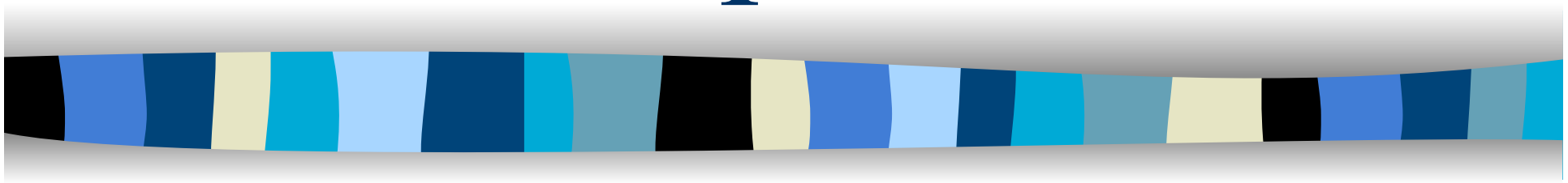
Javascript – 3

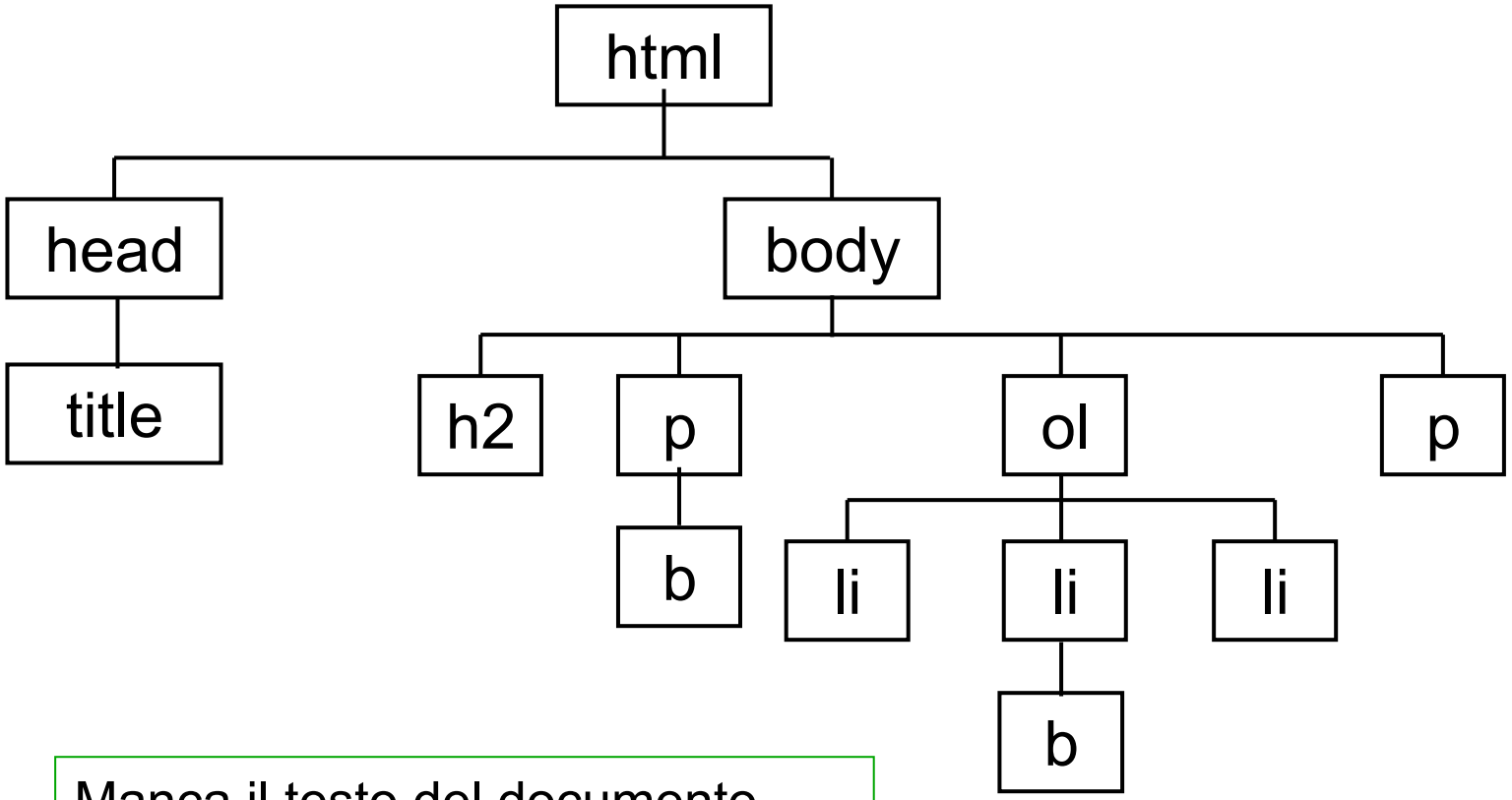


DOM – Document Objet Model

Albero del documento

- La chiave per capire il DOM è capire come un documento HTML sia modellato come un albero
- Possiamo rappresentare ogni documento HTML attraverso un albero
  - Alla radice c'è il nodo che rappresenta il tag **HTML**, i suoi figli sono i nodi che rappresentano **HEAD** e **BODY**
