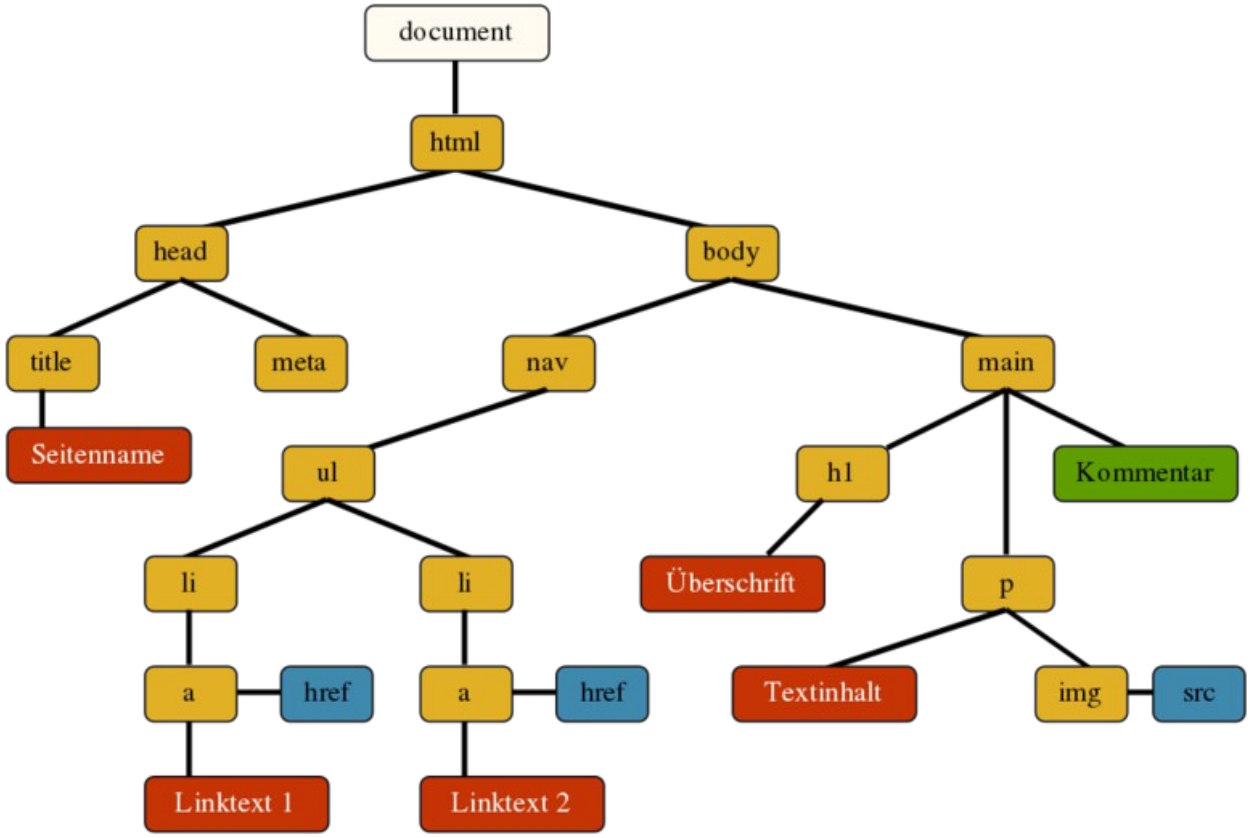
DOM / DOM-Manipulation

DOM

DOM ist die Abkürzung für **D**ocument **O**bject **M**odel.
Darunter versteht man die Schnittstelle zwischen dem HTML und dem Javascript, so dass eine dynamische Webseite entsteht (reines HTML ist statisch, d.h. die Seite wird einmal vom Browser kreiert und dann nicht verändert).

Eine Webseite existiert zunächst nur als reiner Text in der Programmiersprache HTML. Wenn der Browser den Code empfängt, wird dieser verarbeitet. Dies übernimmt der Parser (to parse *engl.*= (grammatikalisch) analysieren).

