JAVASCRIPT

Programmazione iterativa, Introduzione al DOM

PROGRAMMAZIONE ITERATIVA

LA PROGRAMMAZIONE ITERATIVA

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

while

Il ciclo **while**

- Il ciclo while ripete un blocco di codice fino a quando una condizione specificata è vera.

Sintassi

```
while (condizione) {
    //blocco di codice da eseguire
}
```

while: esempio 1

Questo ciclo continua fino a che **i** è minore di **10**

```
function myFunction() {
    var i = 0;
    while (i < 10) {
        var text =    "<br>Il numero è " + i;
        document.getElementById("demo").innerHTML += text;
        i++;
    }
    document.getElementById("demo").innerHTML += "<br>Il ciclo è finito!";
}
```

!!! ATTENZIONE !!!

Se si dimentica di aumentare la variabile usata nella condizione, il ciclo diventerà infinito . Questo manderà in crash il browser.

while: esempio 1

Questo ciclo continua di **10**

Inizializzazione della variabile di controllo

```
function myFunction() {
    var i = 0;
    while (i < 10) {
        var text =    "<br>Il numero è " + i;
        document.getElementById("demo").innerHTML += text;
        i++;
    }
    document.getElementById("demo").innerHTML += "<br>Il ciclo è finito!";
}
```

!!! ATTENZIONE !!!

Se si dimentica di aumentare la variabile usata nella condizione, il ciclo diventerà infinito . Questo manderà in crash il browser.

while: esempio 1

Questo ciclo continua fino a che **i** è minore di **10**

Verifica della condizione

```
function myFuncti
    var i = 0;
    while (i < 10) {
        var text =    "<br>Il numero è " + i;
        document.getElementById("demo").innerHTML += text;
        i++;
    }
    document.getElementById("demo").innerHTML += "<br>Il ciclo è finito!";
}
```

!!! ATTENZIONE !!!

Se si dimentica di aumentare la variabile usata nella condizione, il ciclo diventerà infinito . Questo manderà in crash il browser.

while: esempio 1

Questo ciclo continua fino a che **i** è minore di **10**

```
function myFunction() {
    var i = 0;
    while (i
        var
        docume
        i++;
    }
    document.getElementById("demo").innerHTML += "<br>Il ciclo è finito!";
}
```

Modifica della variabile

!!! ATTENZIONE !!!

Se si dimentica di aumentare la variabile usata nella condizione, il ciclo diventerà infinito . Questo manderà in crash il browser.

while: esempio 2

```
/*
 Creo una node listi che contiene tutti gli elementi della pagina
 HTML che hanno la classe out
*/
var elementi = document.getElementsByClassName("out");

// Il ciclo continua fino a che la nodeList elementi contiene item
while (elementi.length > 0){
      elementi[0].classList.remove("out");
      // Eliminando la classe "out" dall'insieme delle classi
      // dell'elemento elimino anche l'elemento dalla collezione
}
```

while: esempio 2

Inizializzazione della variabile

```
/*
 Creo una no                contiene tutti gli elementi della pagina
 HTML che ha                out
*/
var elementi = document.getElementsByClassName("out");

// Il ciclo continua fino a che la nodeList elementi contiene item
while (elementi.length > 0){
      elementi[0].classList.remove("out");
      // Eliminando la classe "out" dall'insieme delle classi
      // dell'elemento elimino anche l'elemento dalla collezione
}
```

while: esempio 2

```
/*
 Creo una node listi che contiene tutti gli elementi della pagina
 HTML che hanno la classe out
*/
var elementi = do                    mentsByClassName("out");

// Il ciclo continu  fino a che la nodeList elementi contiene item
while (elementi.length > 0){
      elementi[0].classList.remove("out");
      // Eliminando la classe "out" dall'insieme delle classi
      // dell'elemento elimino anche l'elemento dalla collezione
}
```

Verifica della condizione

while: esempio 2

```
/*
 Creo una node listi che contiene tutti gli elementi della pagina
 HTML che hanno la classe out
*/
var elementi = document.getElem                ame("out");

// Il ciclo continua fino a che la nodeList elementi contiene item
while (elementi.length > 0){
      elementi[0].classList.remove("out");
      // Eliminando la classe "out" dall'insieme delle classi
      // dell'elemento elimino anche l'elemento dalla collezione
}
```

