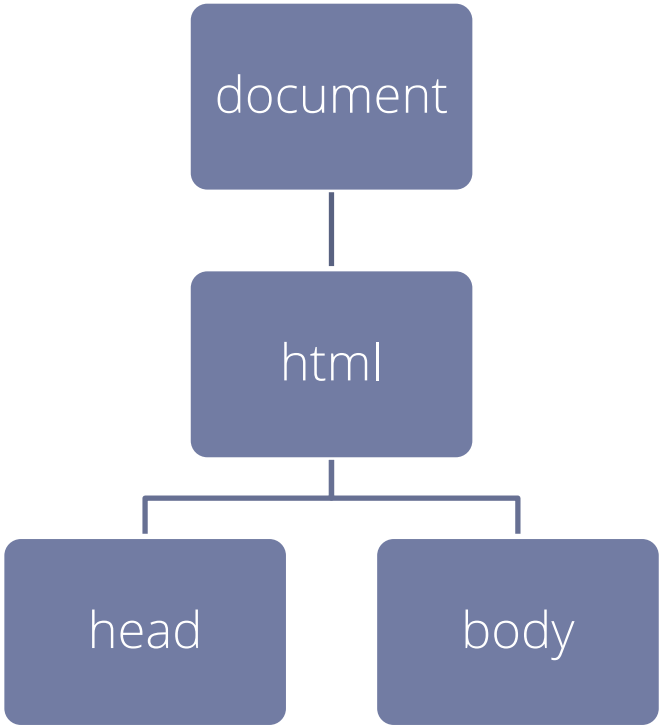
- L'oggetto location rappresenta l'url da cui è stata caricata la pagina
- La sua funzione principale è quella di caricare una pagina diversa nella corrente finestra o frame.
- Allo script è consentito di accedere ad informazioni solo sulla url da cui è stato caricato.

