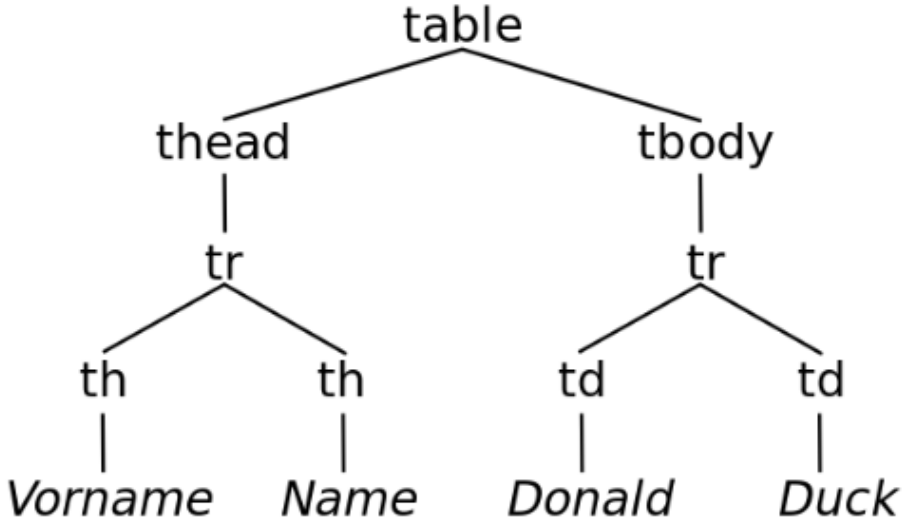
JavaScript im DOM

Was ist das DOM?

- Das Document Object Model (DOM) ist eine W3C-Spezifikation einer Programmierschnittstelle, welche HTML- oder XML-Dokumente als eine Baumstruktur darstellt.
- In der Baumstruktur ist jeder Knoten ein Objekt, welches einen Teil des Dokumentes repräsentiert, wie
  - ein Absatz,
  - eine Überschrift,
  - ein Video oder
  - eine Tabellenzelle.
- Die Schnittstelle ist plattform- und programmiersprachenunabhängig und erlaubt damit standardisiert, die Struktur und das Layout eines Dokumentes zu verändern.
- In einem Internet-Browser bildet dies einen wichtigen Baustein für client-seitige dynamische Webseiten.

Der DOM-Baum im HTML

```
<table>
    <thead>
        <tr>
            <th>Vorname</th>
            <th>Name</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Donald</td>
            <td>Duck</td>
        </tr>
    </tbody>
</table>
```



DOM Level 0

- Der Browser stellt bei DOM Level 0 ein Objekt-Modell zur Verfügung.
- Das Objekt-Modell enthält Collections von DOM-Knoten, auf die zugegriffen werden kann.

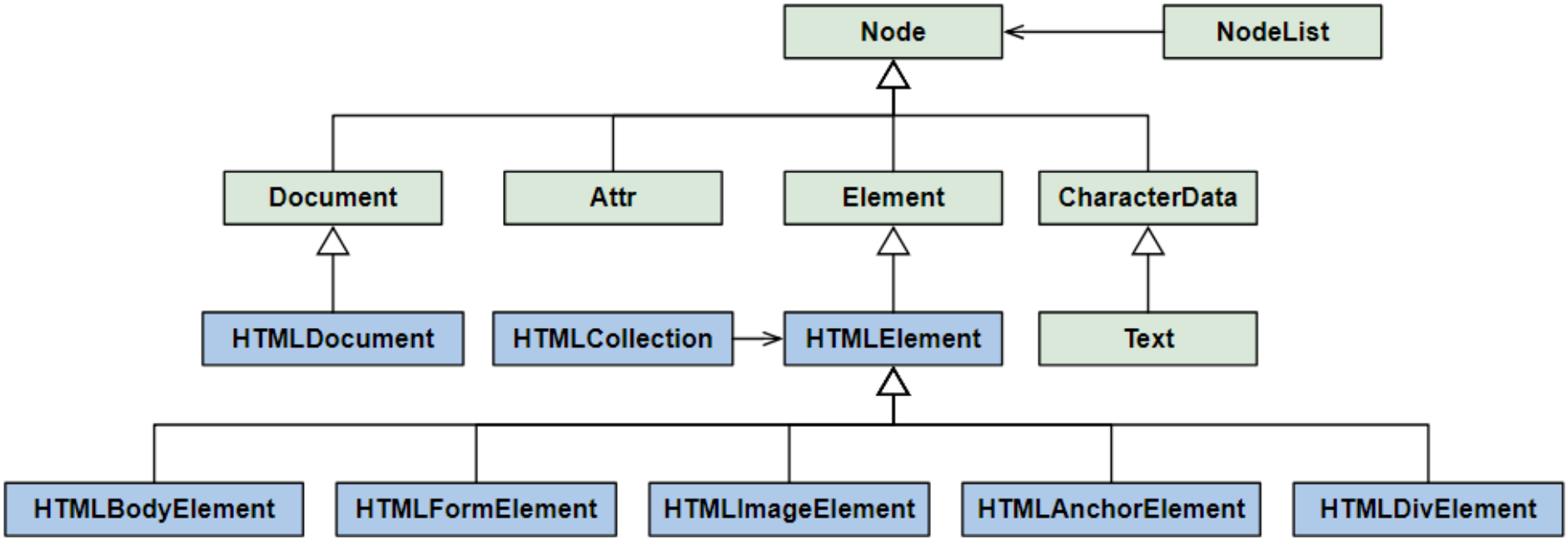
- DOM Level 0 ist **keine** offizielle DOM-Version.
  - Sie wurde eingeführt von einigen Browser-Herstellern.
  - Sie wurde nie formal spezifiziert.
  - Sie ist inzwischen veraltet und sollte nicht mehr verwendet werden.

DOM Level 2

- DOM Level 2 ist die Schnittstelle zum Zugriff auf HTML-Dokumente.
  - Sie ist formal spezifiziert und standardisiert durch W3C.
  - Sie wird von allen gängigen Browsern unterstützt.
  - Sie kann auch von anderen Programmiersprachen als JavaScript verwendet werden.

