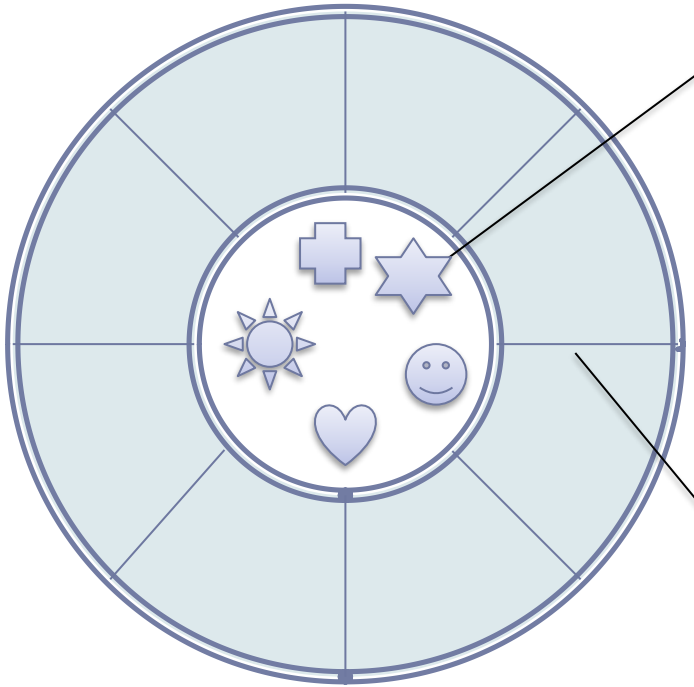
LEZIONE 6

OGGETTI

- l'oggetto è un esemplare (in inglese: ***instance***, comunemente anche se impropriamente tradotto con istanza) di una **classe**.
- Ogni istanza è separata dalle altre, ma condivide le sue caratteristiche generali con gli altri oggetti della stessa classe.

OGGETTO

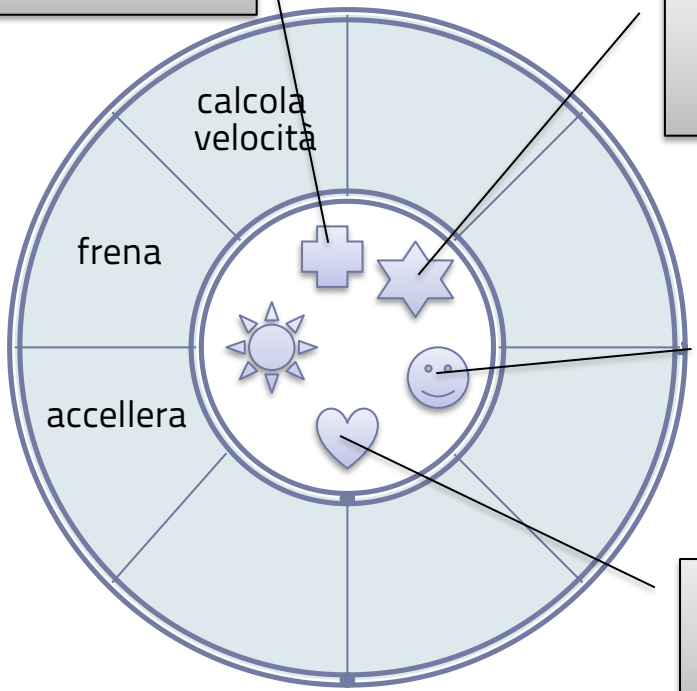


stati
(variabili)

comportamenti o
metodi
(funzioni)