# 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.

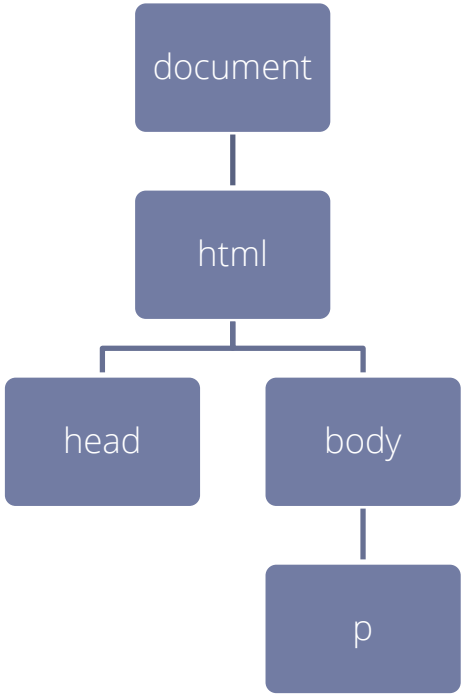
# Documento vuoto

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



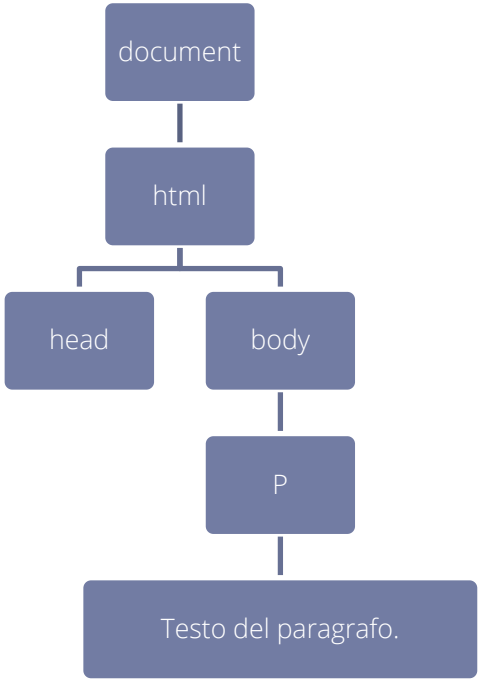
# AGGIUNTA DI UN PARAGRAFO VUOTO

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



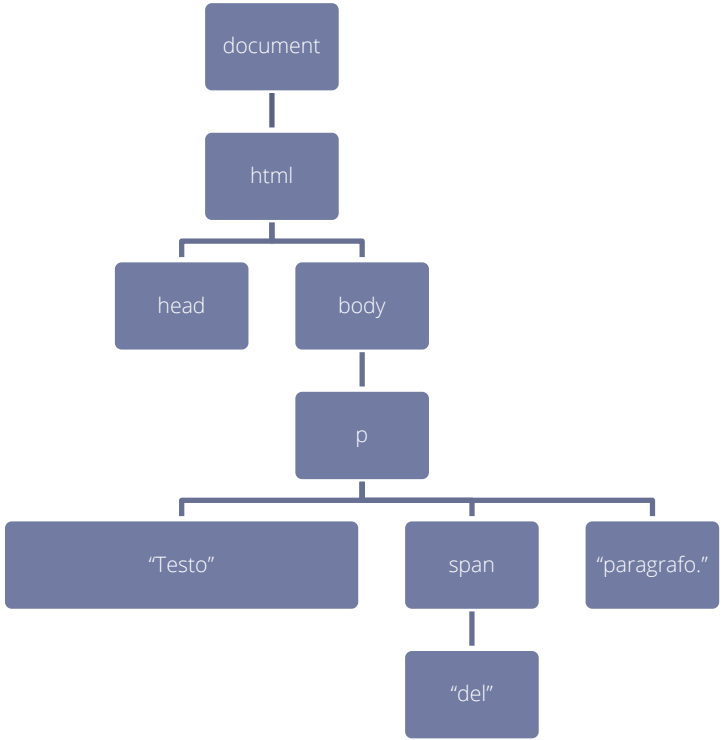
# AGGIUNTA DI TESTO AL PARAGRAFO

```
<html>
   <head></head>
   <body>
        <p>Testo delparagrafo.</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

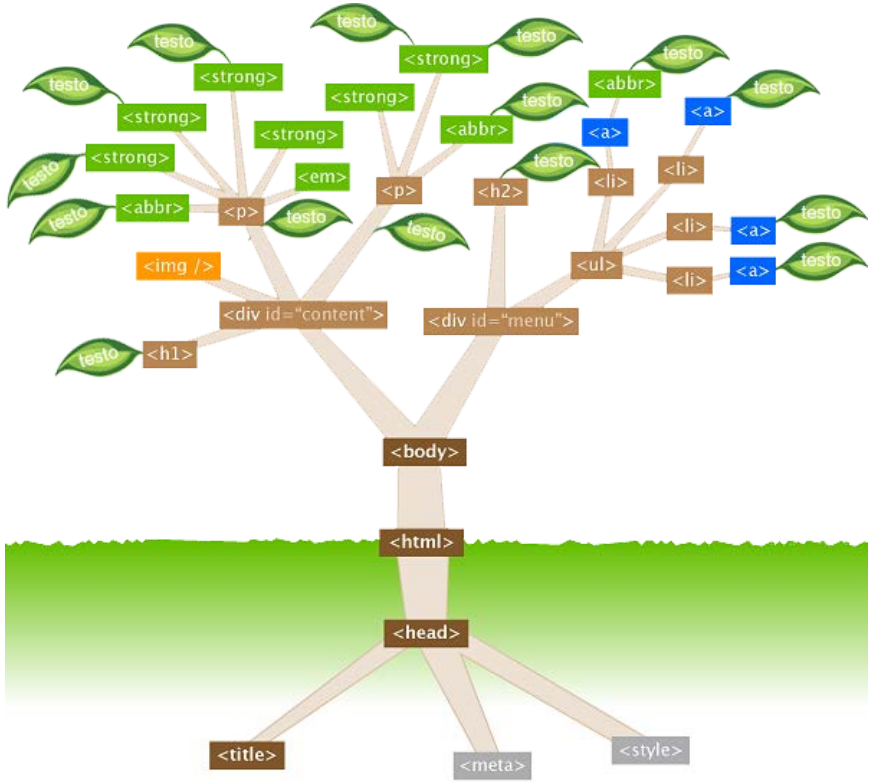
- Javascript agisce sul DOM modificando, eliminando e aggiungendo oggetti.
- Per agire sul DOM lo script deve interagire con qualcuno dei nodi presenti nella struttura ad albero:
  - Per modificarlo
  - Per aggiungere testo
  - Per aggiungere un figlio ecc.
- Avrà bisogno di un riferimento unico al nodo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

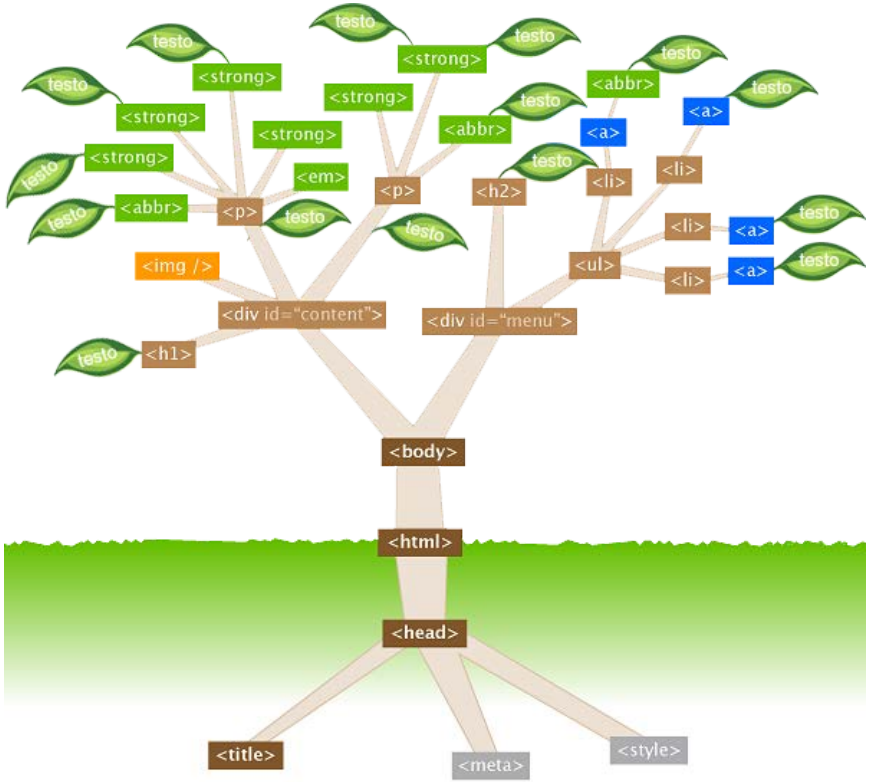
- Per poter essere utilizzato facilmente in uno script l'ID di un oggetto deve seguire alcune regole:
  - non può contenere spazi
  - non devono contenere segni di punteggiatura tranne che per il carattere di sottolineatura (es.: primo_paragrafo)
  - deve essere racchiuso tra virgolette quando viene 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



# 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:

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

# RECUPERARE GLI ELEMENTI

- Il metodo **getElementsByTagName (tag)** restituisce un insieme di tutti gli elementi del documento con il nome **tag** specificato.
- Viene restituito un oggetto NodeList che in sostanza si comporta come un Array. I nodi sono accessibili, ad esempio, attraverso l'indice che rispetta l'ordine con cui gli elementi appaiono nella pagina. L'indice parte da 0.
- Passando il valore * come parametro vengono restituiti tutti gli elementi del documento.
- È possibile utilizzare la proprietà length dell'oggetto NodeList per determinare il numero di elementi con il nome tag specificato, e scorrerli con un ciclo for o while.
- La sintassi è:

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

- Se scrivo:

```
var elem_array= document.getElementsByTagName("a");
```

- **elem_array** conterrà tutti gli elementi **a** presenti nella pagina, **elem_array.length** mi dirà quanti sono, **elem_array[0]** conterrà il primo elemento, **elem_array[1]** il secondo e così via.

# RECUPERARE GLI ELEMENTI

- Il metodo **getElementsByClassName(nomeClasse)** restituisce un insieme di tutti elementi del documento che hanno la classe nomeClasse.
- Viene restituito un oggetto NodeList che in sostanza si comporta come un Array. I nodi sono accessibili, ad esempio, attraverso l'indice che rispetta l'ordine con cui gli elementi appaiono nella pagina. L'indice parte da 0.
- È possibile utilizzare la proprietà length dell'oggetto NodeList per determinare il numero di elementi con il nome tag specificato, e scorrerli con un ciclo for o while.
- NodeList è un oggetto dinamico. Eliminando la classe nomeClasse dall'elemento l'elemento viene eliminato dalla lista.
- 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.
- Esempi:

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

# ELABORARE GLI ELEMENTI

- innerHTML
  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>";
var 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);
```

# CLASS

- La proprietà **classList** restituisce la lista delle classi di un elemento, come un oggetto DOMTokenList.
- Questa proprietà è utile per aggiungere e rimuovere le classi CSS di un elemento.
- La proprietà classList è di sola lettura, tuttavia, è possibile modificarla applicando i metodi .add (), .remove () e toggle().
- **Nota**: La proprietà classList non è supportato in IE9 e precedenti.

# classList

| Metodo | Descrizione |
|---|---|
| **add(*class1, class2, ...*)** | Aggiunge una o più classi a un elemento |
| **contains(*class*)** | Restituisce true se la lista contiene la classe specificata altrimenti false. |
| **item(*index*)** | Restutische l'elemento della lista corrispondente a index. |
| **remove(*class1, class2, ...*)** | Elimina una o più classi dalla lista |
| **toggle(*class*)** | Se la classe esiste viene eliminata e il metodo restituisce false, altriementi vene aggiunta e restituisce true. |