Modifica della variabile

for

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

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

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

for: esempio

```
/*
    Creo un certo numero (espresso dalla varibiale quanti)
    di numeri casuali e li aggiungo ad un stringa (str) separandoli
    con una virgola e uno spazio
*/
var str = "";                                   // Inizializzazione della varibile str
for (var i = 0; i < quanti; i++) {
    var num = Math.round(Math.random() * 100); // Numero random tra 0 e 100
    if ( i > 0) {
        str = str + ", ";                           // Aggiungo virgola e spazio
    }
    str = str + num;                                // Aggiungo il numero generato a str
}
// Mostro sullo schermo i numeri generati utilizzando l'elemento con id  "casualiGenerati"
document.getElementById("casualiGenerati").innerHTML = "Numeri generati: " + str;
```

for: esempio

```
/*
     Creo u                    presso dalla varibiale quanti)
     di num                    ggiungo ad un stringa (str) separandoli
     con un                    azio
*/
var str = ""                                    // Inizializzazione della varibile str
for (var i = 0; i < quanti; i++) {
     var num = Math.round(Math.random() * 100); // Numero random tra 0 e 100
     if ( i > 0) {
          str = str + ", ";                          // Aggiungo virgola e spazio
     }
     str = str + num;                                // Aggiungo il numero generato a str
}
// Mostro sullo schermo i numeri generati utilizzando l'elemento con id  "casualiGenerati"
document.getElementById("casualiGenerati").innerHTML = "Numeri generati: " + str;
```

Inizializzazione della variabile di controllo

for: esempio

```
/*
     Creo un certo numero (espresso dalla varibiale quanti)
     di numeri casual                          un stringa (str) separandoli
     con una virgola
*/
var str = "";                                        // Inizializzazione della varibile str
for (var i = 0; i < quanti; i++) {
     var num = Math.round(Math.random() * 100); // Numero random tra 0 e 100
     if ( i > 0) {
          str = str + ", ";                              // Aggiungo virgola e spazio
     }
     str = str + num;                                    // Aggiungo il numero generato a str
}
// Mostro sullo schermo i numeri generati utilizzando l'elemento con id  "casualiGenerati"
document.getElementById("casualiGenerati").innerHTML = "Numeri generati: " + str;
```

Verifica della condizione

for: esempio

```
/*
     Creo un certo numero (espresso dalla varibiale quanti)
     di numeri casuali e li ag                    ga (str) separandoli
     con una virgola e uno spa
*/
var str = "";                                   // Inizializzazione della varibile str
for (var i = 0; i < quanti; i++) {
     var num = Math.round(Math.random() * 100); // Numero random tra 0 e 100
     if ( i > 0) {
          str = str + ", ";                          // Aggiungo virgola e spazio
     }
     str = str + num;                                // Aggiungo il numero generato a str
}
// Mostro sullo schermo i numeri generati utilizzando l'elemento con id  "casualiGenerati"
document.getElementById("casualiGenerati").innerHTML = "Numeri generati: " + str;
```