BICICLETTA

diametro ruota



colore

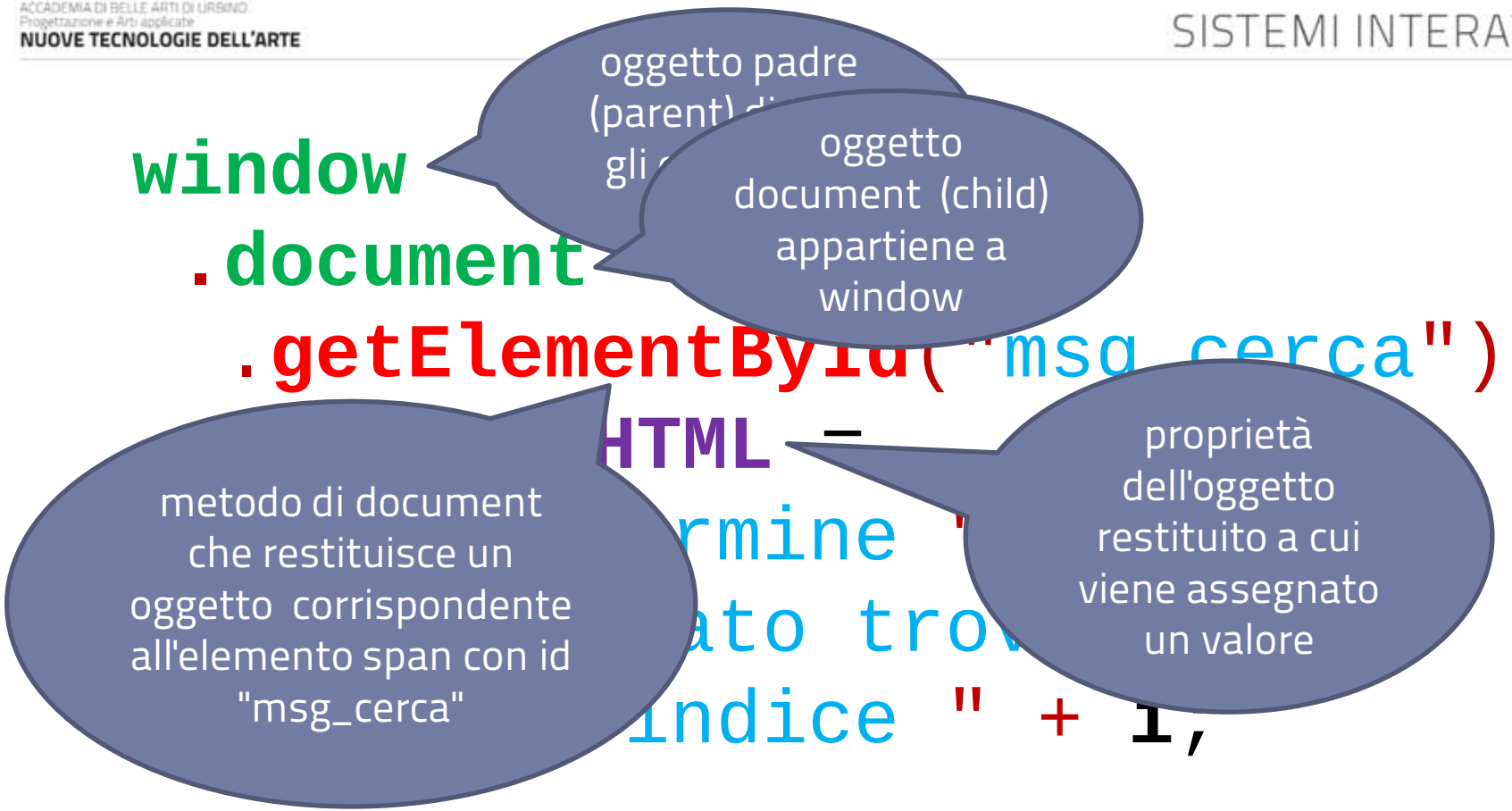
pedalate / minuto

rapporto

Modello mountain-bike

- Telaio alluminio
- Moltiplica anteriore a 3 rapporti
- Cambio a 8 rapporti
- Freni a disco
- ecc...

- Alle proprietà e ai metodi di un oggetto si accede tramite l'operatore punto.
- Gli operatori punto possono essere usati in catena.