- DOM level 1 fornisce un API per accedere ai nodi dell'albero

Documento HTML visto come un albero

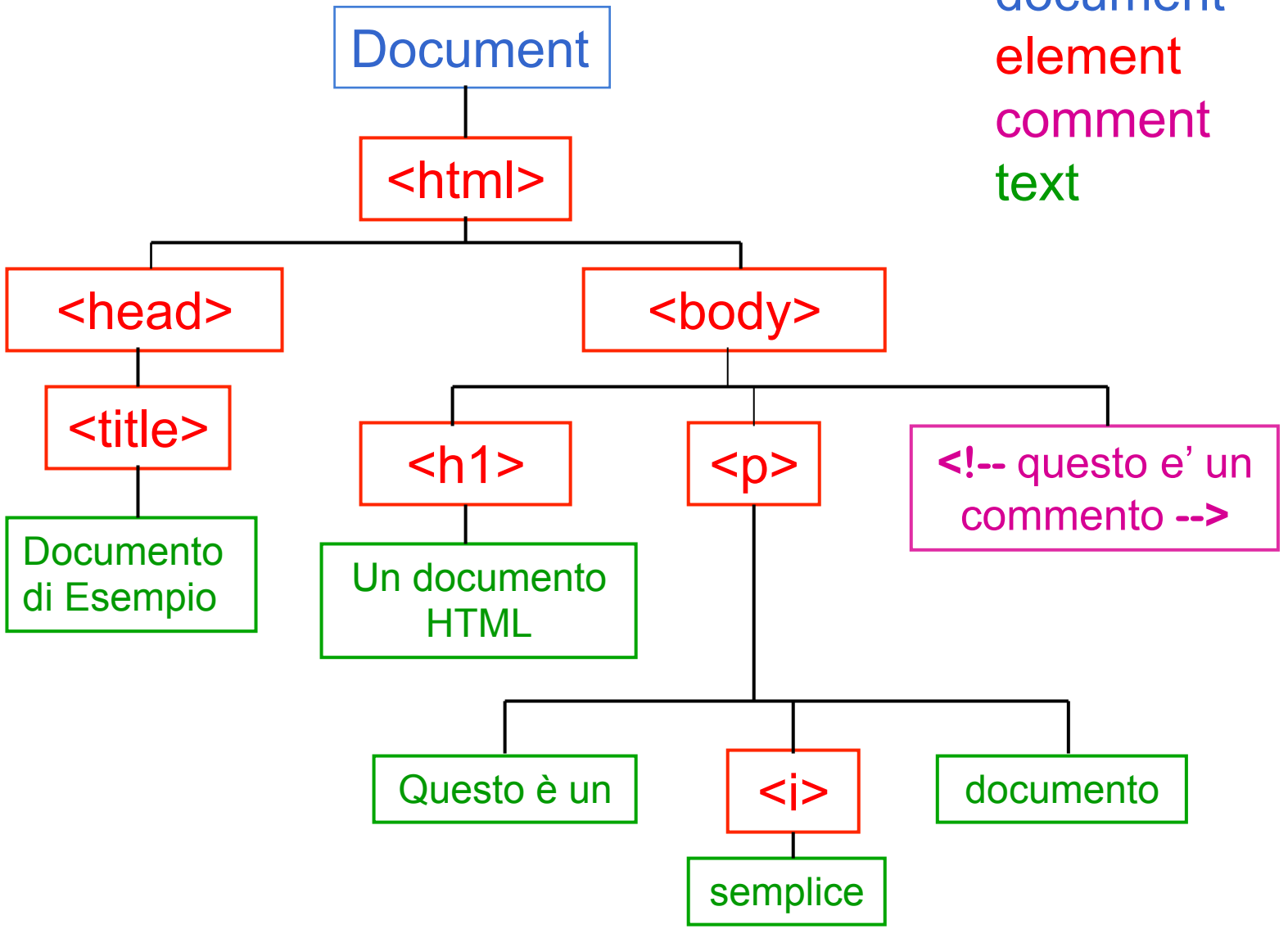


Manca il testo del documento.
C'è solo la struttura

Esempio – HTML

```
<html>
  <head>
      <title>Documento di Esempio</title>
  </head>
  <body>
      <h1>Un documento HTML </h1>
      <p> Questo &egrave; un <i>semplice</i>
          documento</p>
      <!-- questo e' un commento -->
  </body>
</html>
```