- Erweiterungen gegenüber DOM Level 0:
  - Alle Elemente und alle Attribute einer HTML-Seite sind voll ansprechbar.
  - Ein Zugriff auf XHTML-Dokumente ist möglich.
  - Die nachträgliche Erzeugung von Elementen und deren Einhängen in den Dokumenten-Baum wird möglich.

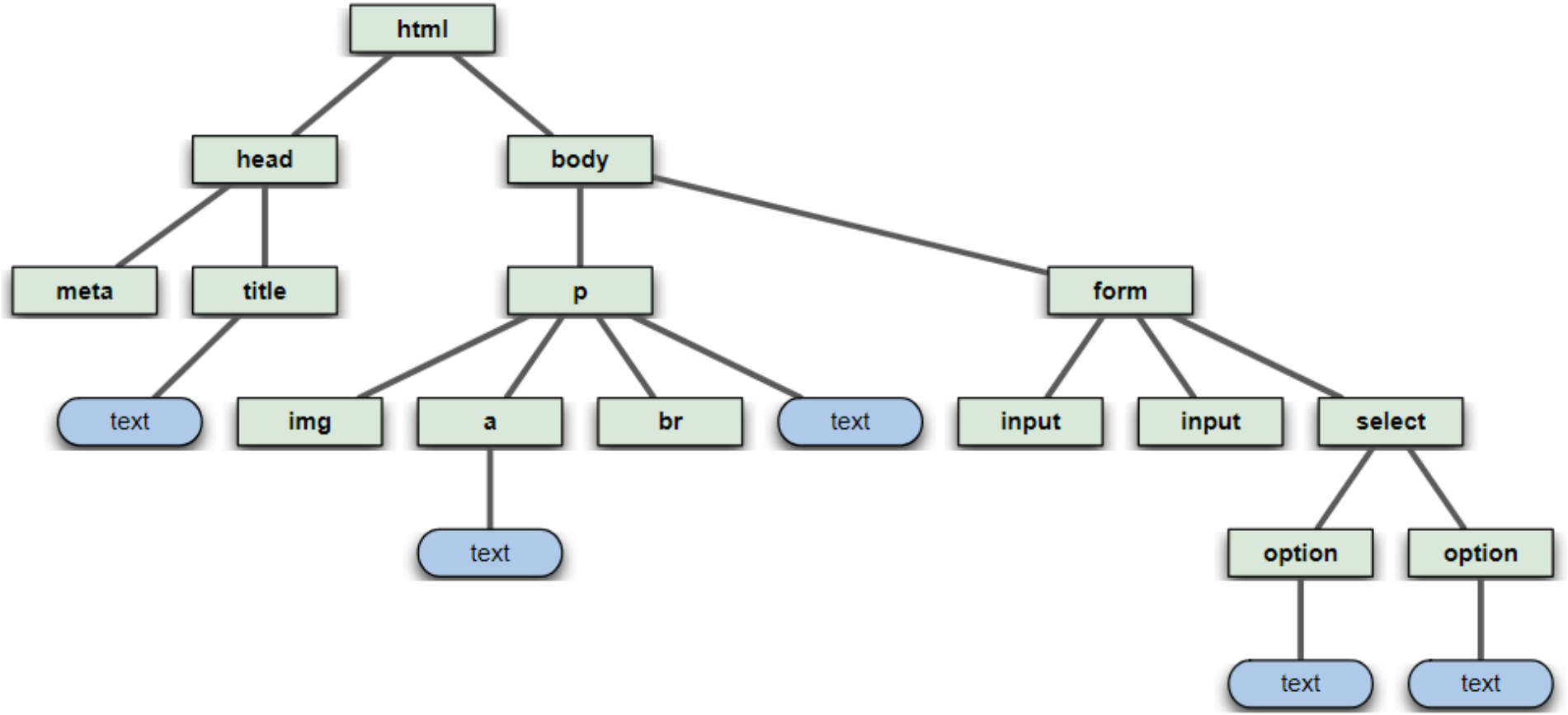
Die DOM Level 2 Struktur



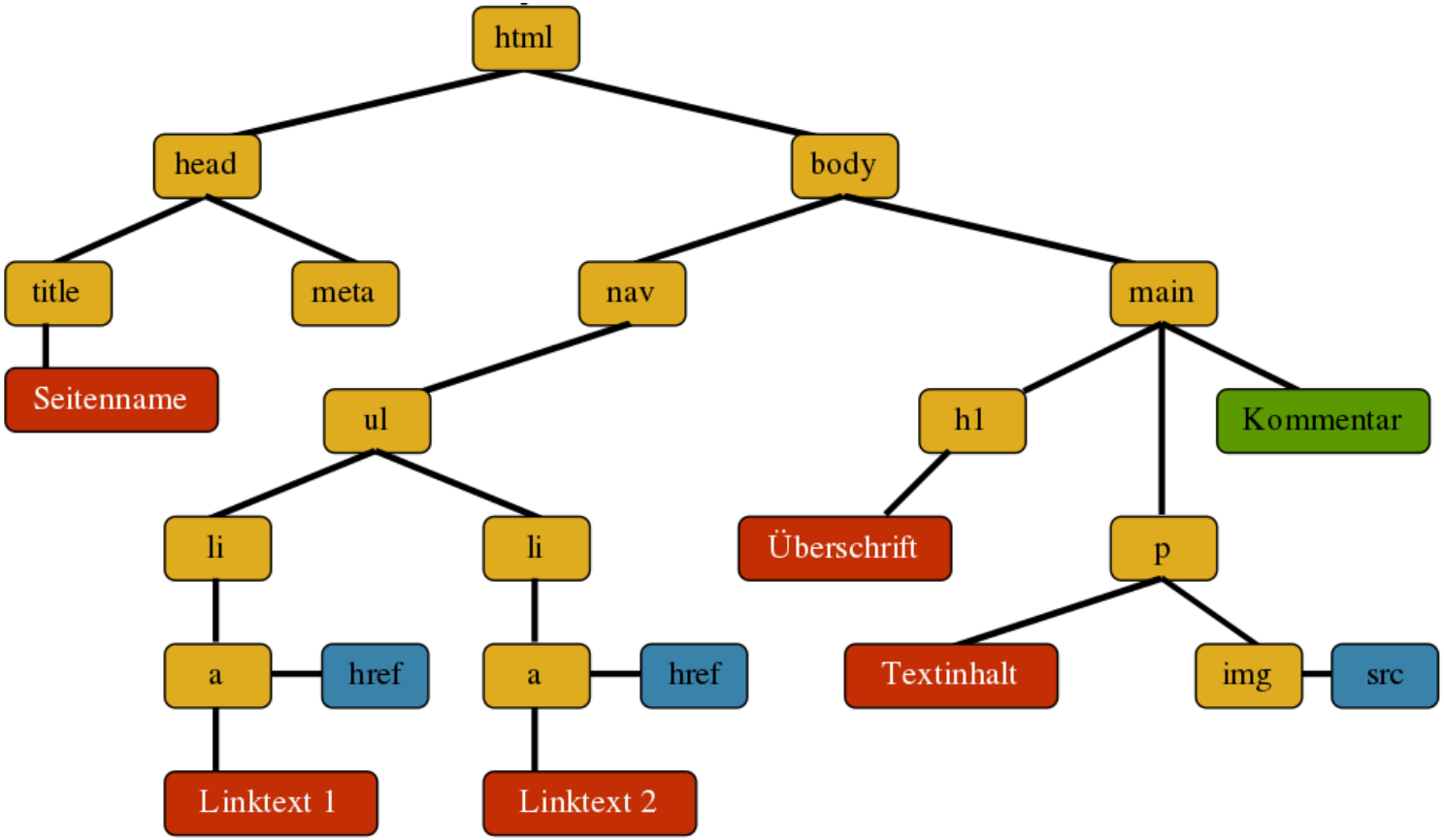
Ein HTML-Dokument als Beispiel

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>DOM-Tree</title>
</head>
<body>
  <p>
    Ein erster Absatz <img src="logo.gif"/> <br/>
    <a href="http://www.hs-mannheim.de">Home</a>
  </p>
  <form action="#" method="GET">
    <input type="text" name="nachname"/>
    <input type="text" name="vorname"/>
    <select name="geschlecht">
      <option value="m">männlich</option>
      <option value="f">weiblich</option>
    </select>
  </form>
</body>
</html>
```

Ein HTML-Dokument als Beispiel: Der dazu passende DOM Level 2 Baum



Noch ein DOM Level 2 Baum...



- Elementknoten
- Attributknoten
- Textknoten

https://wiki.selfhtml.org/wiki/JavaScript/DOM

Erweiterungen durch DOM Level 2

- DOM Level 2 erweitert das document-Objekt um:
  - Methoden zum Erstellen von Knoten.
  - Methoden zum Auffinden von Knoten über ID, über das Name-Attribut und über den Elementnamen.

Methode	**Bedeutung**
createElement(tagName)	Erstellt Element vom Typ tagName
createAttribute(attrName)	Erstellt Attribut mit dem Namen attrName