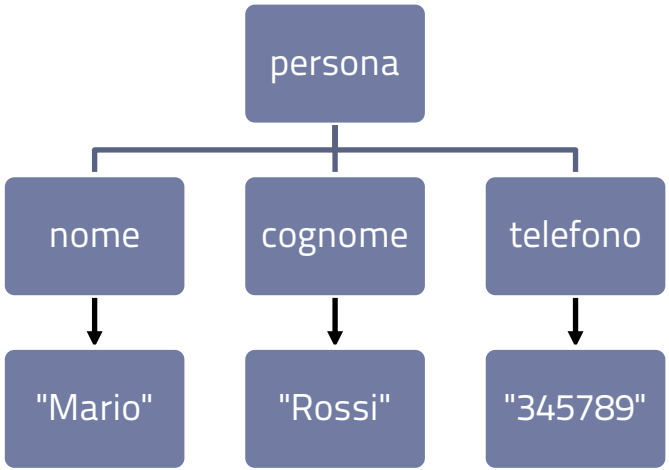

window
.document
.getElementById("msg_cerca")
HTML =
rmine "
ato trov
ndice " + i;
oggetto padre
(parent) di
gli
oggetto
document (child)
appartiene a
window
metodo di document
che restituisce un
oggetto corrispondente
all'elemento span con id
"msg_cerca"
proprietà
dell'oggetto
restituito a cui
viene assegnato
un valore

AGENDA TELEFONICA

```
function agendaTelefonica (nom, co, tel)
{
  this.nome = nom;
  this.cognome = co;
  this.telefono = tel;
}
var persona = new agendaTelefonica(...)
```

AGENDA TELEFONICA

```
var persona = new
  agendaTelefonica("Mario", "Rossi",
  "345789");
```



AGENDA TELEFONICA

```
var persona = new
  agendaTelefonica("Mario", "Rossi",
  "345789");
var nomePersona;
var prop = "nome";
nomePersona = persona.nome;
nomePersona = persona["nome"];
nomePersona = persona[prop];
```

PROPRIETÀ DI UN OGGETTO

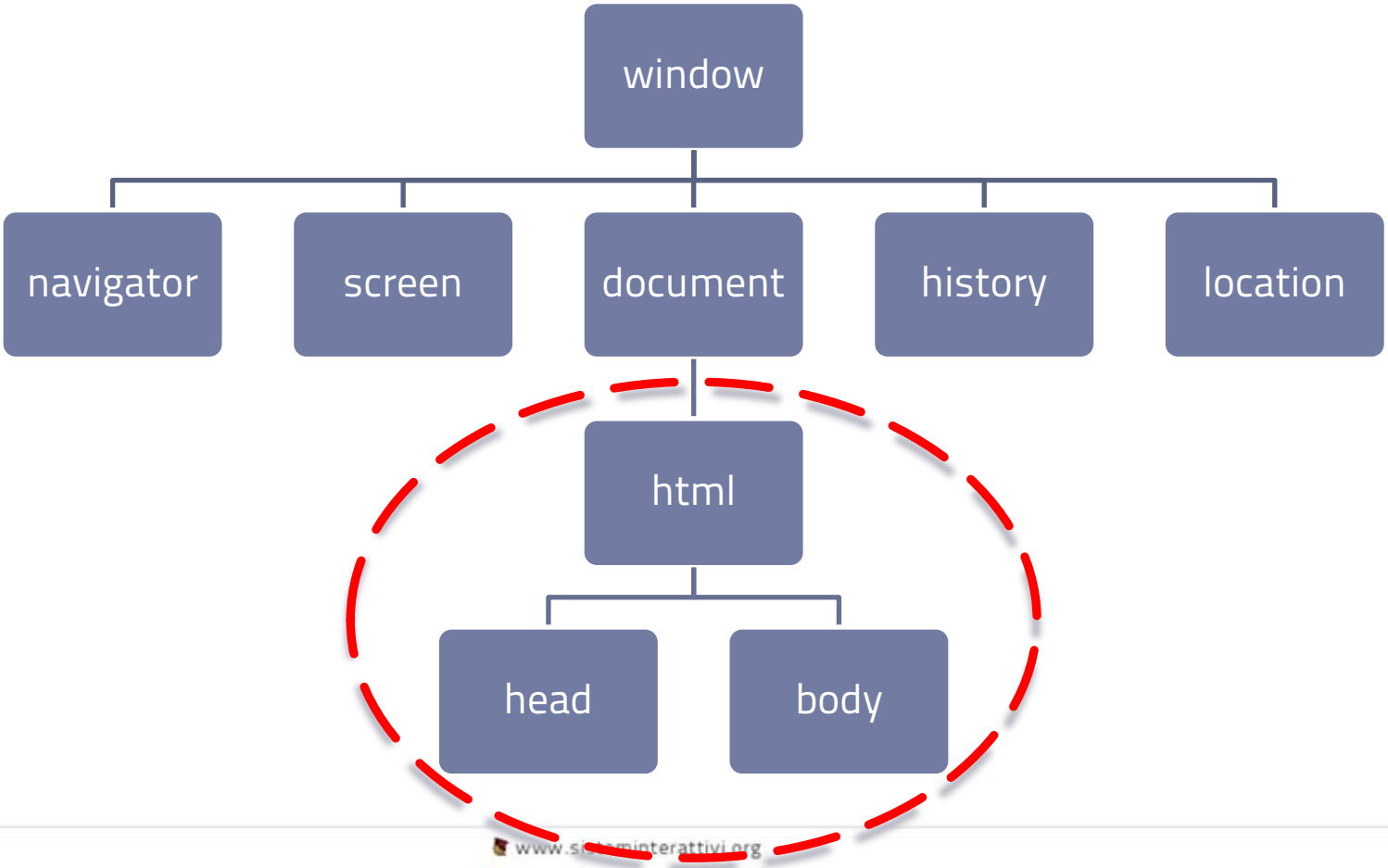
- Ho due modi per recuperare il valore della proprietà di un oggetto:
  - Operatore punto.
  - Sintassi Array Associativo: la stessa sintassi dell' Array numerico con al posto dell'indice numerico il nome della proprietà in formato stringa di caratteri.

