




# DOCUMENT OBJECT MODEL (DOM)

Dynamische Manipulation von Webseiten mit JavaScript

# DOCUMENT OBJECT MODEL

- Wir haben bereits gesehen, dass der Browser ein HTML-Dokument in einer baumartigen Datenstruktur verwaltet
- Das **Document Object Model** (DOM) beschreibt diese Datenstruktur und definiert ein API zum Zugriff und zur Manipulation der Datenstruktur per JavaScript
- Das DOM ist analog zu HTML5 standardisiert:
  - [Lebender Standard der WHATWG](#) 
  - Feste Versionen des W3C, zuletzt [DOM4](#) als "Schnappschuss" des lebenden Standards

# DOCUMENT OBJECT MODEL (2)

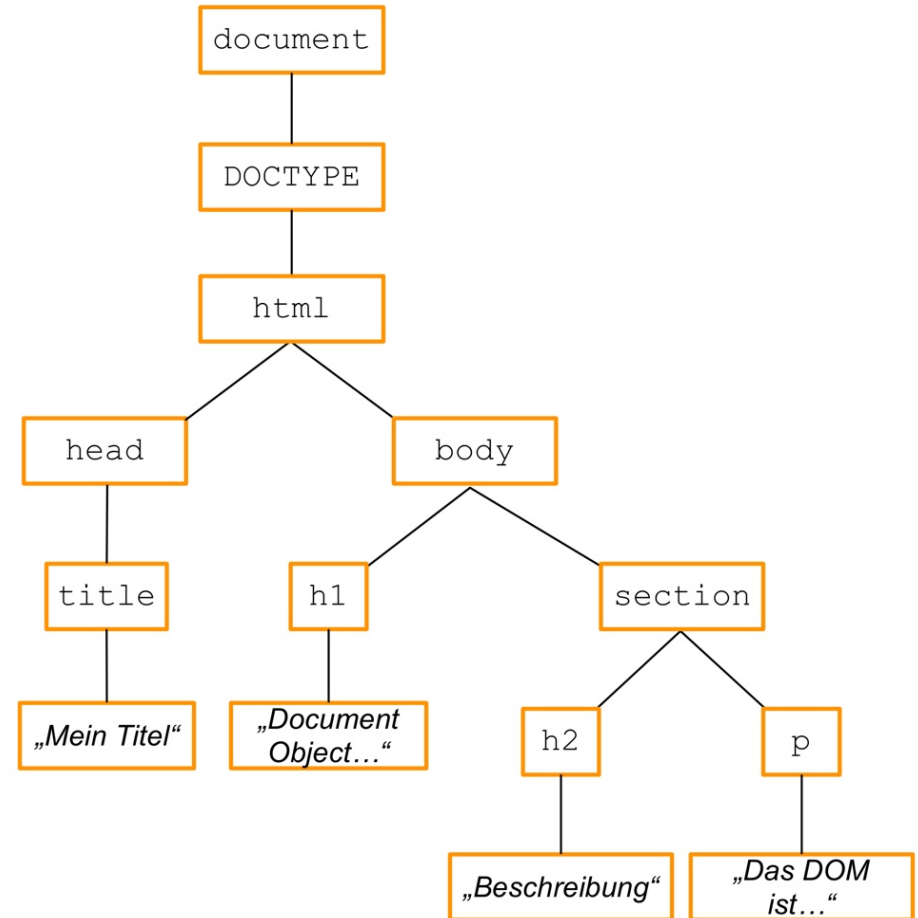
- Das DOM an sich ist für allgemeine Dokumente definiert (*Core DOM*)
- Zudem existieren Ausprägungen für:
  - HTML-Dokumente (*HTML DOM*)
  - XML-Dokumente (*XML DOM*)

! Wir betrachten im Folgenden HTML DOM.