Modifica della variabile

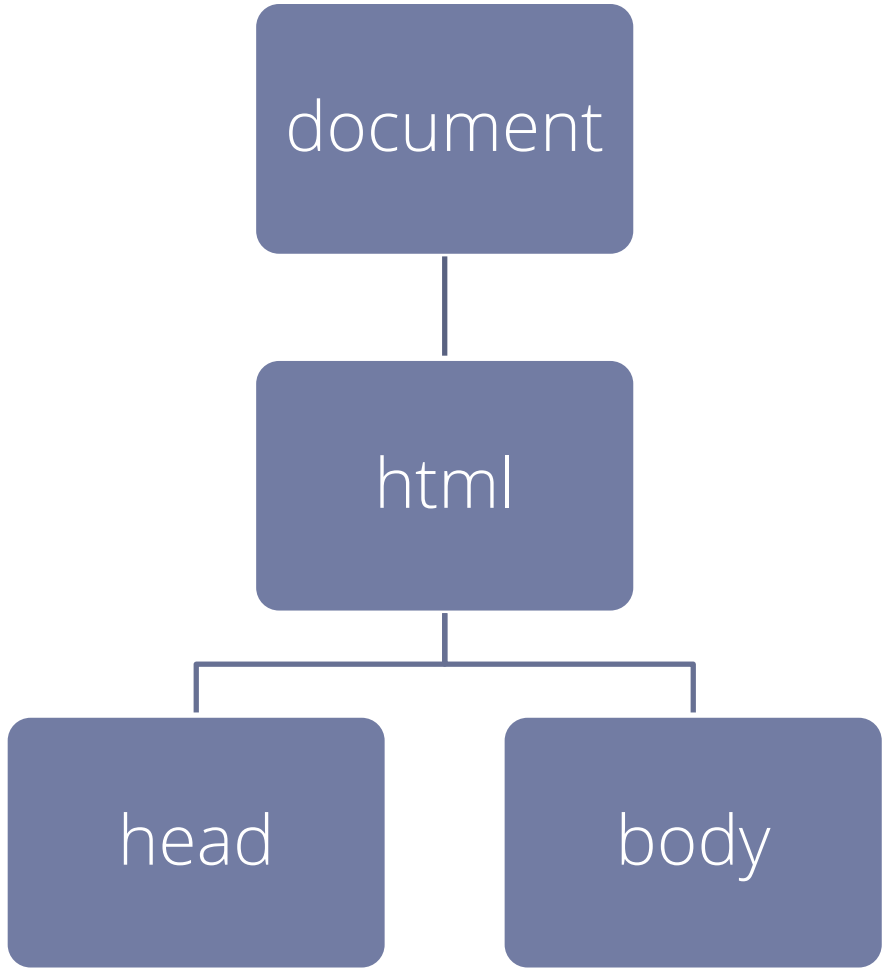
DOCUMENT OBJECT MODEL

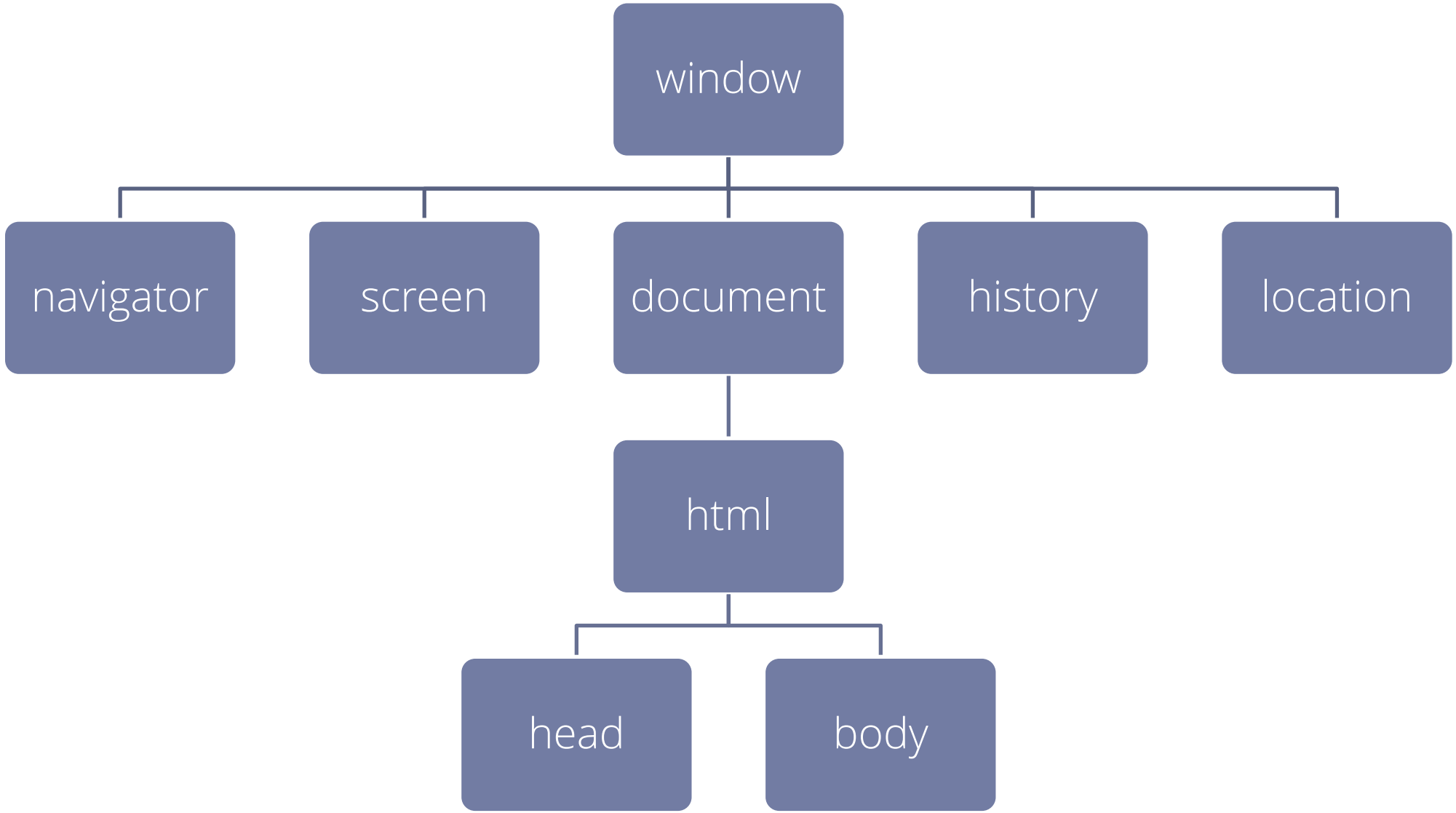
D O M

- HTML ha 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 rappresenta 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.