IL COSTRUTTO FOR IN

```
for (var prop in persona)
 {
   str = str + prop + " = " +
         persona[prop] + ", ";
 }
```

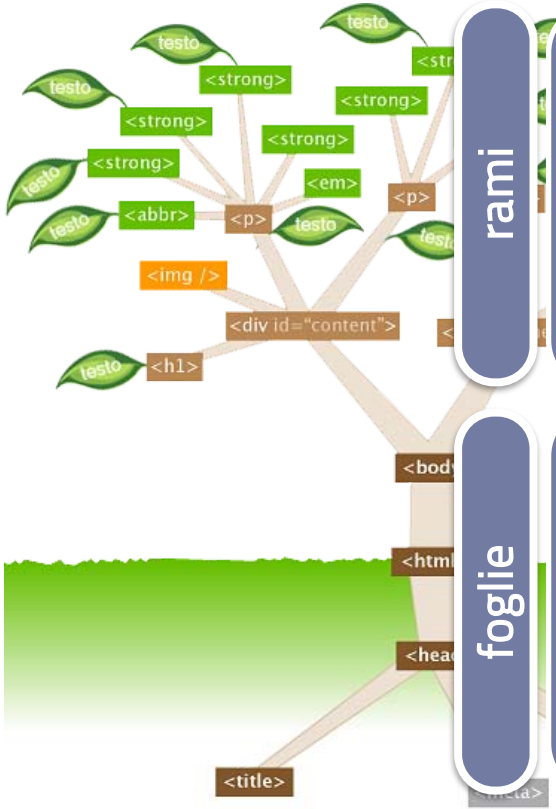
IL DOM

- Il DOM è un'API (Application Programming Interface - **interfaccia per la programmazione di applicazioni):** un insieme di funzioni, metodi e proprietà, che consento di interagire col sistema costituito da una pagina WEB.
- Ogni elemento HTML è interpretato dall'interfaccia come un oggetto.
- Il DOM è un modello che descrive come i diversi oggetti di una pagina sono collegati tra loro.