HTML visto come albero



Nodi dell'albero

- Ogni elemento dell'albero è chiamato genericamente nodo
  - Rappresentato tramite un oggetto di tipo **Node**
- L'interfaccia **Node** definisce proprietà e metodi per attraversare e manipolare i nodi dell'albero del documento
- Ogni nodo dell'albero è di un tipo predefinito, all'interno dell'oggetto **Node** c'è la proprietà nodeType che specifica di che tipo è il nodo

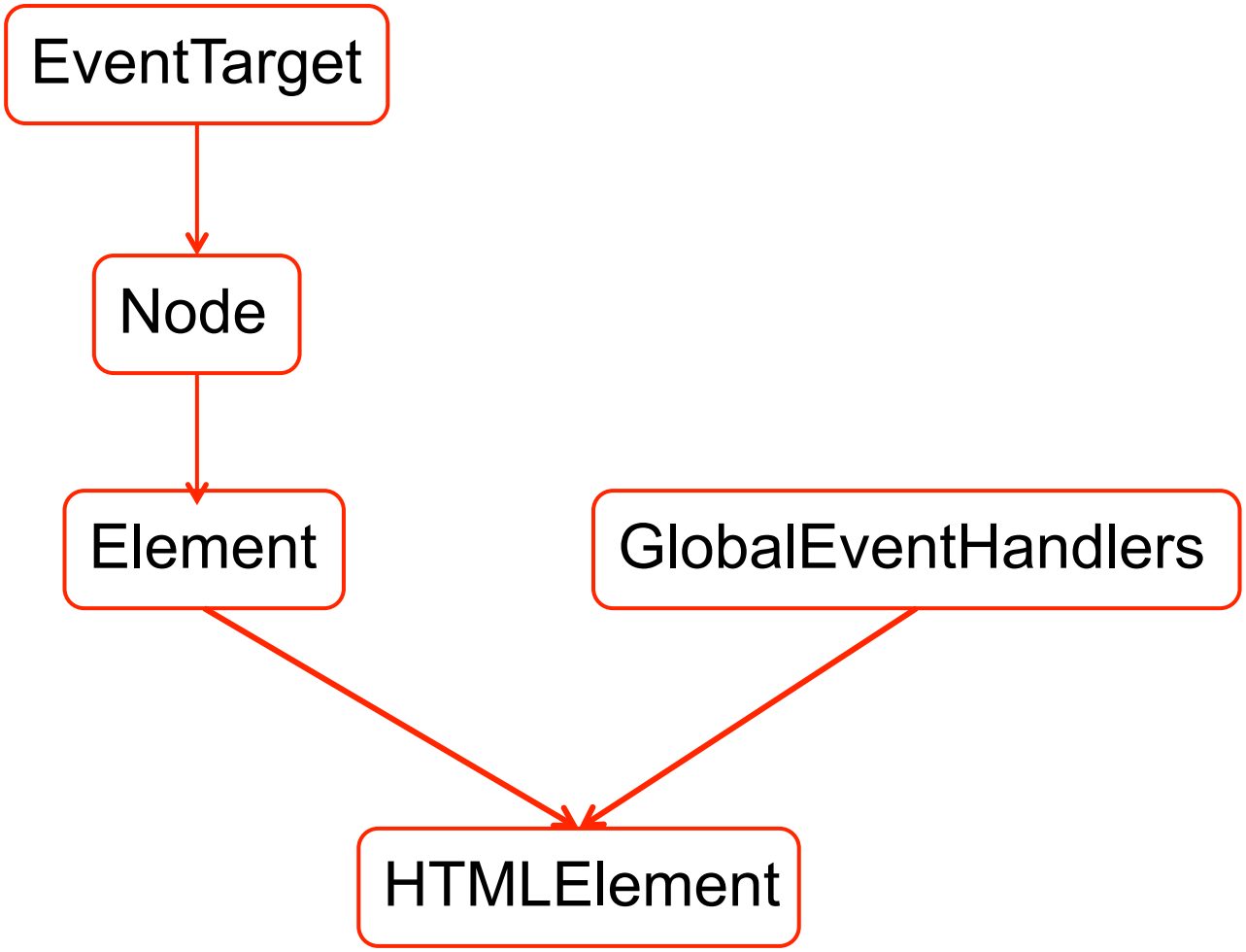
Tipi di nodo più comuni

- Il tipo di ogni nodo può essere identificato attraverso una costante

Interfaccia	Costante nodeType	Valore
Element	**Node.ELEMENT_NODE**	1
Text	**Node.TEXT_NODE**	3
Document	**Node.DOCUMENT_NODE**	9
Comment	**Node.COMMENT_NODE**	8
Attr	**Node.ATTRIBUTE_NODE**	2
DocumentType	**Node.DOCUMENT_TYPE_NODE**	10

Nodi ed interfacce

- Se la proprietà nodeType di un nodo è uguale a Node.ELEMENT_NODE (assume il valore 1) vuol dire che il nodo implementa le interfacce **Node** ed **Element**
  - possiamo usare su di esso tutti i metodi e proprietà delle interfacce Element e Node


EventTarget
Node
Element
GlobalEventHandlers
HTMLElement

Navigare nel DOM

- Una volta ottenuto un riferimento ad un nodo possiamo navigare nel sottoalbero associato al nodo tramite un insieme di metodi e proprietà
- Il modo più semplice per ottenere un riferimento ad un nodo dell'albero di un documento è tramite

  document.getElementById(elementID)
- Per poter utilizzare tale metodo dobbiamo associare un nome univoco ad ogni tag nel documento HTML attraverso l'attributo **ID**

Proprietà e metodi di Node – 1

- parentNode
  - Restituisce un riferimento al padre del nodo corrente, se tale nodo esiste