# DOM: SCHEMATISCHER AUFBAU

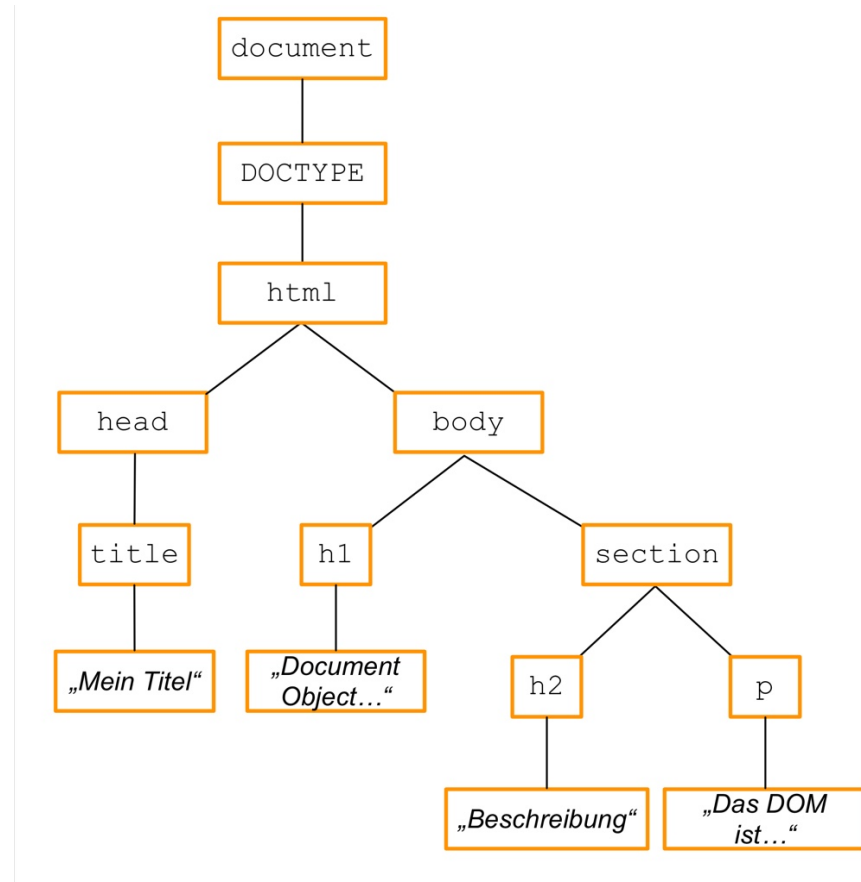
Beispiel:

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
</head>
<body>
  <h1>Document Object Model</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>
      Das DOM ist eine Baumstruktur.
    </p>
  </section>
</body>
</html>
```



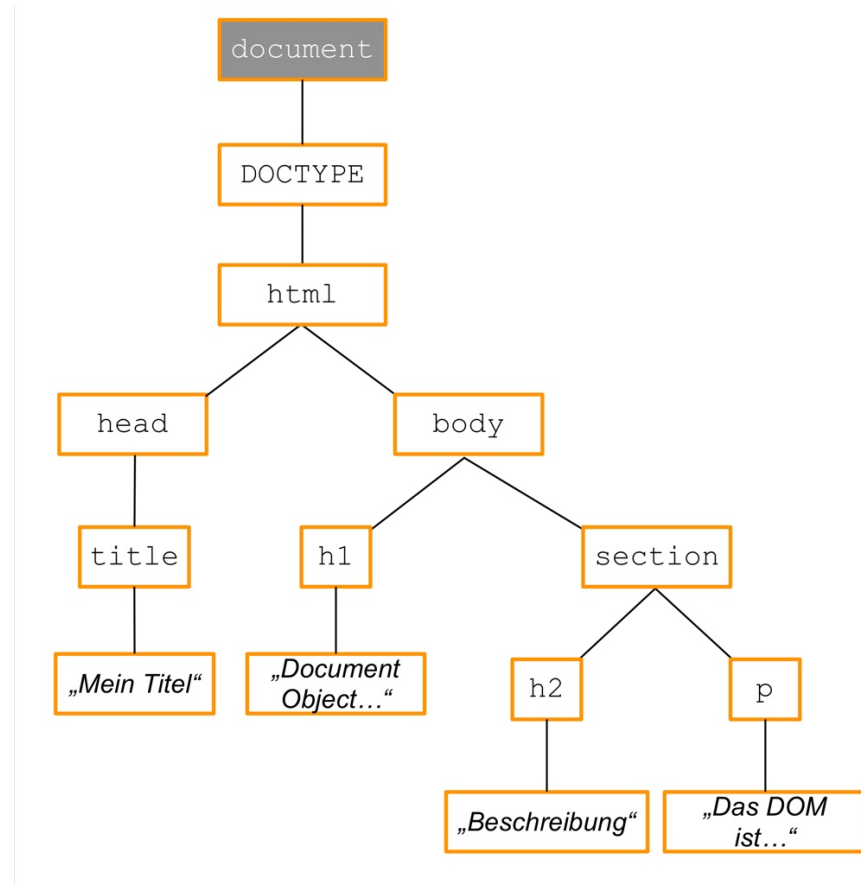