window
navigator
screen
document
history
location
html
head
body

L'OGGETTO DOCUMENT

LA METAFORA DELL'ALBERO



element
sono i nodi che corrispondono ai tag HTML

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

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 los tesso 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;
```

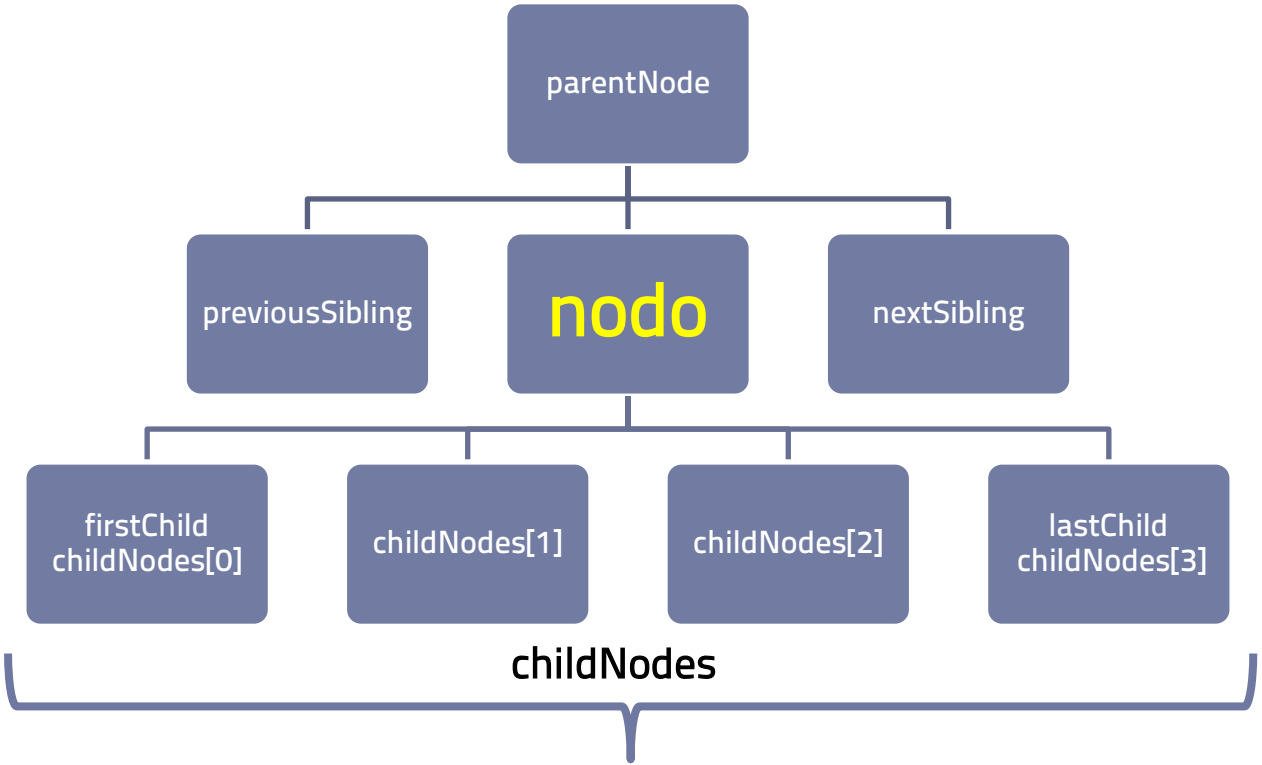
ATTRIBUTI

- **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

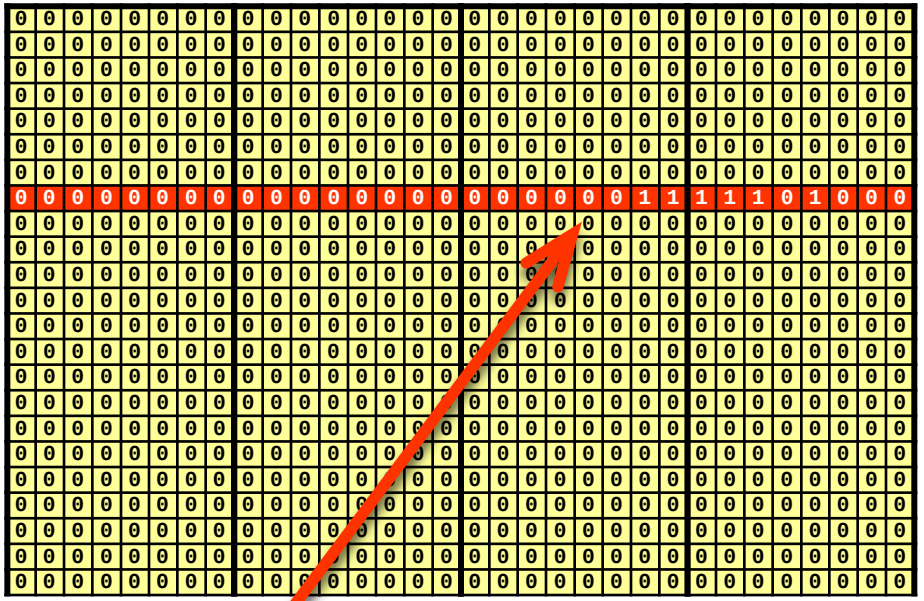
- Quando assegno un valore a una variabile l'interprete javascript riserva uno spazio di memoria per quella variabile.
- Possiamo dire che ad ogni variabile corrisponde una cella della memoria fisica del computer.
- Ognuna di queste celle è raggiungibile per l'elaborazione attraverso un riferimento anch'esso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

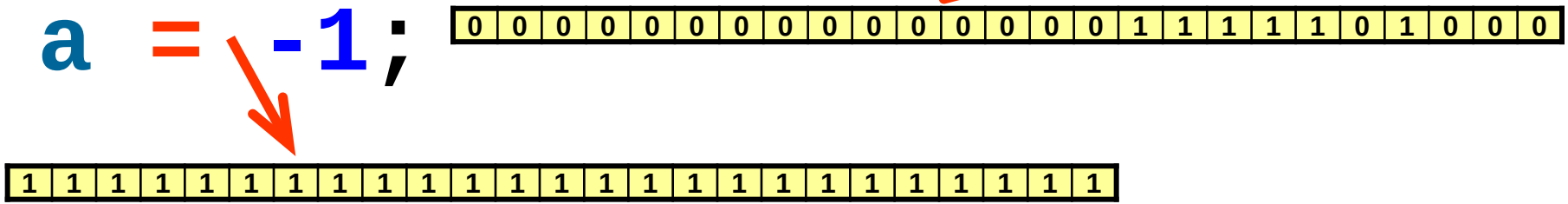