- childNodes
  - Restituisce un array contenente tutti i figli del nodo corrente
- firstChild
  - Restituisce un riferimento al primo figlio del nodo corrente, se tale nodo esiste
- lastChild
  - Restituisce un riferimento all'ultimo figlio del nodo corrente, se tale nodo esiste

Proprietà e metodi di Node – 2

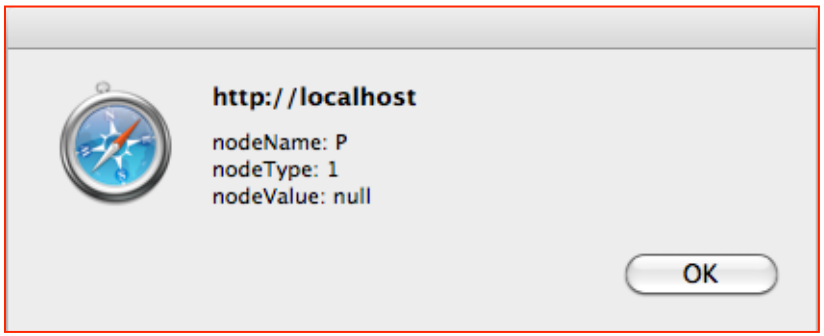
- previousSibling
  - Restituisce un riferimento al fratello precedente del corrente nodo
- nextSibling
  - Restituisce un riferimento al fratello successivo del corrente nodo
- hasChildNodes()
  - Restituisce true se il nodo corrente ha dei nodi figli

Altre proprietà di Node

- nodeName
  - Contiene il nome del nodo (il tag, P, EM, TD, …)
- nodeValue
  - Contiene il valore del nodo, si applica generalmente solo a nodi di testo
- nodeType
  - Costante che specifica il tipo di nodo
- attributes
  - Elenco degli attributi dell'elemento
- ownerDocument
  - Puntatore all'oggetto **Document** in cui l'elemento corrente è contenuto

Esempio

```
<body>
<p id="p1">Un esempio di un<em>paragrafo</em>di
   testo</p>
<script type="text/javascript">
var nodo = document.getElementById('p1');
var msg = "nodeName: "+nodo.nodeName+"\n";
msg += "nodeType: "+nodo.nodeType+"\n";
msg += "nodeValue: "+nodo.nodeValue+"\n";
alert(msg);
</script>
</body>
```



Nota

- Il motivo per cui nodeValue è null deriva dal fatto che dobbiamo esaminare un nodo di tipo **Text** per poter "vedere" il testo all'interno
- La stessa cosa vale per il nodo associato al tag **EM**, esso è di tipo Node.ELEMENT_NODE, mentre il suo unico figlio è di tipo Node.TEXT_NODE e contiene la stringa "paragrafo"