SCOPE

SCOPE

- In JavaScript, scope (*l'ambito*) è l'insieme delle variabili, degli oggetti e delle funzioni a cui è posso accedere in un determinato punto di un programma.
- Esistono solo due scope:
  - locale
  - globale

SCOPE LOCALE

- Appartengono allo scope locale tutte le variabili, funzioni e oggetti che creo all'interno del codice di una funzione.
- Tutti questi elementi saranno accessibili solo dal codice della funzione stessa.

SCOPE GLOBALE

- Appartengono allo scope globale tutte le variabili, funzioni e oggetti che non creo all'interno del codice di una funzione, ma in qualsiasi altro punto del codice Javascript in una pagina web,
- Tutti questi elementi saranno accessibili a tutto il codice che appartiene alla pagina web.

L'OGGETTO WINDOW

- In una pagina web l'ambito globale in cui è racchiuso il programma Javascript è l'oggetto window che rappresenta la finestra del browser in cui gira il programma Javascript.
- Ogni funzione, oggetto o variabile globale può essere anche scritta come proprietà dell'oggetto predefinito window.
- Se scrivo:

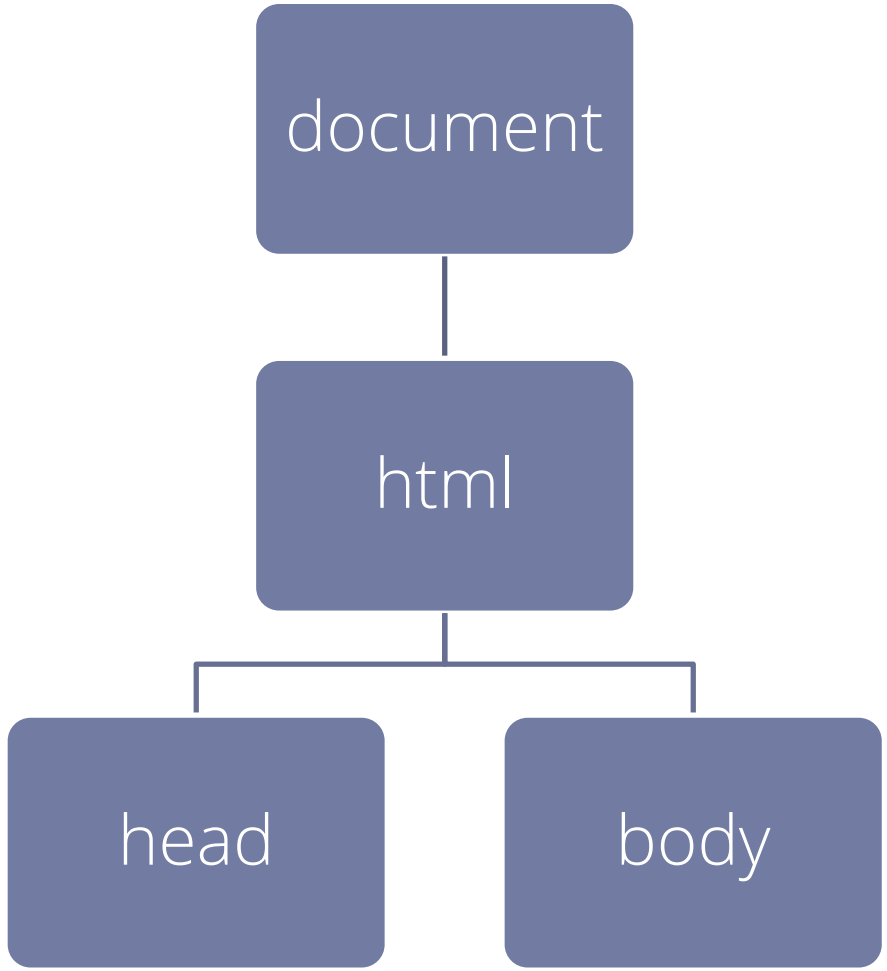
```
var frullatore = "Frullatore rosso";
```

- Nel codice posso indifferentemente usare **`frullatore`** o **`window.frullatore`**