# className

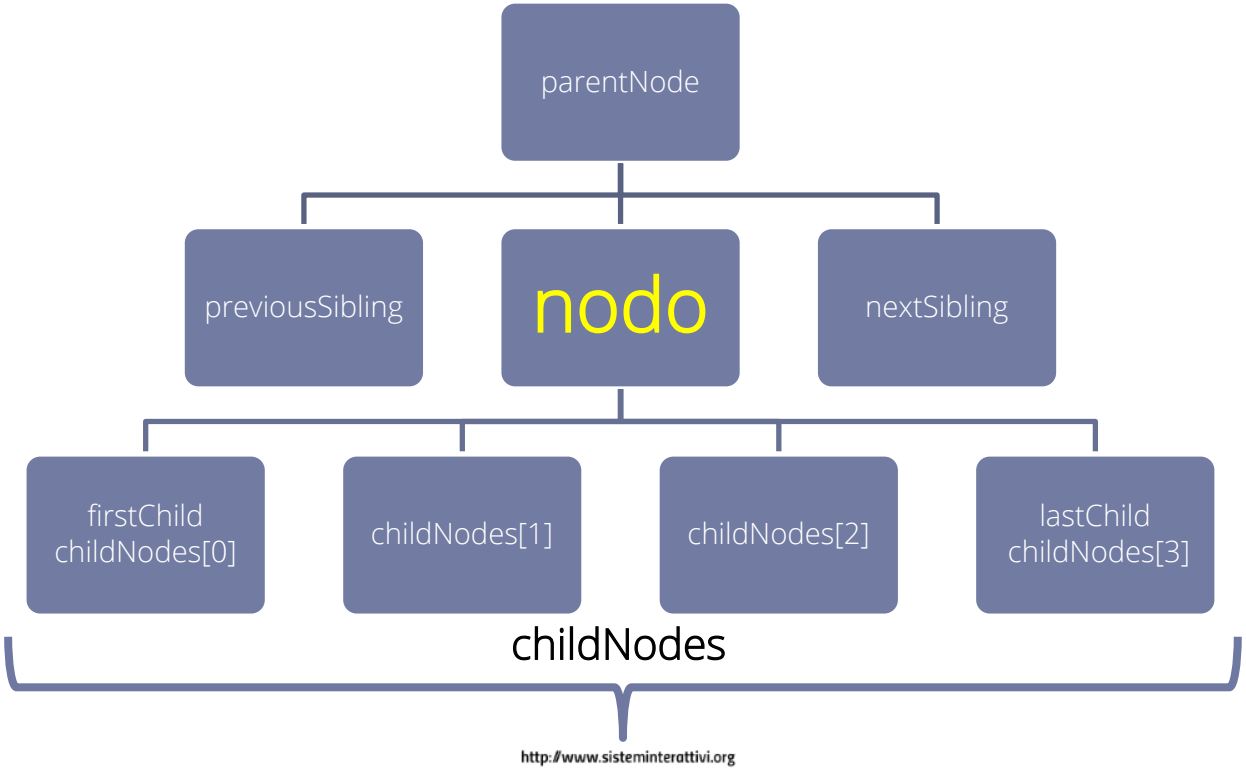
- La proprietà **className** restituisce o imposta il contenuto dell'attributo class di un elemento.
- Questa proprietà è utile per aggiungere e rimuovere le classi CSS di un elemento.
- Costituisce l'aternativa a **classList** per i browser che non la supportano..

# STYLE

- La proprietà **style** viene utilizzata per leggere o impostare lo stile in-line di un elemento.
- Nota: La proprietà **style** è un oggetto. Le varie regole CSS si assegnano modificando, aggiungendo e togliendo una proprietà "CSS" allo stile e specificare un valore:
  - `element.style.backgroundColor = "red";`
- La sintassi JavaScript per l'impostazione delle proprietà CSS è leggermente diverso da quello CSS (backgroundColor invece di background-color).
- La proprietà style restituisce solo le dichiarazioni CSS in-line, impostate nell'attributo **style** dell'elemento. Non è possibile utilizzare questa proprietà per ottenere informazioni sulle regole di stile dalla sezione <head> del documento o fogli di stile esterni.

# 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.
- Il nodi di testo (il contenuto di testo degli elementi) vengono restituiti come textNode separati

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

# RELAZIONE TRA NODI

- children
  proprietà che restituisce una nodeList di riferimenti agli elementi che discendono direttamente dal nodo corrente, compreso il testo che contengono.
- Gli elementi sono nello stesso ordine in cui appaiono nella pagina.

```
var nodiFigli = nodo.children;
```

# 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,figlio);`

- Esempio:

```
var nuovo = document.createElementi("p");
nuovo.innerHTML = "Ciao sono un nuovo paragrafo";
nodo.insertBefore(nuovo, nodo.children[0]);
// Inserisce un nuovo paragrafo come primo elemento nel contenitore
```