Esempio: contatore

```
<body style="font-size:large" onload="conta()">
So contare da uno fino a dieci. Guarda:
<span id="cnt">1</span>
</body>
<script type="text/javascript">
var i=1;
function conta() {
if (i < 11) {
    var elem = document.getElementById("cnt");
    elem.firstChild.nodeValue = i++;
    setTimeout(conta,500); } //Dopo mezzo secondo
                                    // invoca nuovamente conta
} </script>
```

Accedere a collezioni di elementi

- Tramite alcuni metodi di document (ereditati anche da Element) possiamo accedere a collezioni di elementi
  - Restituiti come lista di nodi (NodeList)
  - Possiamo far riferimento agli elementi restituiti usando anche la notazione degli array
- Possiamo applicare i seguenti metodi a document o ad un qualsiasi nodo di tipo HTMLElment dell'albero

Metodi di document per collezioni di tag – 1

- getElementsByName('nome')
  - Restituisce tutti gli elementi che hanno l'attributo NAME settato a nome
- getElementsByTagName('tag')
  - Restituisce tutti gli elementi di tipo tag
- getElementsByClassName('classe')
  - Restituisce tutti gli elementi che hanno l'attributo CLASS settato a classe

Metodi di document per collezioni di tag – 2

- querySelector('selector')
  - Restituisce il primo elemento che corrisponde al selettore CSS selector
- querySelectorAll('selector')
  - Restituisce tutti gli elementi che corrispondono al selettore CSS selector

```
var elems = document.querySelectorAll("p > img.apple")
```

```
var elems = document.querySelectorAll("p, img#apple")
```

Esempio

- Per ottenere una lista dei riferimenti a tutti i paragrafi del documento

```
paragrafi = document.getElementsByTagName('p');
```

- Per ottenere una lista dei riferimenti a tutti i paragrafi contenuti nell'elemento con ID=x3c

```
radice = document.getElementById('x3c');

paragrafi = radice.getElementsByTagName('em');
```

Accesso agli elementi

- Per accedere all'i-esimo elemento si usa la stessa notazione degli array
  - paragrafi[i-1]
- Invece di accedere a paragrafi[i-1] si può accedere a
  - paragrafi.item(i-1)
- I metodi presentati in precedenza restituisco una lista di nodi NodeList
  - Proprietà: length //numero di elementi
  - Metodo: item

Modificare il documento

- Esistono dei metodi e proprietà per modificare il documento
- Si possono aggiungere/rimuovere nodi dell'albero che rappresenta il documento
- Frammenti di codice HTML possono essere inseriti in qualsiasi punto del documento
- Si possono cancellare porzioni di documento

Creare elementi

- document.createElement(nomeTag)
  - Crea un elemento del tipo specificato dal parametro nomeTag
  - Restituisce un nodo di tipo **HTMLElement**
- document.createTextNode(str)
  - Crea un nodo testuale contenente la stringa str passata come parametro
  - Restituisce un nodo di tipo **Text**

Inserire nodi nell'albero

- elem.appendChild(HTMLElement)
  - Inserisce un elemento come ultimo figlio di elem (nodo corrente)
- elem.insertBefore(<newElem>, <childElem>)
  - Inserisce l'elemento newElem come figlio di elem prima dell'elemento childElem
- elem.insertAdjacentHTML(<pos>, <text>)
  - Inserisce il codice HTML rappresentato da text relativamente al nodo corrente (elem) secodo quanto indicato da pos

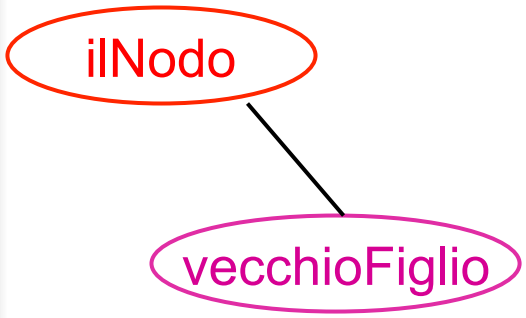
Valori per pos

```
<!-- beforebegin -->
<p>
<!-- afterbegin -->
foo
<!-- beforeend -->
</p>
<!-- afterend -->
```