createTextNode(initText)	Erstellt und initialisiert Textknoten
getElementById(id)	Liefert Element mit id-Attributwert id
getElementsByName(name)	Liefert Elemente mit dem Namen name
getElementsByTagName(tag)	Liefert Elemente des Tag-Typs tag

HTML5-Erweiterungen

Methode	Bedeutung
querySelector(selector)	Liefert das erste Element, auf das der Selektor zutrifft.
querySelectorAll(selector)	Liefert ein Array aller Elemente auf die der Selektor zutrifft.

Selector	Example
.class	.intro
.class1.class2	...
.class1 .class2	 ...
#id	#firstname
*	*
element	p

element.class	p.intro
element,element	div, p
element element	div p
element>element	div > p
element+element	div + p
element1~element2	p ~ ul
[*attribute*]	[target]
[*attribute=value*]	[target=_blank]
[*attribute~=value*]	[title~=flower]
[*attribute|=value*]	[lang|=en]

[*attribute^=value*]	a[href^="https"]
[*attribute$=value*]	a[href$=".pdf"]
[*attribute*=value*]	a[href*="w3schools"]
:active	a:active
::after	p::after
::before	p::before
:checked	input:checked
:default	input:default
:disabled	input:disabled

https://www.w3schools.com/cssref/css_selectors.asp

Baum-Operationen

- **`node`**, **`document`** und **`element`** haben Methoden zum Durchwandern und Manipulieren des Baums, insbesondere:
  - zur Erzeugung neuer Knoten mit Hilfe des **`document`**-Objektes.
  - zum Durchwandern des Baums mit Hilfe der **`node`**-Objekte.
  - Methoden zur Strukturänderung der **`node`**-Objekte.