# 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

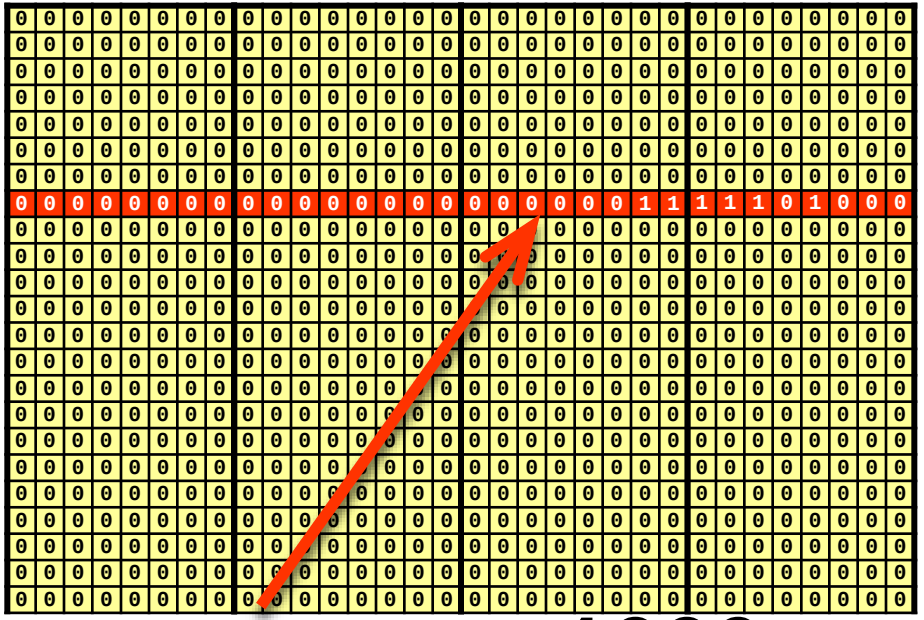
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

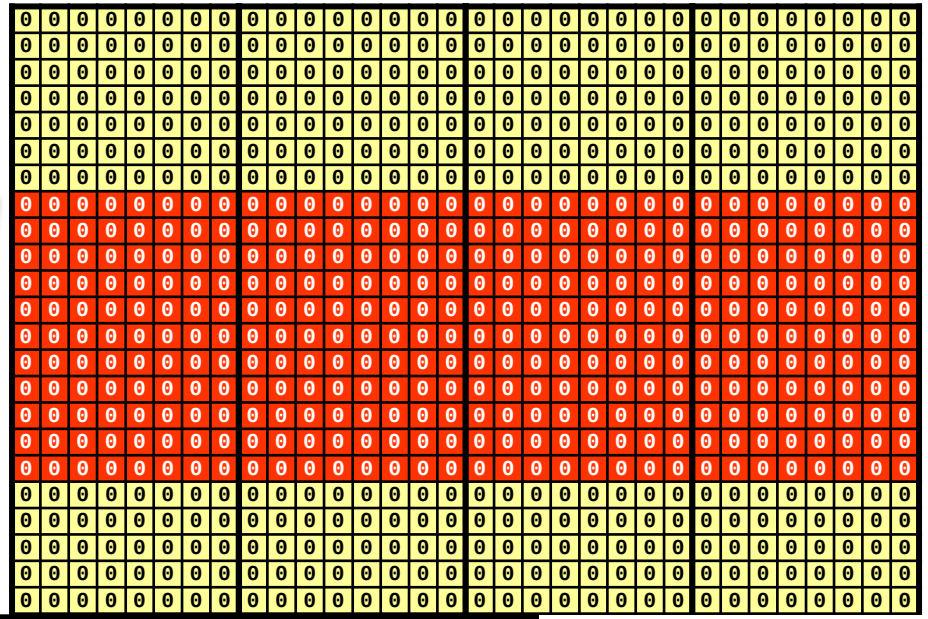
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

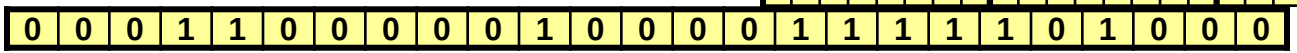
# Valori e puntatori

```
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.