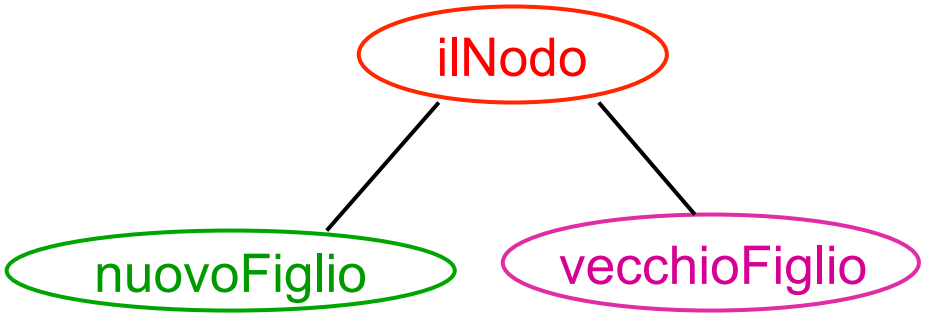
- 'beforebegin'
  - Prima dell'elemento stesso
- 'afterbegin'
  - Subito dopo l'inizio dell'elemento e prima del suo primo figlio
- 'beforeend'
  - Subito prima della fine dell'elemento e dopo del suo ultimo figlio
- 'afterend'
  - Dopo dell'elemento stesso

```
var nuovoNodo = document.createElement('p');
var testo = document.createTextNode('Ciao mondo');
nuovoNodo.appendChild(testo);
var elemento = document.getElementById('rd3');
elemento.appendChild(nuovoNodo);
```

```
ilNodo. insertBefore(nuovoFiglio, vecchioFiglio)
```



prima



dopo

Inserire HTML nell'albero

- innerHTML
  - Legge o scrive il contenuto dell'elemento
- outerHTML
  - Legge o scrive l'intero elemento (tag e contenuto)

```
<table border="1">
  <thead><tr><th>Fruit</th><th>Color</th></tr></thead>
  <tbody>
        <tr id="applerow"><td>Plum</td><td>Purple</td></tr>
  </tbody>
</table>
```

```
document.getElementById("applerow").innerHTML
```

```
<td>Plum</td><td>Purple</td>
```

```
document.getElementById("applerow").outerHTML
```

```
<tr id="applerow"><td>Plum</td><td>Purple</td></tr>
```

Confrontare elementi

- elem.isEqualNode(HTMLElement)
  - Restituisce true se HTMLElement e elem sono uguali
    - Tipo uguale, attributi uguali, figli uguali nel medesimo ordine
- elem.isSameNode(HTMLElement)
  - Restituisce true se HTMLElement e elem si riferiscono allo stesso nodo

Rimuovere nodi

- elem.removeChild(HTMLElement)
  - Cancella l'elemento HTMLElement figlio di elem

```
var figlio = document.getElementById('p1');
var genitore = figlio.parentNode;
genitore.removeChild(figlio);
```

- elem.replaceChild(NEW_elem, OLD_elem)
  - Sostituisce il figlio OLD_elem di elem con l'elemento NEW_elem

Copiare nodi

- elem.cloneNode(true)
  - Restituisce una copia dell'elemento elem e di tutti i suoi figli
- elem.cloneNode(false)
  - Restituisce una copia del solo elemento elem
  - I figli di elem vengono ignorati

Manipolare il testo

- Possiamo modificare direttamente il contenuto di un nodo di tipo testo andando a modificare la proprietà data oppure nodeValue
- Ci sono un gran numero di metodi come

  appendData(), deleteData(), insertData(), replaceData(), splitText(), e replaceWholeText()

  che possono essere usati per modificare nodi di tipo testo, ma poiché la proprietà data (nodeValue) è una stringa si possono usare direttamente i metodi dell'oggetto **String**

Aggiungere del testo

- Per aggiungere una stringa al nodo testo di tipo **Text** è possibile usare il metodo appendData()
- Esempio

```
nodo = document.getElementById("p1");
testo = nodo.firstChild;
testo.appendData("Testo aggiunto");
```

DOM ed HTML

- Abbiamo visto che JavaScript può manipolare in qualsiasi modo un arbitrario elemento HTML attraverso il DOM
- Ciò richiede che il documento sia *ben formato* altrimenti i risultati possono essere imprevedibili
- Non solo JavaScript ma anche Java può manipolare documenti HTML attraverso gli stessi metodi
- Gli stessi metodi possono essere anche usati per manipolare documenti XML

Proprietà di HTMLElement

- Tutti gli elementi derivano da questa classe. Le proprietà di base sono
  - id
  - title
  - lang
  - dir
  - className (dettagli in seguito)
- Sono accessibili in lettura e scrittura.
- Corrispondo ai rispettivi attributi globali HTML (className corrisponde a class)