Der Parser wandelt den HTML-Code in eine Objektstrukur um, welche im Arbeitsspeicher abgelegt wird und der Browser nutzt für alle weiteren Aktionen diese Objektstruktur.
Diese Objektstruktur besteht aus verschachtelten Knotenpunkten, vor allem Elementknoten, Attributknoten und Textknoten.
Diese sind in einer Baumstruktur (tree) angeordnet.
Wie bereits erwähnt, nutzt der Browser für alle weiteren Operationen diese Objektstuktur und nicht den ursprünglichen HTML-Quellcode.
CSS und Javascript beziehen sich nicht auf den HTML-Code als Text, sondern auf die im Arbeitsspeicher abgelegte Baumstruktur.



Quelle: https://wiki.selfhtml.org/wiki/Datei:DOM-1.svg

Die Bestandteile dieser Baumstruktur sind die Knoten, es gibt verschiedene Knotentypen.

Die wichtigsten Knotentypen innerhalb eines HTML-Dokuments sind:

Elementknoten

Attributknoten

Textknoten

Diese Objektstruktur (Baumstruktur) kann durch DOM-Manipulation im Javascript ausgelesen, verändert und erweitert werden

DOM-Manipulation

Mit DOM-Manipulation kann die Objektstrukur im Arbeitsspeicher geändert werden.
Dazu muss das ursprüngliche Dokument ausgelesen werden. Das passiert mit dem Befehl *document*.
Die Elemente, die angesprochen werden sollen, müssen ausgewählt werden. Dies passiert im Javascript. Dazu gibt es zwei unterschiedliche Methoden:

getElement...

getElementById()
greift auf ein einziges Element zu, welches ein dokumentenweites einmaliges ID-Attribut hat. Man erhält ein einziges Element.

getElementsByTagName()
greift auf alle Elemente zu, welche im Dokument den entsprechenden HTML-Tag (z.B. div, ul, p …) haben. Man erhält einen **Array** mit den verschiedenen gefundenen Elementen (selbst wenn nur ein Element gefunden wird, dann erhält man einen Array mit einem Item, wird kein Element gefunden, so erhält man einen leeren Array).

getElementsByClassName()
greift auf alle Elemente zu, welche im Dokument die entsprechende Klasse (class) (beliebige Klassen-Namen wie „movie“, „actors“ …) haben. Man erhält ebenfalls einen Array mit den verschiedenen gefundenen Elementen

Die Attribute müssen dabei als String übergeben werden.

document.getElementByClassName(`movie`)
// holt vom Dokument alle Elemente die die Klasse “movie“ haben. Dies können auch unterschiedliche HTML-Tags sein.

Wenn als Ergebnis ein Array erwartet wird, können auch die einzelnen Items direkt ausgelesen werden.

```
document.getElementByClassName(`movie`)[5]
// holt vom Dokument das sechste Elemente (Index-Nummer 5!) welches die Klasse “movie“ hat.
```

Zur einfacheren Weiterverarbeitung im Javascript empfiehlt es sich, die vom Dokument geholten Objekte in einer Variablen (auch in einer Konstanten) als einzelnen Wert oder als Array zu speichern

```
const movieObjekt = document.getElementByClassName(`movie`);
```

querySelector/querySelectorAll

document.querySelector()
greift auf das erste Element zu, welches das gewählte Attribut hat.
Als Attribut können HTML-Tags, Klassen (Classes) und Ids angegeben werden.
Das Attribut wird auch hier als String übergeben, wie im CSS werden HTML-Tags direkt angesprochen, Klassen brauchen zur Identifizierung einen Punkt und IDs einen Hashtag.

```
document.querySelector(`p`)
// holt vom Dokument den ersten Paragraphen

document.querySelector(`.movie`)
// holt vom Dokument das erste Element, welches die Klasse “movie“ hat.

document.querySelector(`#firstmovie`)
// holt vom Dokument das Element mit der ID “firstmovie“.
```

document.querySelectorAll()
greift auf alle Element zu, welche das gewählte Attribut haben.
Auch hier werden die gewählten Attribute HTML-Tags oder Klassen (Classes) als Strings angegeben.
Klassen brauchen auch hier zur eindeutigen Identifizierung einen Punkt.

Auch hier erhält man einen Array mit den gefundenen Elementen.

Wenn man mit diesen Methoden die Elementknoten ermittelt hat, lässt sich leicht auf diese zugreifen und die Objektstruktur ändern.

createElement...