```
var a = 1000;
```

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");
```

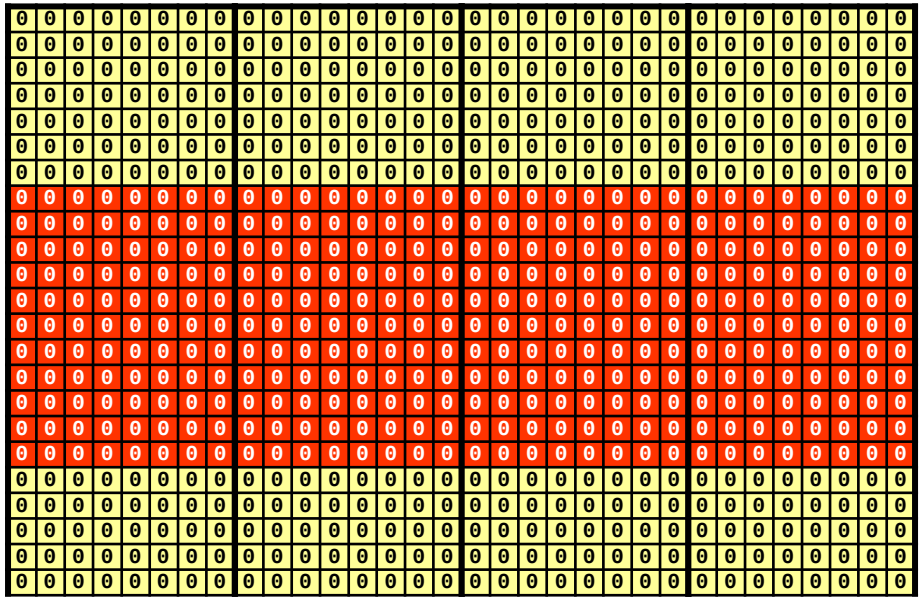
La cella di memoria di 32 bit rappresentata da elemento non conterrà l'elemento html creato ma l'indirizzo fisico di memoria in cui è memorizzato.