Altre proprietà di HTMLElement

- Altre proprietà che sono accessibili dipendono dell'elemento. Ad esempio
  - checked
  - classList (dettagli in seguito)
  - disabled
  - hidden
  - spellcheck
  - tabIndex
  - tagName

Lavorare con le classi

- Possiamo manipolare lo stile di un elemento modificando il valore associato alla proprietà style che corrisponde all'attributo HTML style
  - Dettagli in seguito con DOM Level 2
- Per il momento, modificheremo le classi CSS associate ad un elemento tramite le proprietà
  - className e classList

Proprietà className

- Utilizzata per aggiungere facilmente una classe ad un elemento

```
document.getElementById("textblock").className += " newclass";
```

spazio

- È complicato rimuovere una classe dalla lista di classi associata ad un elemento

Proprietà classList - 1

- Restituisce un oggetto DOMTokenList
- Proprietà/metodi
  - elem.classList.length
    - Restituisce il numero di classi dell'elemento elem
  - elem.classList.add(<class>)
    - Aggiunge la classe class all'elemento elem
  - elem.classList.contains(<class>)
    - Restituisce true se l'elemento elem appartiene alla classe class

Proprietà classList - 1

- elem.classList.remove(<class>)
  - Rimuove la classe class dall'elemento elem
- elem.classList.toggle(<class>)
  - Aggiunge la classe class all'elemento elem se non è già presente, altrimenti la rimuove

Lavorare con gli attributi degli elementi

- Ci sono proprietà per gli attributi globali
- Altri attributi sono supportati (in lettura/ scrittura) tramite altre proprietà e metodi dell'oggetto HTMLElement
- attributes
  - È un array di oggetti tipo Attr
  - Ogni elemento rappresenta un attributo
  - Ha due proprietà
    - name (nome dell'attributo)
    - value (valore dell'attributo)

```
<p id="textblock" class="fruit numbers" lang="it">
    Lorem ipsum dolor sit amet, consectetur adipiscing elit.
   Nulla eu diam non magna varius varius.
</p>

<p id="results"></p>
```

```
var results = document.getElementById("results");
var elem = document.getElementById("textblock");
var attrs = elem.attributes;
for (var i = 0; i < attrs.length; i++) {
     results.innerHTML += " Nome: " + attrs[i].name;
     results.innerHTML += " Valore: " + attrs[i].value + "<br/>";
}
```

```
Name: id Value: textblock
Name: class Value: fruit numbers
Name: lang Value: it
```

Lavorare con gli attributi

- elem.getAttribute(<name>)
  - Restituisce il valore dell'attributo name di elem oppure null se elem non ha quell'attributo
- elem.hasAttribute(<name>)
  - Restituisce true se elem ha l'attributo name
- elem.removeAttribute(<name>)
  - Cancella l'attributo name; restituisce true se l'operazione è avvenuta con successo
- elem.setAttribute(<name>, <value>)
  - Setta il valore dell'attributo name a value

DOM e moduli – 1

- Ogni elemento in un modulo (controllo) è rappresentato da un oggetto che ne espone le proprietà.
- Proprietà/metodi comuni
  - value, name,
  - form, type
  - onfocus, onblur
  - focus(), blur()

DOM e moduli – 2

- Altre proprietà dipendono dal tipo di controllo
- checkbox – radiobutton
  - checked, defaultChecked
  - onclick, click()
- button, submit, reset
  - onclick
- text, password, textarea
  - onchange, select()

DOM e moduli – 2

- select
  - onchange, options[], selectedIndex
- option
  - selected, defaultSelected, text
- form
  - onsubmit, onreset, submit(), reset()
  - action, method
  - elements[]

```
<select name="prov" onChange="controlla(this.value); ">
```

Attivato quando l'utente seleziona un elemento della lista di selezione

this è un riferimento all'oggetto su cui è invocata controlla, in questo caso è la lista di selezione di nome prov

```
<input type="submit"  value="mostra"
        onClick="return mostra(); ">
```

Se la funzione mostra restituisce false l'azione prevista dalla pressione di submit (invio del modulo) è annullata

Gestori di eventi e moduli

- onchange
  - Attivato quando l'utente lascia una casella di testo
- onkeydown
  - Attivato quando l'utente preme un tasto
- onkeyup
  - Attivato quando l'utente un rilascia il tasto
- onkeypress
  - Attivato tra onkeydown e onkeyup
- ondblclick
  - Attivato quando l'utente preme due volte il tasto sinistro del mouse

Costruttori ed elementi

- Oggetti di tipo **Option** possono essere costruiti dinamicamente, come segue, con il costruttore Option()

```
new Option(text, value, defaultSelected, selected)
```

boolean

```
var a = new Option("salerno", 1, false, false);
var b = new Option("napoli", 2, true, false);
```