DOCUMENT

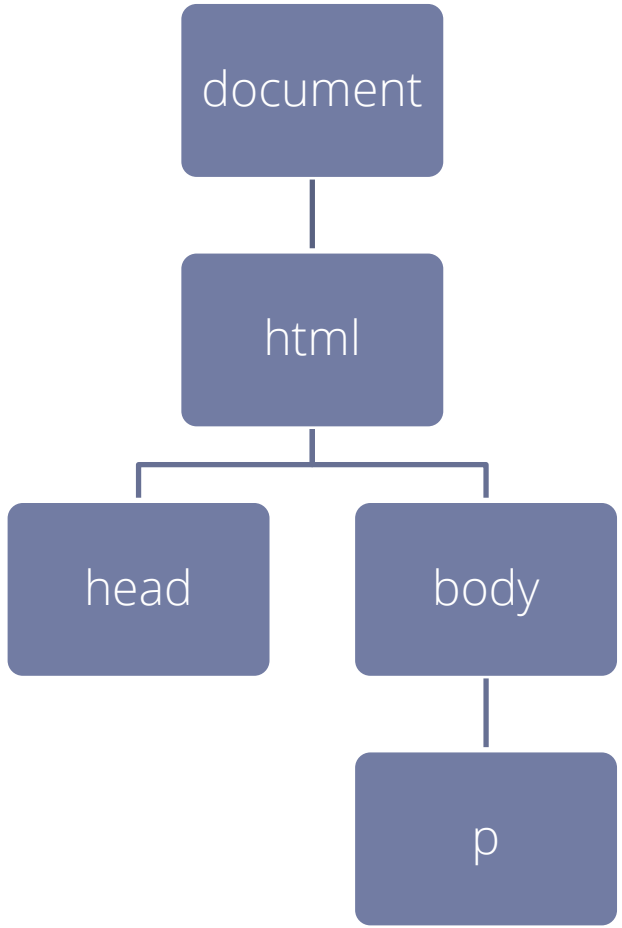
DOCUMENTO VUOTO

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



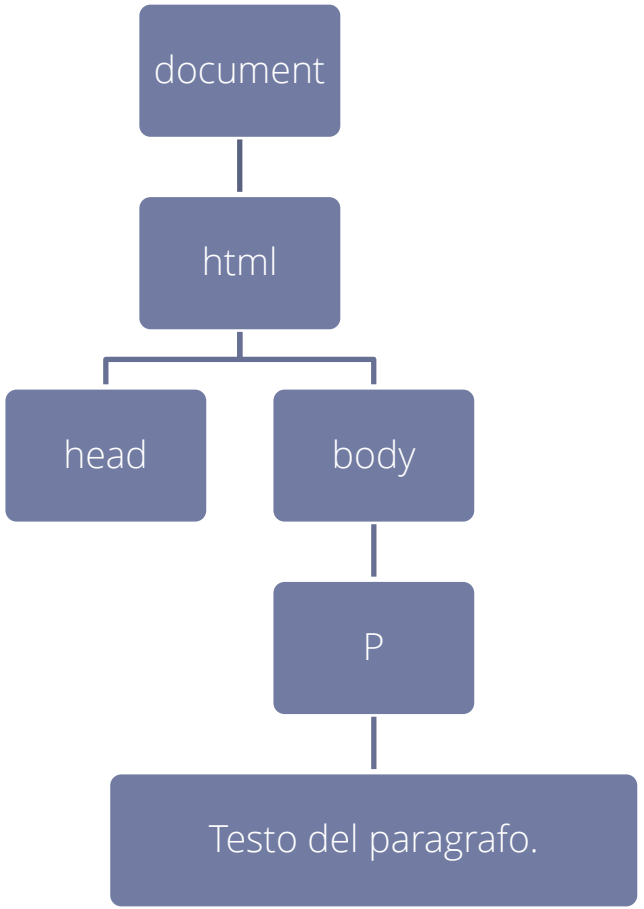
AGGIUNTA DI UN PARAGRAFO VUOTO

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



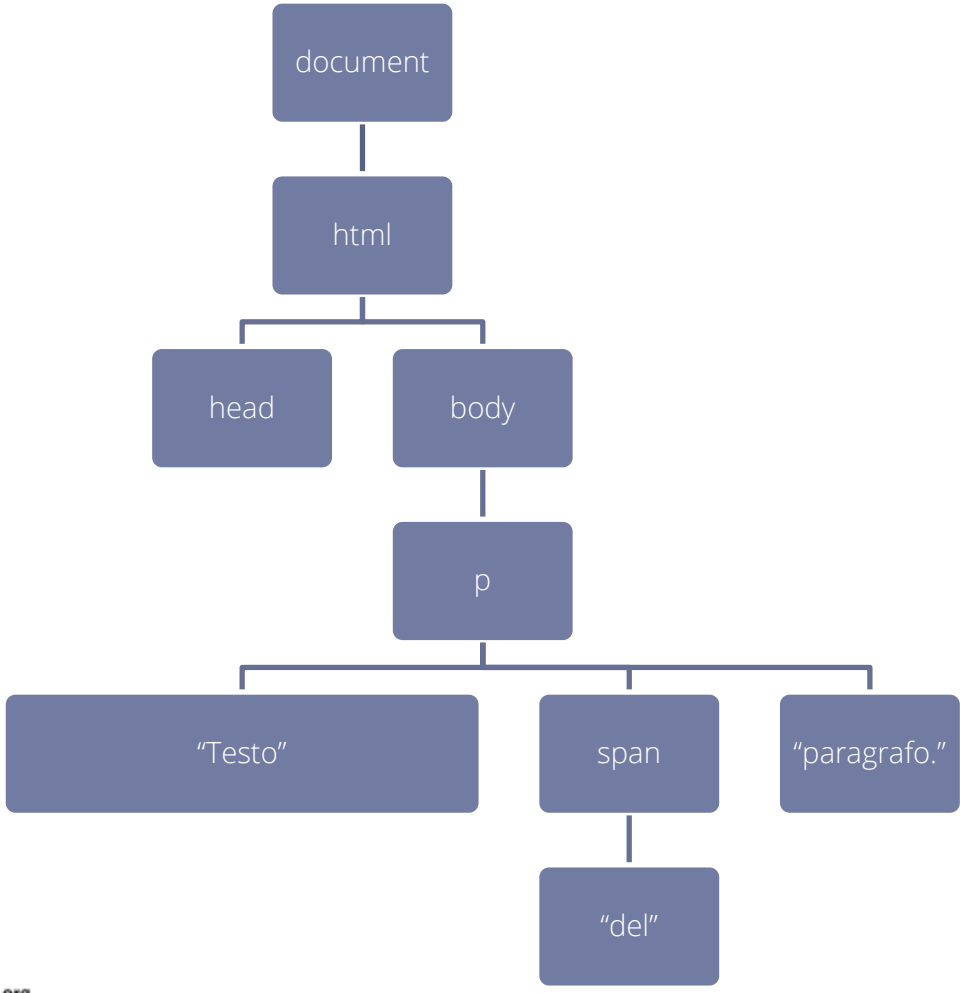
AGGIUNTA DI TESTO AL PARAGRAFO

```
<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.

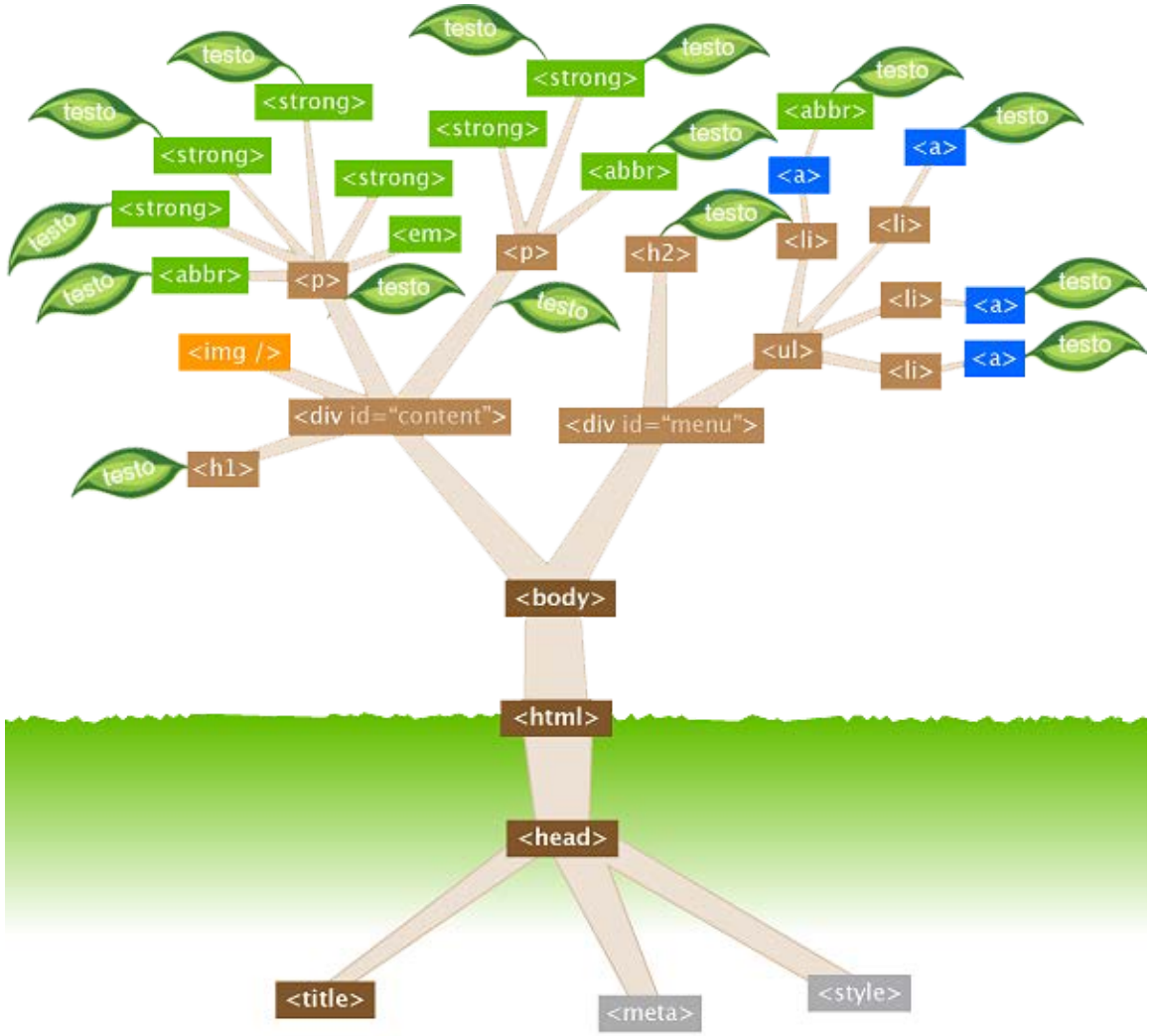
```
<p id="primoParagrafo" ></p>