Zugriff auf DOM-Knoten

- Direkter Zugriff auf Element per eindeutiger ID:
  - Jedes benötigte HTML-Element muss id-Attribut haben.
  - **`knoten = document.getElementById("ID")`**

- Zugriff über Element-Typ:
  - **`knoten = document.getElementsByTagName("h3")[2]`**

- Zugriff auf Elemente über deren name-Attribut:
  - Nicht jeder Tag darf ein name-Attribut haben, evtl. mehrere Elemente mit demselben Namen, beispielsweise Radiobuttons.
  - **`knoten = document.getElementsByName("abc")[0]`**

- Verwendung von DOM Level 0-Pfaden.

Weitere Möglichkeiten zum Zugriff

- Speziell beim Aufruf von Handlerfunktionen:
  - **`this`** verweist auf das Element, in dem der Code steht.
  - **`<div onclick="verbergen(this);"> ... </div>`**
    - Nur gültig innerhalb des HTML-Tags.

- Eine veraltete Methode für manche Objekte ist der Zugriff ohne Nennung der Collection über das **`name`**-Attribut:
  - **`document.MeinFormular.Eingabe.value = "alt";`**
  - **`document.MeinBild.src = "bild.gif";`**
  - Dieser o.g. Zugriff sollte vermieden werden!
    - Verwenden Sie stattdessen besser:
      `document.getElementsByName("...")`

name oder id für die Adressierung?

- Für die Adressierung ist **`id`** zu bevorzugen!
  - **`name`** ist nur bei manchen Tags zulässig.
  - **`name`** ist nicht eindeutig.
    - Bei mehreren gleichartigen Elementen ggf. **`class`** verwenden.
  - **`name`** hat unterschiedliche Bedeutungen:
    - **`<a>`** Sprungmarke
    - **`<input>`** Parametername
  - **`name`** ist nur noch aus Kompatibilitätsgründen zulässig.
  - **`name`** ist im DOM 2 Core nicht enthalten.

  - Die **`id`** ist hingegen bei allen Tags zulässig!

Das node-Objekt

- Knoten des DOM-Baumes werden durch node-Objekte repräsentiert.
- Mögliche Arten von Knoten sind:
  - Elementknoten
  - Textknoten
  - Attributknoten

Das node-Objekt

Attribut	Bedeutung
nodeName	Elementname
nodeType	Knotentyp (1=Element, 2=Attribut, 3=Text)
nodeValue	Wert des Knotens
data	Rumpftext (nur bei Textknoten)
attributes	Array aller Attribute
parentNode	Vaterknoten
childNodes	Array aller Kinderknoten
firstChild	erster Kindknoten
lastChild	letzter Kindknoten
previousSibling	linker Nachbarknoten
nextSibling	rechter Nachbarknoten

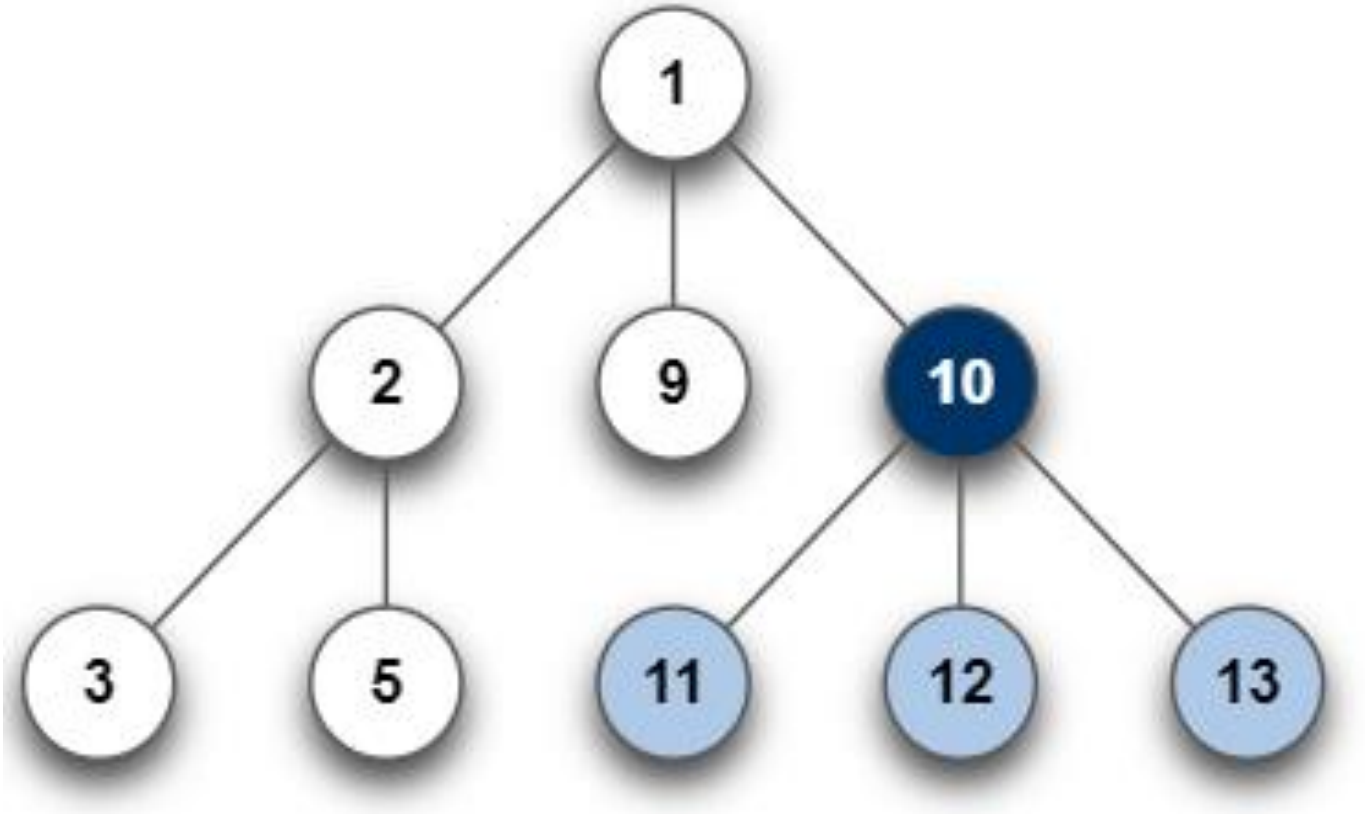
Das node-Objekt

Methode	Bedeutung
cloneNode(recursiveFlag)	Liefert Knotenkopie (rekursiv falls recursiveFlag=true)
hasChildNodes()	Liefert true, wenn Kindknoten vorhanden sind, sonst false
appendChild(new)	Knoten new als letztes Kind anfügen
insertBefore(new, node)	Knoten new links von bestehendem Knoten node einfügen
removeChild(node)	Kind node entfernen
replaceChild(new, old)	Ersetzt Kind old durch new
getAttribute(name)	Liefert Wert des Attributes name
setAttribute(name, value)	Setzt Wert des Attributes name auf value
removeAttribute(name)	Attribut mit Namen name entfernen