# DOM: KNOTEN UND BEZIEHUNGEN

Das DOM besteht aus **Knoten** (*nodes*), die in Beziehung zueinander stehen



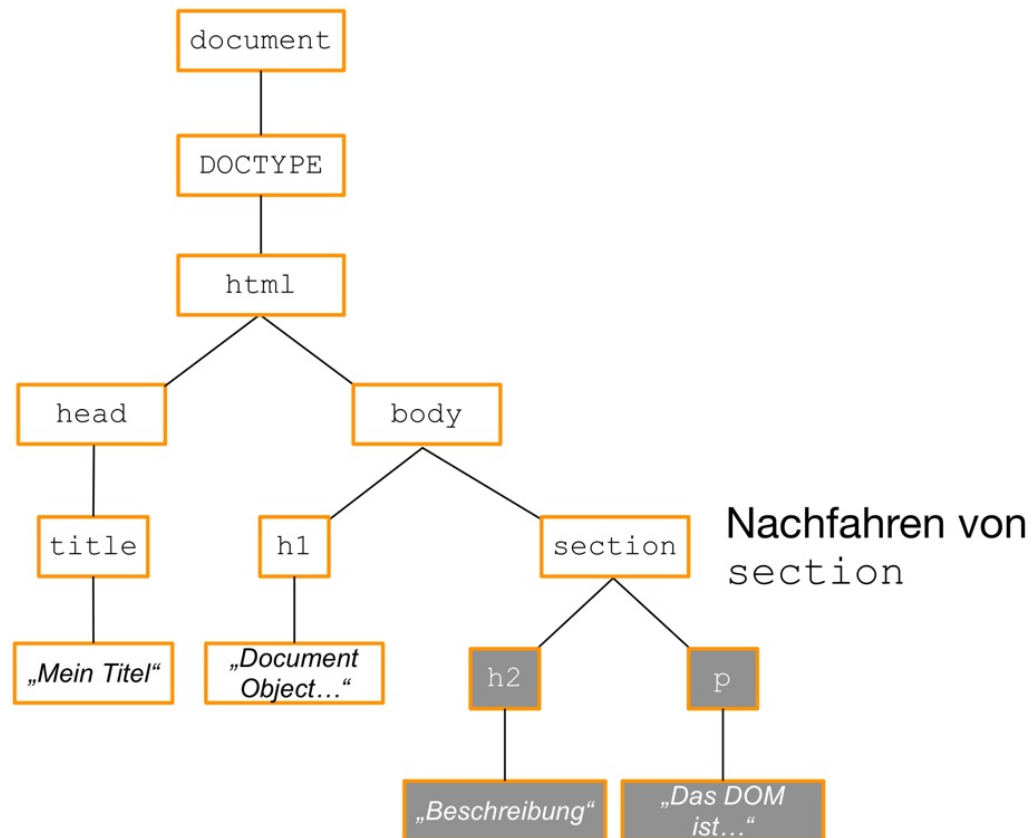
# DOM: KNOTEN UND BEZIEHUNGEN

`document` ist der Wurzelknoten des DOM und ermöglicht den Zugriff auf alle weiteren Knoten



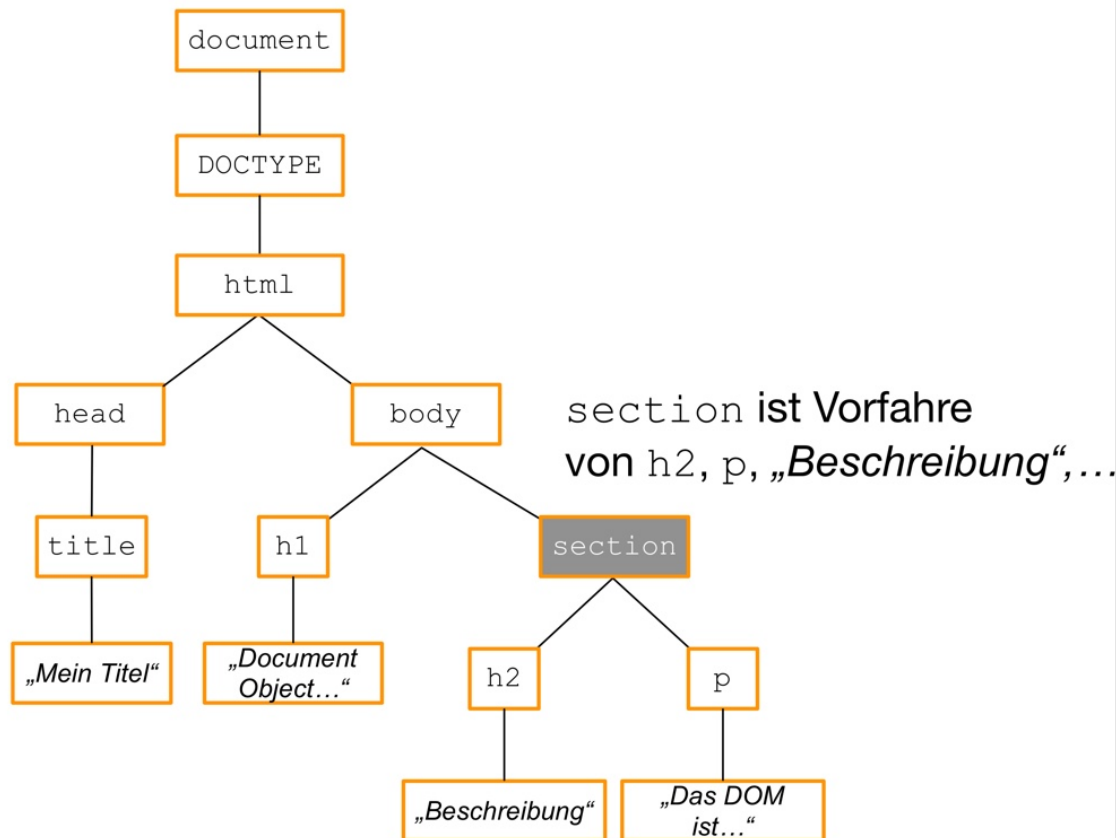