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.

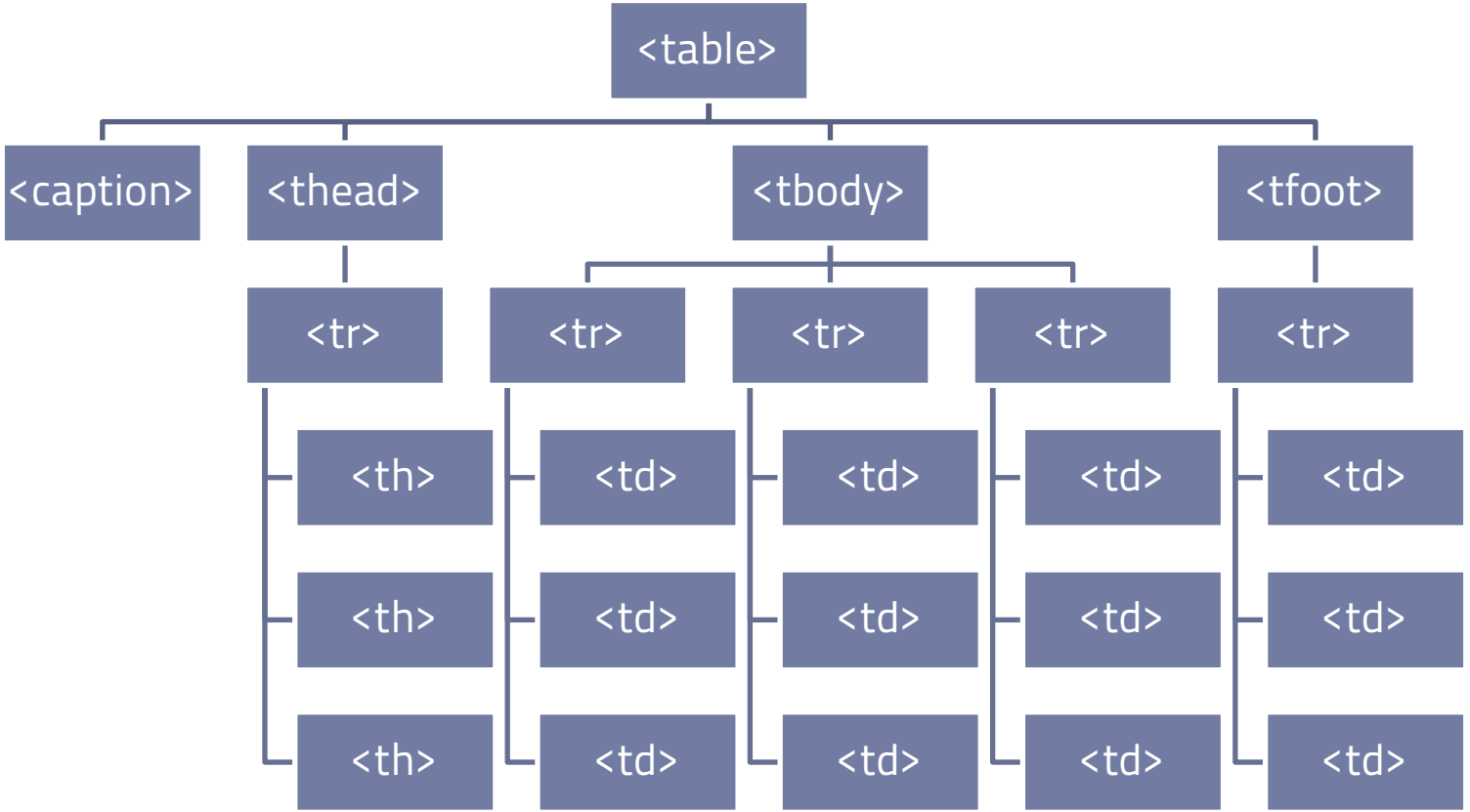
TABLE

TABELLA


<th>
<caption>
<td>
Agenda telefonica
nome
cognome
telefono
Mario
Rossi
052232323
Giuseppe
Bianchi
0541960505
Pietro
Verdi
3334567890
Marco
Viola
3331234567
nome
cognome
telefono
<tr>
<thead>
<tbody>
<tfoot>

```
<table id="tab_rubrica">
        <caption>Agenda telefonica</caption>
        <thead>
                <tr>
                        <td>telefono</td>
                        <td>cognome</td>
                        <td>nome</td>
                </tr>
        </thead>
        <tbody>
                <tr>
                        <td>Mario</td>
                        <td>Rossi</td>
                        <td>052232323 </td>
                </tr>
                ...
        </tbody>
</table>
```

TABLE NEL DOM



CREARE UNA TABELLA

- **document.createElement("table")**
  per creare un'oggetto table bisogna usare il metodo di document **createElement**.

```
var tabella = document.createElement("table");
var myCaption = tabella.createCaption();
...
```

METODI SPECIFICI DI TABLE

- **createCaption(), createTFoot() e createTHead()**
  Questi tre metodi se applicati a una tabella creano rispettivamente elementi **caption**, **thead** e **tfoot** vuoti e li inseriscono nell'oggetto **table** a cui sono applicati.
- Sono una scorciatoia all'uso di createElement. Scrivere:

```
tabella.createTHead() ;
```

- corrisponde a:

```
var myHead = document.createElement("thead");
tabella.appendChild(myHead);