getAttributeNode(name)	Liefert Knoten des Attributs name
setAttributeNode(node)	Attributknoten node hinzufügen
removeAttributeNode(node)	Attributknoten node entfernen

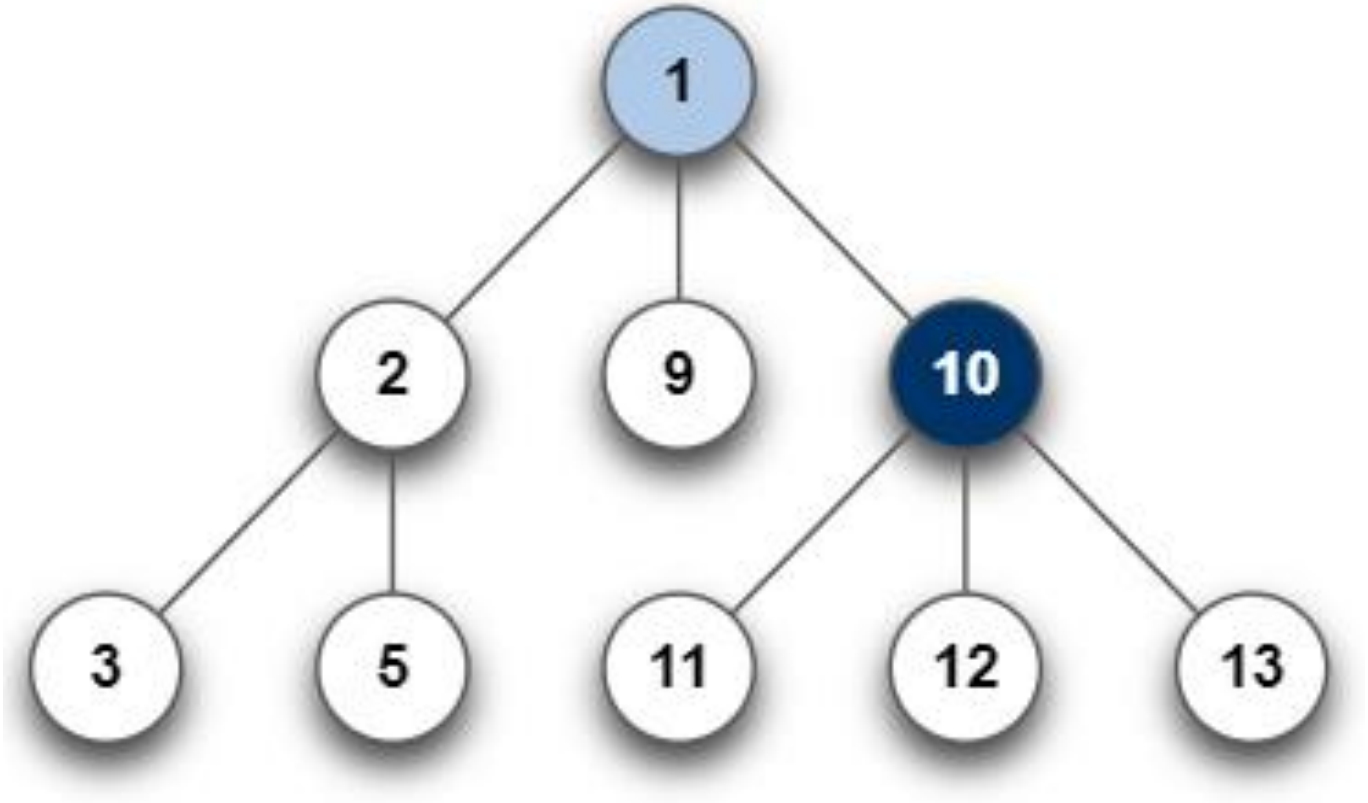
Das node-Objekt

Methode	Bedeutung
appendData(text)	Hängt text an
insertData(start, text)	Fügt text ab der Position start ein
replaceData(start, len, text)	Ersetzt len Zeichen des Textes ab start mit text
deleteData(start, text)	Entfernt len Zeichen des Rumpftextes ab start

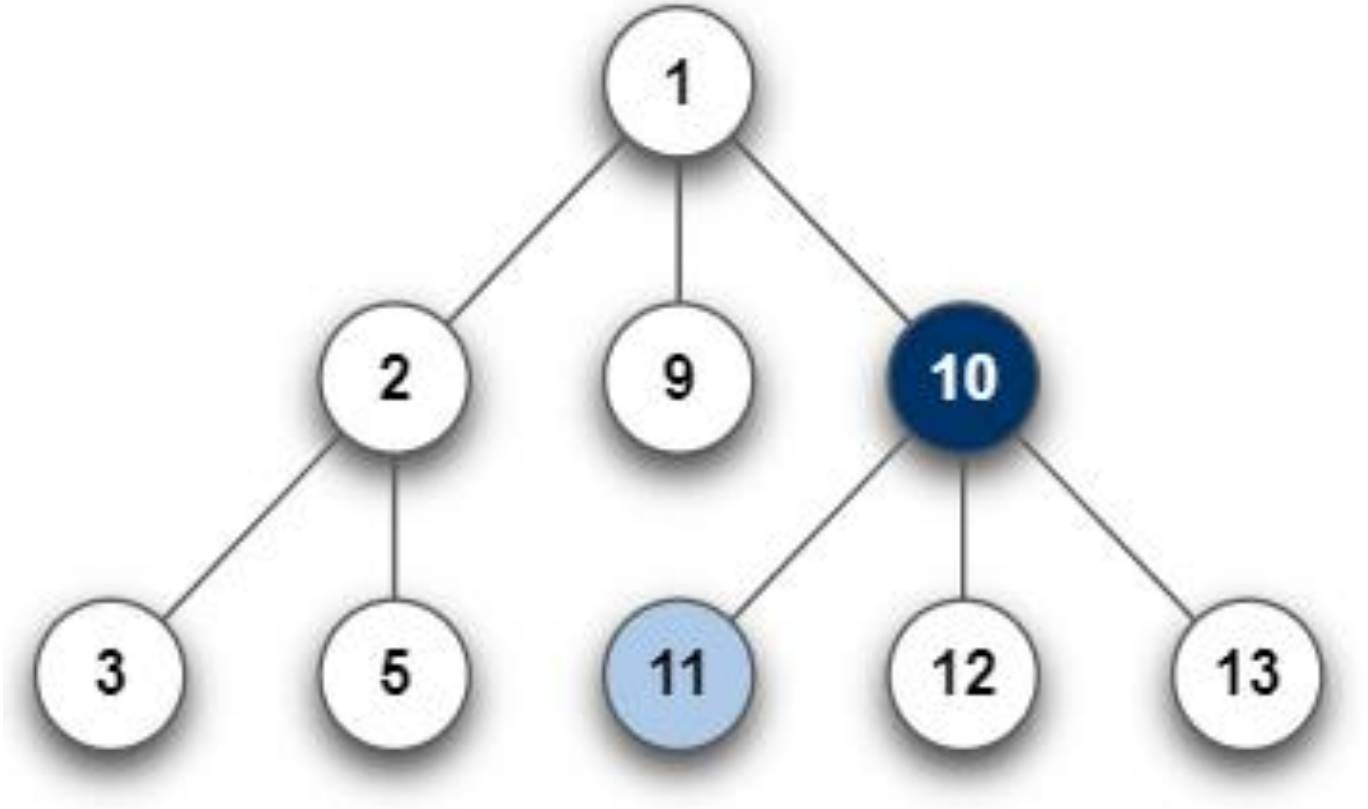
Knoten-Beziehungen: childNodes



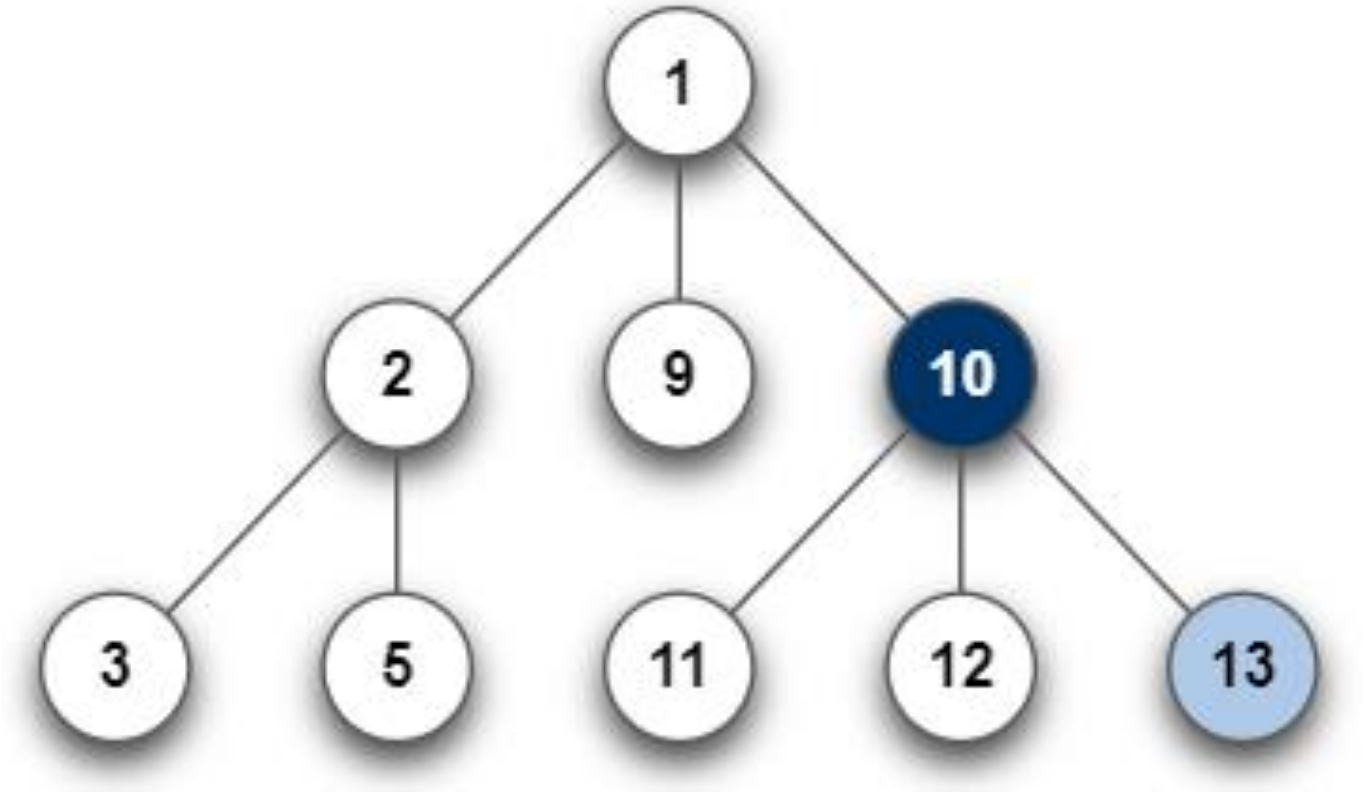
Knoten-Beziehungen: parent



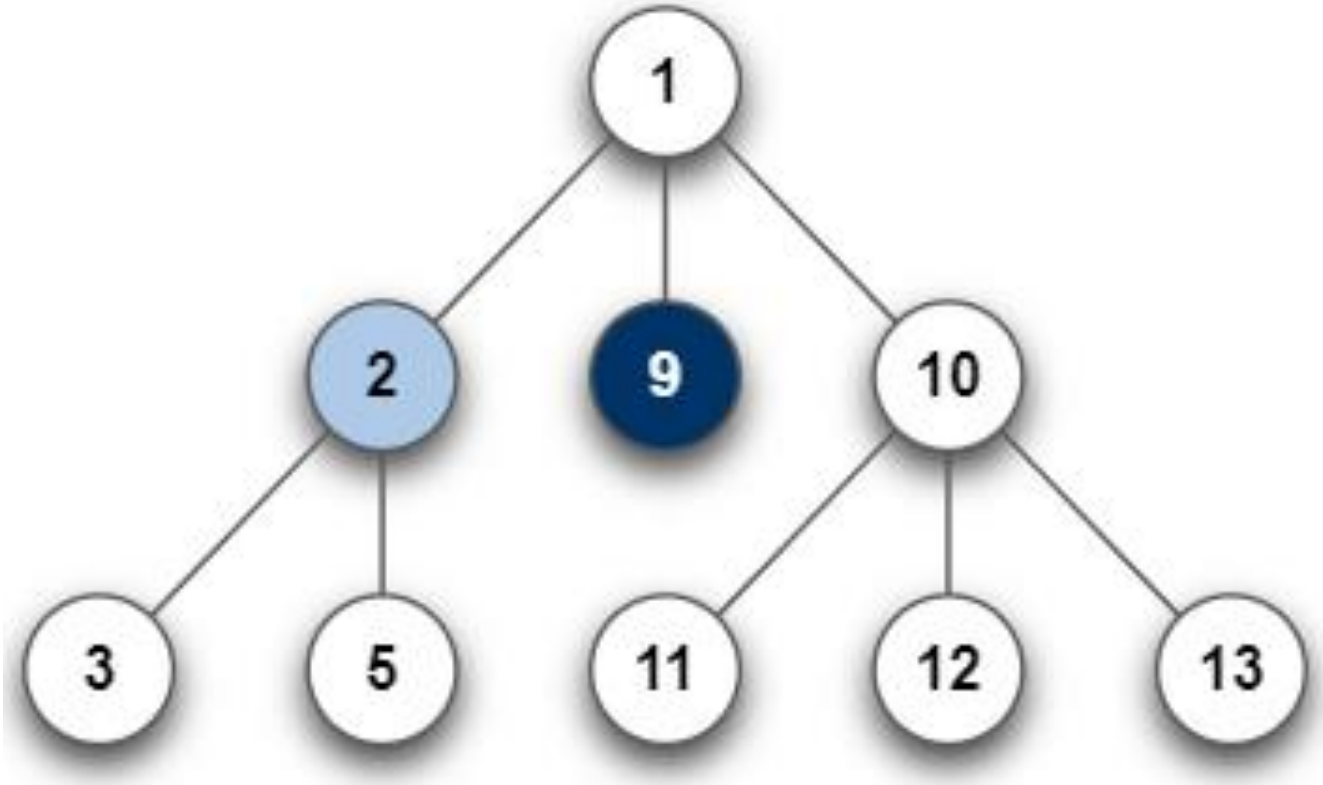
Knoten-Beziehungen: firstChild



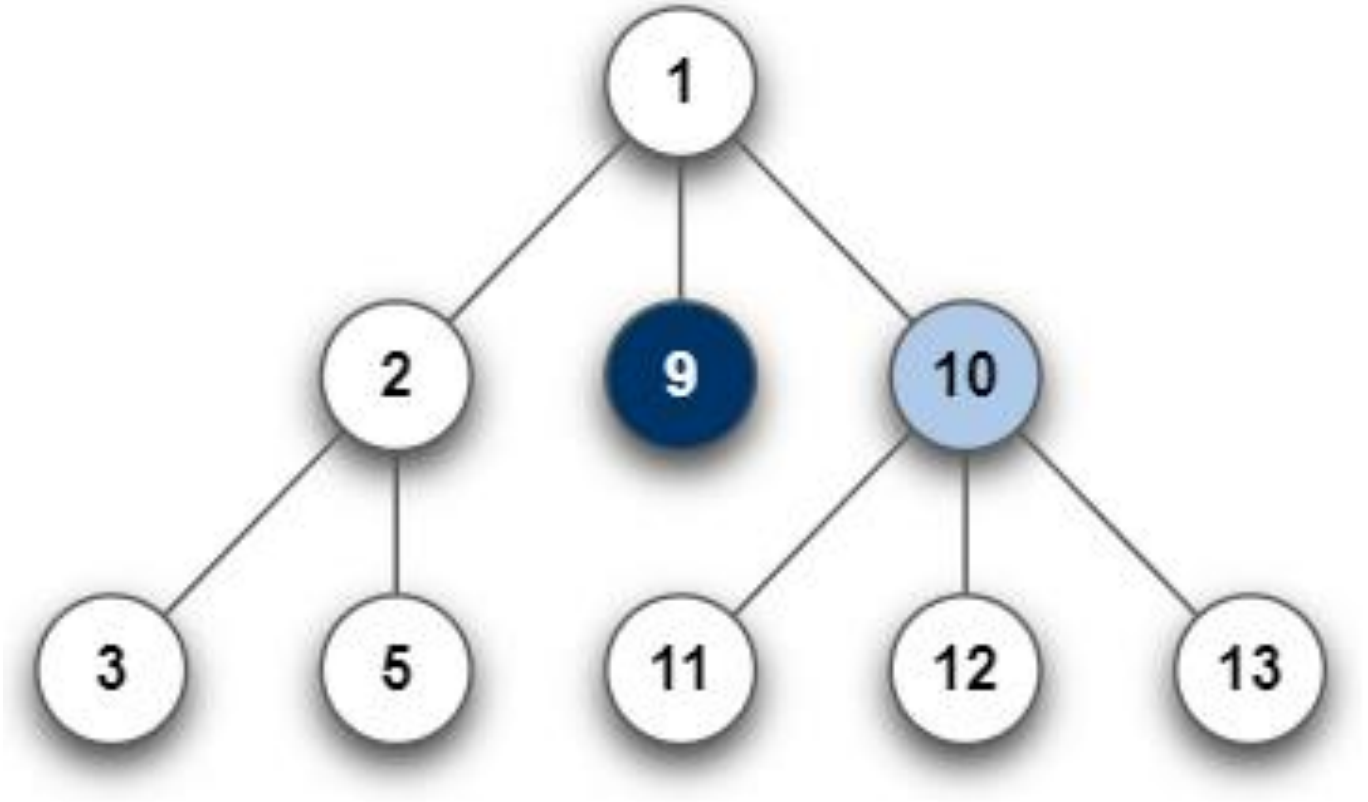
Knoten-Beziehungen: lastChild



Knoten-Beziehungen: previousSibling



Knoten-Beziehungen: followingSibling



Zugriff auf Attribute

- Zugriff über die Kernobjekte von DOM 2:
  - Die Attribute sind in Unterknoten gespeichert.
  - Der Zugriff erfolgt über **`element.getAttribute()`** bzw. **`element.setAttribute()`**
  - z.B. **`img.setAttribute("src", "img/portrait.jpg");`**

- Zugriff über die HTML-Erweiterungen von DOM 2:
  - Die Objekte haben Attribute, die direkt angesprochen werden können.
  - Die Namen entsprechen HTML-Namen.
  - Ausnahme: **`class`** wird zu **`className`**
  - z.B. **`img.src = "img/portrait.jpg";`**

Erzeugung von Knoten: DOM-Stil vs. HTML-Stil