# DOM: KNOTEN UND BEZIEHUNGEN

**Nachfahren** (*descendants*): Alle Unterknoten eines Knotens  
(beliebige Tiefe)



# DOM: KNOTEN UND BEZIEHUNGEN

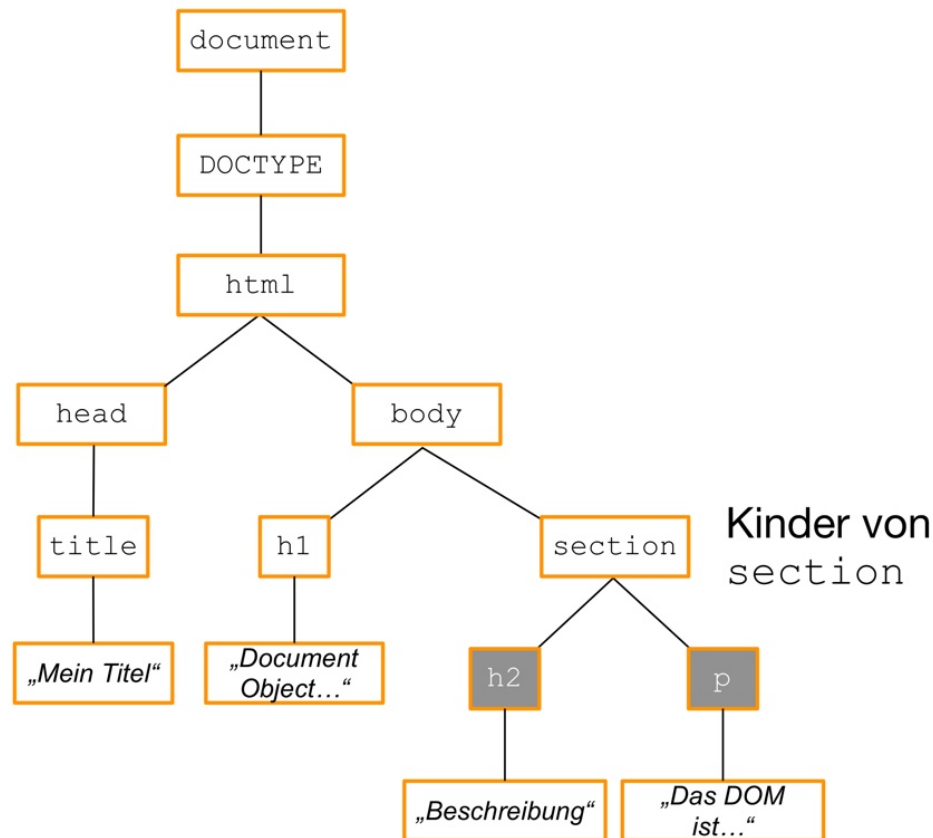
**Vorfahren (*ancestors*):** Alle Oberknoten eines Knotens (beliebige Tiefe)





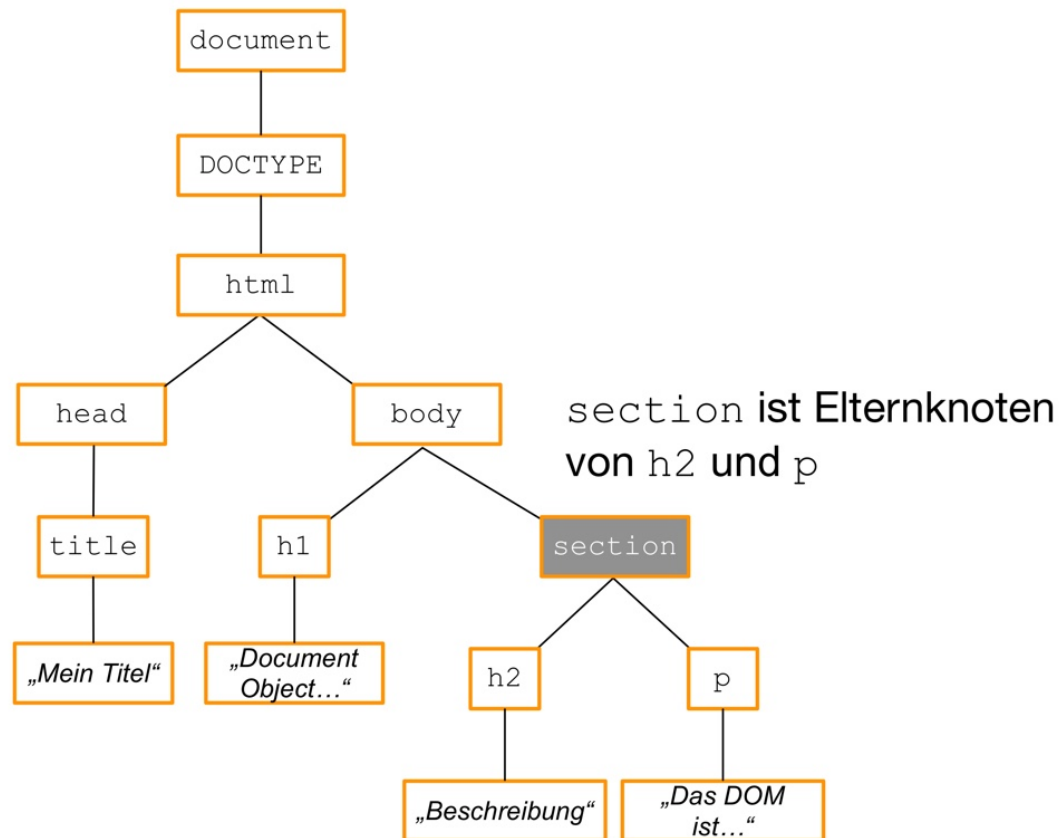