Parametri di Option()

- text
  - Specifica il testo che verrà mostrato all'utente
- value
  - Specifica il valore che sarà inviato al server
- defaultSelected (valore booleano)
  - Indica se l'opzione è selezionata quando l'oggetto è creato (corrisponde all'attributo SELECTED)
- selected (valore booleano)
  - Indica se l'opzione corrente è selezionata o meno

```
<FORM NAME="cantina">
        <SELECT NAME ="vini">
                <OPTION value="1"> Taurasi
                <OPTION value="2"> Greco di Tufo
        </SELECT>
</FORM>
```

```
function inserisci(nome, valore) {
    var Vini = document.cantina.vini;
   var numVini = Vini.options.length;
   var NuovoVino = new Option(nome, valore, false, true);
   Vini.options[numVini] = NuovoVino;
}
```

```
function elimina() {
      document.cantina.vini.options.length =0;
}
```

Cancella le opzioni nella lista di selezione

Metodi di Select

- remove(pos)
  - Rimuove l'opzione in posizione pos
    - Si parte da zero
- add(option, pos)
  - Inserisce l'opzione option in posizione pos
    - Si parte da zero
- add(option)
  - Aggiunge l'opzione option in ultima posizione

Oggetto **Checkbox**

- Possiamo far riferimento ad una casella di spunta nei seguenti due modi

  form.NomeCheckbox

  form.elements[i]

Dove form è un oggetto di tipo **Form**

- Se ci sono più caselle di spunta con lo stesso nome, possiamo far riferimento ad esse tramite

  form.NomeCheckbox[i]

- Per sapere quante caselle di spunta con lo stesso nome ci sono, leggiamo la proprietà

  form.NomeCheckbox.length

Tutto questo vale per qualsiasi controllo

Esempio – HTML

```
<FORM ACTION="#" NAME="elenco" >
<input type="checkbox" name="provincia"
        value="Avellino" onclick="mostra()"> Avellino<br>
<input type="checkbox" name="provincia"
        value="Benevento" onclick="mostra()"> Benevento
   <br>
<input type="checkbox" name="provincia"
        value="Caserta" onclick="mostra()"> Caserta <br>
<input type="checkbox" name="provincia" value="Napoli"
        onclick="mostra()"> Napoli<br>
<input type="checkbox" name="provincia"
        value="Salerno" onclick="mostra()"> Salerno<br>
</FORM>
```

Esempio – JavaScript

```
function mostra() {
var msg="";
var prov = document.elenco.provincia;
var numProv = prov.length;
for (var i=0; i < numProv; i++ )
   if(prov[i].checked)
         msg = msg + " " + prov[i].value;
if(msg != "") alert("Hai selezionato:\n" + msg);
else alert("Non ci sono città selezionate "); }
```

Ricalcoliamo tutto !!

Caso particolare

- Presumibilmente, se il modulo invoca uno script PHP, useremo come nome delle checkbox provincia[]
  - In tal caso, la funzione mostra() non funziona

- Dobbiamo cambiare la funzione mostra()
- Se sappiamo in che posizione i controlli relativi alle checkbox compaiono nel modulo (in che posizione compaiono in elements), allora una possibile soluzione è nella prossima slide
  - La soluzione proposta non è generale, dobbiamo adattarla ogni volta che modifichiamo la "posizione" dei controlli nella form

```
function mostra() {
var msg="";
var numProv = 5;      // consideriamo 5 province
var posCheck = 12  // la prima checkBox è il
                          // tredicesimo controllo
for (var i=0; i < numProv; i++ ) {
    var prov = document.elenco.elements[posCheck+i];
    if(prov.checked)
          msg = msg + " " + prov.value;
  }

if(msg != "")
    alert("Hai selezionato:\n" + msg);
else
    alert("Non ci sono città selezionate "); }
```

Esiste un'altra soluzione assegnando un id *opportunamente formattato* ad ogni checkbox ed utilizzando getElementById

Altra soluzione possibile

```
function mostra() {
var msg="";
var numElementi = document.forms[0].elements.length;
for(i=0; i< numElementi; i++) {
    controllo = document.forms[0].elements[i];
    if(controllo.name == "provincia[]" && controllo.checked )
            msg = msg + " " + controllo.value;
}

if(msg != "")    alert("Hai selezionato:\n" + msg);
else     alert("Non ci sono città selezionate ");
}
```

L'oggetto document supporta anche il metodo getElementsByTagName…