```
var div = document.getElementById("leer_dom");
var img = document.createElement("img");
img.setAttribute("src", "img/html-portrait.jpg");
img.setAttribute("alt", "Sir Karl Popper");
img.setAttribute("width", "70px");
img.setAttribute("style", "border: 1px solid; border-color: red;");
div.appendChild(img);
var p = document.createElement("p");
var txt = document.createTextNode("Dies ist ein Absatz");
p.appendChild(txt);
div.appendChild(p);
```

Erzeugung von Knoten: DOM-Stil vs. HTML-Stil

```
var div = document.getElementById("leer_html");
var img = document.createElement("img");
img.src = "img/html-portrait.jpg";
img.alt = "Sir Karl Popper";
img.width = 70;
img.style.borderWidth = "1px";
img.style.borderStyle = "solid";
img.style.borderColor = "red";
div.appendChild(img);
var p = document.createElement("p");
var txt = document.createTextNode("Dies ist ein Absatz");
p.appendChild(txt);  div.appendChild(p);
```

Zugriff auf CSS Inline-Styles

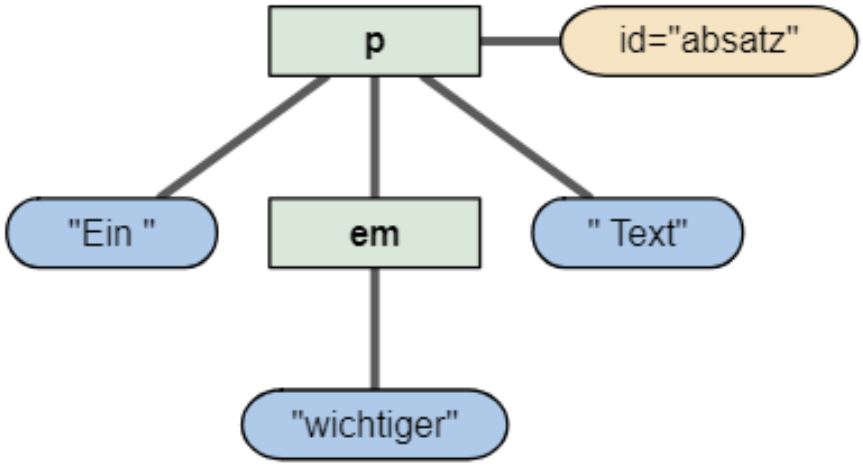
- Zugriff auf Style des HTML-Elements (nicht CSS-Datei):
  - Sie haben gemäß der Kaskadierungsregeln Vorrang vor einer CSS-Datei.
  - Werte sind Strings und enthalten px, %, pt, em.
  - Style ist als Unterobjekt realisiert:
    - `document.getElementById("Hugo").style.fontSize = "12pt";`

- Bildung der CSS-Attributnamen:
  - Bindestriche sind nicht zulässig in JavaScript-Bezeichnern.
  - Also den Bindestrich entfernen und den nächsten Buchstaben groß schreiben:
    - fontSize
    - fontWeight
    - backgroundColor …

Zugriff auf Text

- Text in einem Tag wird in Unterknoten gespeichert:
  - Der Text steht im Attribut **`nodeValue`**
  - Er kann durch Zuweisung geändert werden.