Mit document.createElement() können neue Elemente erzeugt werden, dazu wird das gewünschte HTML-Element als String im Befehl angegeben.

```
const newParagraph = document.createElement(`p`);
// erzeugt einen einen neuen Paragraphen p für die Objektstruktur.
```

appendChild / insertBefore

Mit diesen Befehlen kann ein kreiertes Element an der gewünschten Stelle in der Objektstruktur angehängt werden.

appendChild()

```
var ulObj = document.getElementById(`liste`);
var newLi = document.createElement("li");
newLi.innerHTML=`Hugh Jackman als letztes`;
ulObj.appendChild(newLi);
```

// Holt vom Document das Element mit der ID “liste“, kreiert ein neues HTML-Element mit dem Variablennamen newLi, “ füllt“ dieses Element mit “Hugh Jackmann als letztes“ und fügt dieses Element als **letztes** Kind-Element an das angesprochene Objekt der Objekt-/Baumstruktur (hier ulObj) an.

InsertBefore()

```
var ulObj = document.getElementById(`liste`);
var newLi2 = document.createElement("li");
newLi2.innerHTML=`Hugh Jackman als erstes`;
ulObj.insertBefore(newLi2, ulObj.firstChild);
```

// Holt vom Document das Element mit der ID “liste“, kreiert ein neues HTML-Element mit dem Variablennamen newLi2, “ füllt“ dieses Element mit “Hugh Jackmann als erstes“ und fügt dieses Element als **erstes** Kind-Element an das angesprochene Objekt der Objekt-/Baumstruktur (hier ulObj) an.

```
var ulObj = document.getElementById(`liste`);
var newLi3 = document.createElement("li");
newLi3.innerHTML=`Hugh Jackman als erstes`;
ulObj.insertBefore(newLi3, ulObj.childNodes[3]);
```

// Holt vom Document das Element mit der ID “liste“, kreiert ein neues HTML-Element mit dem Variablennamen newLi3, “ füllt“ dieses Element mit “Hugh Jackmann als drittes“ und fügt dieses Element als **drittes** Kind-Element an das angesprochene Objekt der Objekt-/Baumstruktur (hier ulObj) an.

ACHTUNG:

Ein einmal kreiertes und in einer Variablen gespeicherte Objekt kann nur einmal mit appendChild oder insertBefore dazu gefügt werden. Um es erneut zuzufügen, muss es unter einem weiteren Variablennamen gespeichert werden und erneut zugefügt werden.

remove()

Mit diesem Befehl kann ein Objekt von der Objekt-/Baumstruktur gelöscht werden.

```
var ulObj = document.getElementById(`liste`);
ulObj.remove();
```

// Holt vom Document das Element mit der ID “liste“ und entfernt es.

Hat mein einen Array von Objekten, so kann das gewünschte Element über die Indexnummer angesprochen und entfernt werden.

```
var ulLiObj = document.getElementsByClassName(`listitem`);
ulLiObj[ulLiObj.length-1].remove();
```

// Holt vom Document alle Elemente mit der Klasse “listitem“ und entfernt das letzte Item.

innerText/innerHTML

Mit diesen Befehlen wird Elementen “Inhalt“ gegeben.

InnerText()

// Gibt einem Element einen Textinhalt oder der bestehende Textinhalt wird überschrieben.

```
var newLi4 = document.createElement("li");
newLi4.innerText="Ich bin ein einfaches List-Item";
```

Hier wird ein neues HTML-Element kreiert, in der Variablen newLi4 gespeichert und dem Element der Text “Ich bin ein einfaches List-Item“ zugewiesen.
Somit ist newLi4 = Ich bin ein einfaches List-Item

InnerHTML()

// Gibt einem Element Code als Inhalt oder der bestehende Code wird überschrieben.

```
var newLi5 = document.createElement("li");
newLi5.innerHTML=`<li><a href="https://www.google.de">Link zu Google</a></li>`;
```

Hier wird ein neues HTML-Element kreiert, in der Variablen newLi5 gespeichert und dem Element der Code “Link zu Google“ zugewiesen.
Somit ist newLi4 = Link zu Google

Questions? You‘re welcome!

holger.zerbe@web.de