<img id="logo" src="images/logo.jpg" alt="Logo Azienda">
<div class="header" id="header"></div>
```

DARE UN NOME AD UN NODO

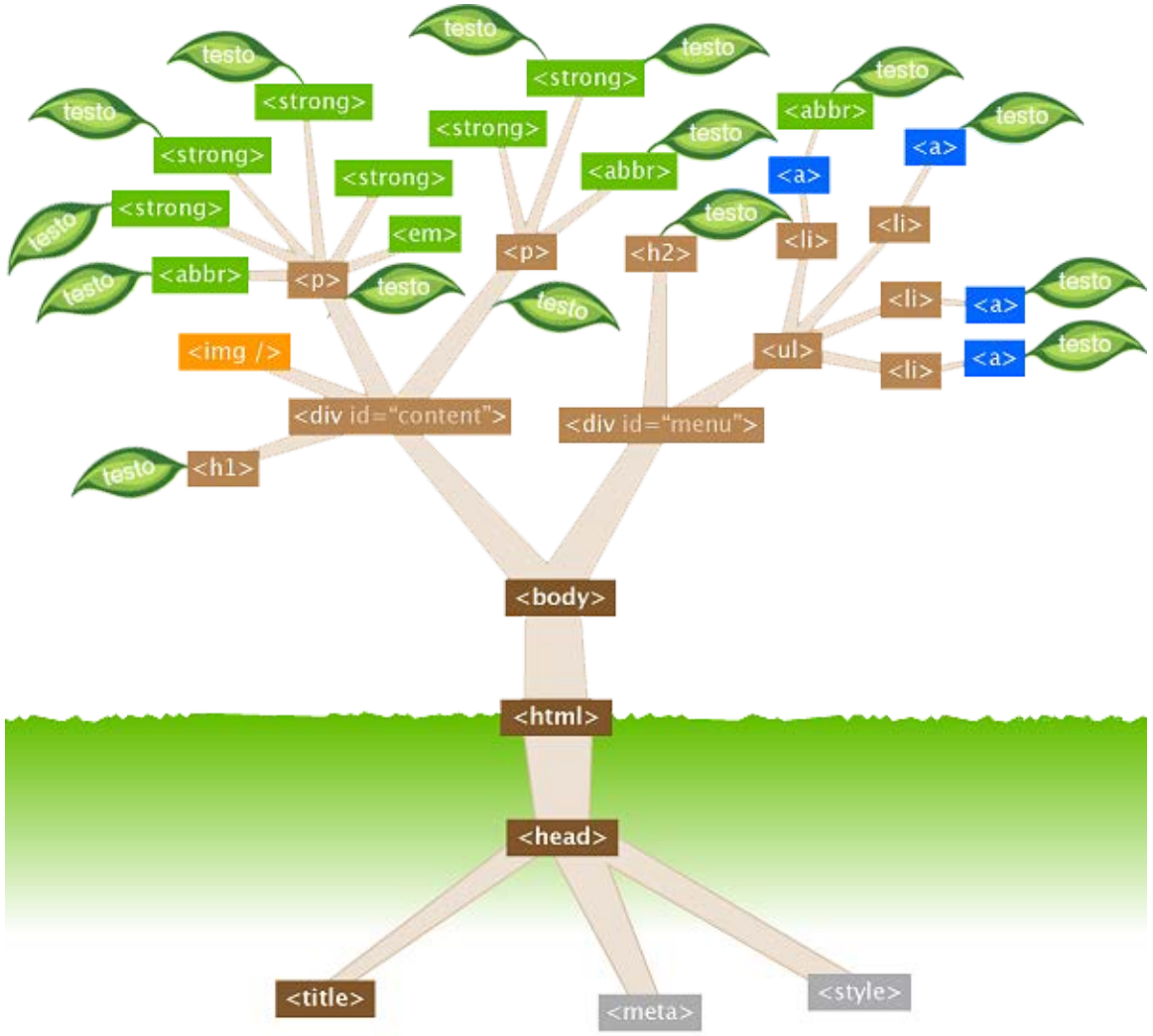
- Per poter essere utilizzato facilmente in uno script l'ID di un oggetto deve seguire alcune regole:
  - non può contenere spazi
  - non devono contenere segni di punteggiatura tranne che per il carattere di sottolineatura (es.: primo_paragrafo)
  - Deve essere unico all'interno dello stesso documento

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:

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

getElementsByTagName(tag)

- Il metodo 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.
- 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.

getElementsByClassName(nomeClasse)

- Il metodo 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, 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.