```
<p id="absatz">
Ein <em>wichtiger</em> Text
</p>
```



Beispiel für eine DOM-Manipulation

- Alle Absätze im Dokument gelb einfärben...

```
var allParas = document.querySelectorAll("p");

for (var i = 0; i < allParas.length; i++) {
    allParas[i].style.backgroundColor = "yellow";
}
```

Ein *wichtiger* Text

Ein *wichtiger* Text

Ein *wichtiger* Text

Ein *wichtiger* Text

DOM innerHTML hacking

- Das nachfolgende Prinzip sollte strikt vermieden werden, da es sehr schnell komplex wird:

```
document.getElementById("myid").innerHTML
    +=  "<p>A paragraph!</p>";

document.getElementById("myid").innerHTML
    +=  "<p style='color: red; " +
        "margin-left: 50px;' " +
        "onclick='myOnClick();'>" +
        "A paragraph!</p>";
```

Besser: Neuen DOM-Knoten anlegen...

- **`document.createElement("tag")`**
  erzeugt einen neuen (leeren) DOM-Knoten vom gegebenen Typ.
- **`document.createTextNode("text")`**
  erzeugt einen Textknoten mit dem angegebenen Text als Inhalt.

```
var heading = document.createElement("h2");
heading.innerHTML = "Dies ist eine H2-Überschrift";
heading.style.color = "green";

var vorhanden = document.getElementById("box1");
vorhanden.appendChild(heading);
```

DOM-Baum verändern

- Jeder DOM-Knoten hat eine Reihe von Methoden, um den Baum zu verändern.

Methode	Beschreibung
appendChild(node)	Fügt Knoten am Ende der Kinder an
insertBefore(new, old)	Fügt den Knoten vor dem angegebenen Kindknoten ein
removeChild(node)	Entfernt das angegebene Kind
replaceChild(new, old)	Ersetzt das Kind mit einem neuen Knoten

DOM-Baum verändern: Beispiele zum Einfügen und Löschen

```
var p = document.createElement("p");
p.innerHTML = "A paragraph!";
document.getElementById("myid").appendChild(p);


var bullets = document.getElementsByTagName("li");
for (var i = 0; i < bullets.length; i++) {
    if (bullets[i].innerHTML.indexOf("child") >= 0) {
        bullets[i].parentNode.removeChild(bullets[i]);
    }
}
```