# DOM: KNOTEN UND BEZIEHUNGEN

Kinder (*children*): Direkte Nachfahren eines Knotens  
(Tiefe 1)



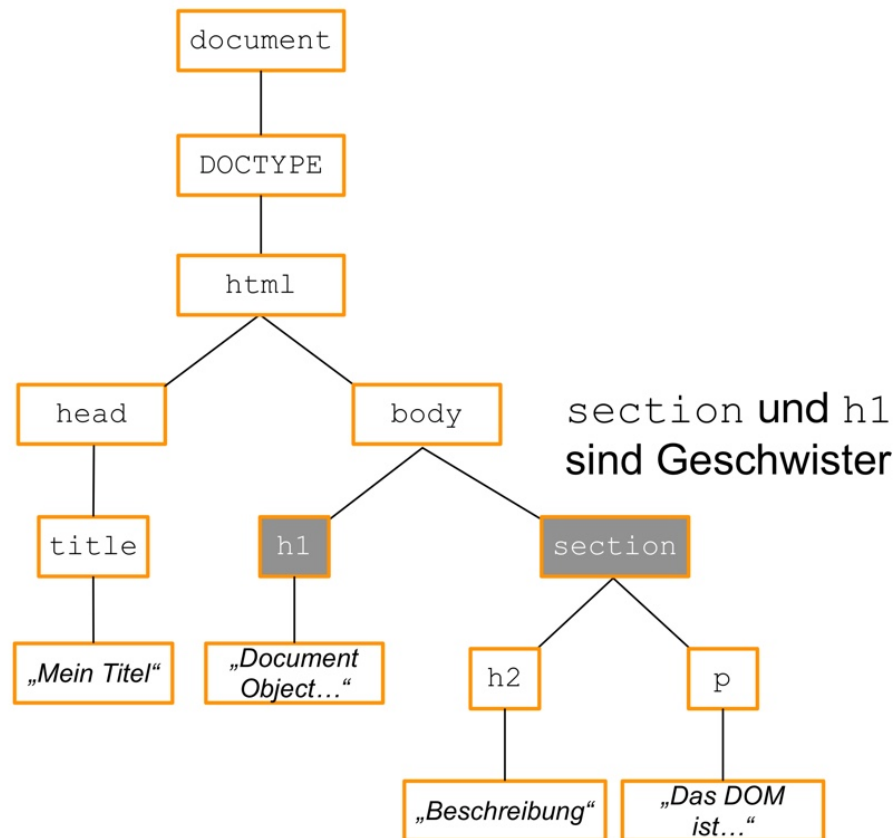
# DOM: KNOTEN UND BEZIEHUNGEN

Elternknoten (*parent*): Direkter Vorfahre eines Knotens  
(Tiefe 1)



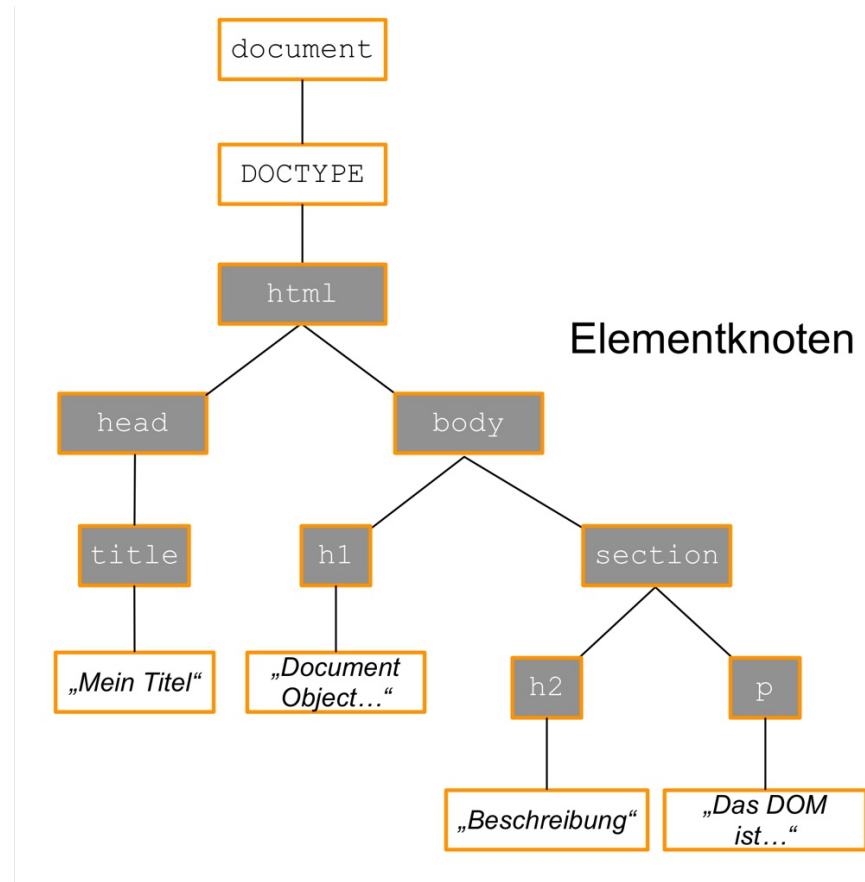
# DOM: KNOTEN UND BEZIEHUNGEN

**Geschwister (*siblings*):** Kinder des selben Elternelements  
(Tiefe 1)