```
var elem_array= document.getElementsByTagName(nomeClasse);
```

CREARE UN ELEMENTO

- `createElement(tagName)`
  Il metodo crea un nuovo elemento di qualunque tipo. Restituisce un riferimento al nuovo elemento creato.

```
var elemento = document.createElement(nomeTag);
```

CREARE NODO DI TESTO

- **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");
```

L'OGGETTO ELEMENT

Principali proprietà e metodi dell'elemento HTML

innerHTML

- Proprietà che restituisce o imposta il codice HTML compreso tra il tag di apertura e il tag di chiusura dell'elemento a cui è applicata.
- Esempio:

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

`classList`

- La proprietà restituisce la lista delle classi di un elemento.
- Questa proprietà è utile per aggiungere e rimuovere le classi CSS di un elemento.
- La proprietà **`classList`** può essere modificato solo applicando i metodi **`.add()`**, **`.remove ()`** e **`toggle()`**.

Nota bene

- La proprietà **`classList`** non è supportato in IE9 e precedenti.

Metodi dell'oggetto **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)`	Restituisce 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.

Uso di `classList`

```
window.onload = function () {
    /* elementi è un collezione di elementi HTML che hanno la classe out */
    var elementi = document.getElementsByClassName("out");
    while (elementi.length > 0){
        /* Quando elimino la classe out dalla classList dell'elemento,
           l'elemento stesso viene eliminato dalla collezione */
        elementi[0].classList.remove("out");
        /* Quando tutti gli elementi sono eliminati dalla collezione
           elementi.length diventa 0 e il ciclo si interrompe*/
    }
};
```

`className`

- La proprietà **`className`** restituisce o imposta il contenuto dell'attributo **`class`** di un elemento.
- Questa proprietà è utile per modificare le classi CSS di un elemento.
- Costituisce l'alternativa 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 cui proprietà rappresentano le varie proprietà CSS. Le varie regole di stile si assegnano modificando le proprietà dell'oggetto.:

```
document.getElementById('pippo').style.backgroundColor = "red";
```

- In JavaScript le proprietà CSS che indicano le caratteristiche grafiche dell'elemento vengono trasformate secondo lo stile camel case (`background-color` diventa `backgroundColor`).
- La proprietà style restituisce solo le dichiarazioni CSS impostate nell'attributo **style** dell'elemento. Non è possibile utilizzare questa proprietà per ottenere informazioni sulle regole di stile impostate nella sezione <head> del documento o in fogli di stile esterni.

RELAZIONE TRA GLI ELEMENTI