```

AGGIUNGERE UNA RIGA

- **insertRow(indice)**
  Inserisce una riga vuota nella tabella alla posizione specificata da **indice**. Indice è obbligatorio. Se vale -1 la riga viene posizionata in coda a quelle esistenti.
- Può essere applicato all'oggetto **table** agli oggetti **thead**, **tbody** e **tfoot**.

```
tabella.insertRow(-1) ;
```

- corrisponde a:

```
var myRow = document.createElement("tr");
tabella.appendChild(myRow);
```

AGGIUNGERE UNA CELLA

- insertCell(indice)
  Inserisce una cella vuota (elemento <td>) in una riga alla posizione specificata da **indice**. Indice è obbligatorio. Se vale -1 la cella viene aggiunta in coda a quelle esistenti.
- Può essere applicato all'elemento **tr**.

```
var myRow = tabella.insertRow(-1) ;
myRow.insertRow(-1) ;
```

- corrisponde a:

```
var myRow = document.createElement("tr");
var myCell = document.createElement("td");
myRow.appendChild(myCell);
tabella.appendChild(myRow);
```

window.localStorage

- Supportato da
  - Internet Explorer 8+, Firefox 3.5+, Safari 4.0+, Chrome 4.0+, Opera 10.5+, iPhone 2.0+, Android 2.0+
- setItem(key, value)
  memorizza il valore value associandolo al nome key
- getItem(key)
  recupera il valore associato a key.
- Il valore memeorizzato è sempre una stringa

window.JSON

- Supportato da
  - Internet Explorer 8+, Firefox 3.5+, Safari 4.0+, Chrome 4.0+, Opera 10.5+, (iOS 5.0+), Android 2.0+
- **stringify**(object)
  converte un oggetto in stringa
- **parse**(jsonString)
  converte una stringa ottenuta con stringify in oggetto.