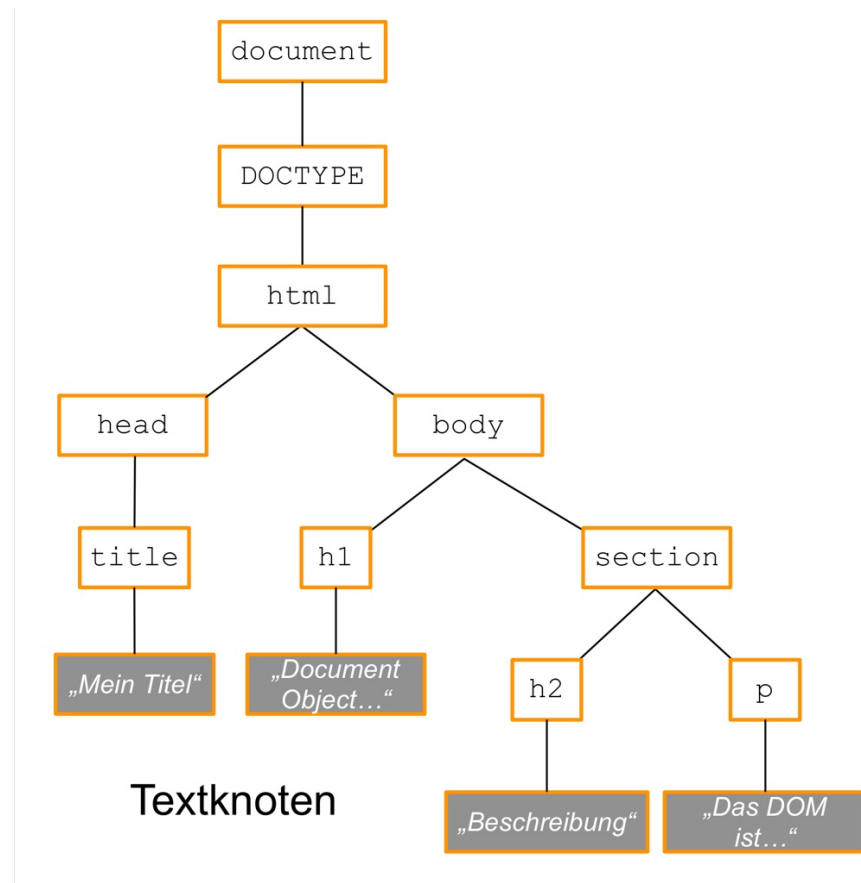
# DOM: KNOTENTYPEN

DOM unterscheidet verschiedene Typen von Knoten,  
Beispiele:



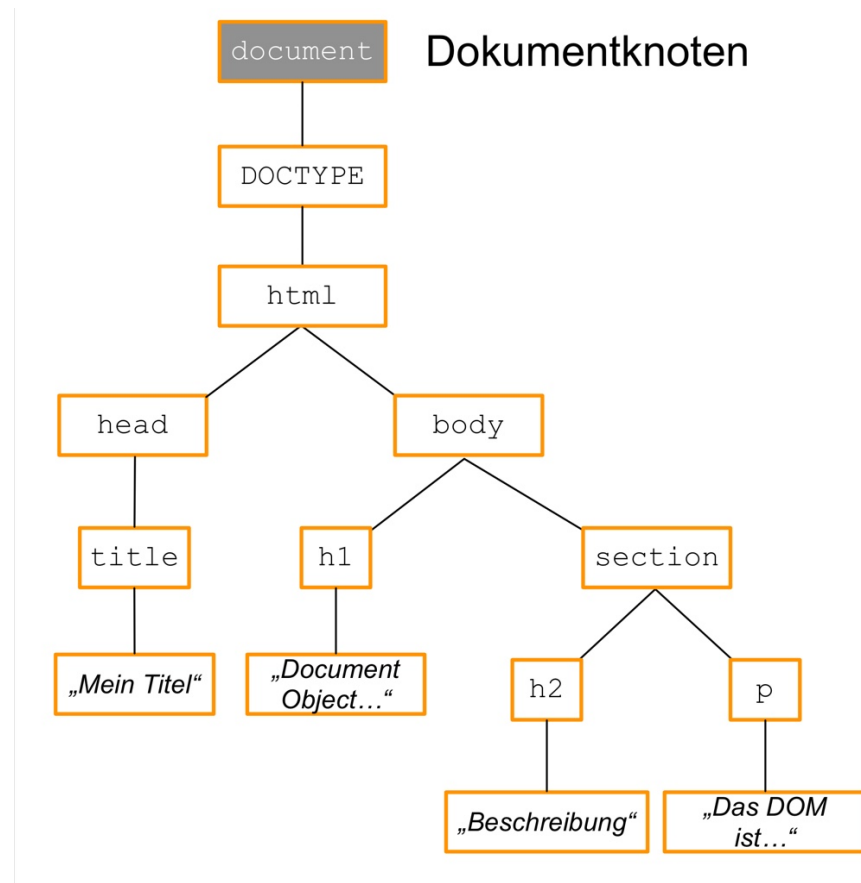
# DOM: KNOTENTYPEN

DOM unterscheidet verschiedene Typen von Knoten,  
Beispiele:



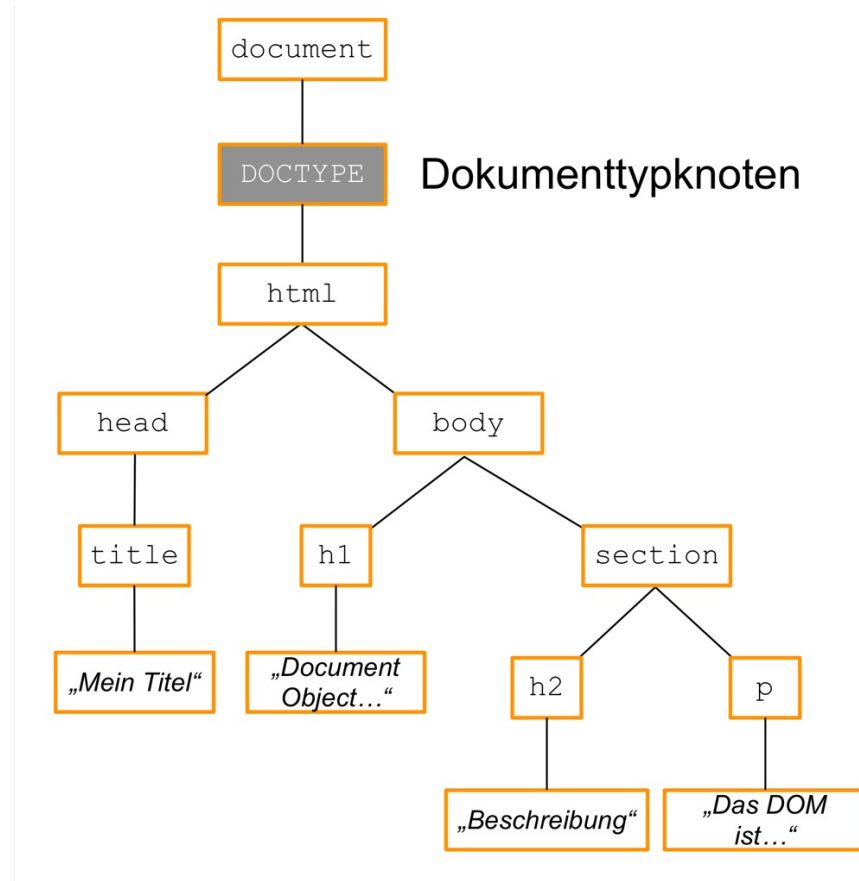
# DOM: KNOTENTYPEN

DOM unterscheidet verschiedene Typen von Knoten,  
Beispiele:



# DOM: KNOTENTYPEN

DOM unterscheidet verschiedene Typen von Knoten,  
Beispiele:



# KNOTEN

- Technisch ist jeder Knoten im DOM ein Node-Objekt (bzw. ein Subtyp davon wie z.B. `HTMLElement`)
- Node bietet Eigenschaften zum Umgang mit einem konkreten Knoten, z.B. zur Navigation im DOM:

Eigenschaft	Bedeutung
<code>parentNode</code>	Zugriff auf den Elternknoten
<code>childNodes</code>	Liefert eine Liste der Kinder
<code>previousSibling/nextSibling</code>	Zugriff auf den vorhergehenden/nachfolgenden Geschwisterknoten



# KNOTEN (2)

- Node bietet auch Eigenschaften zur Analyse eines Knotens, z.B.:

Eigenschaft	Bedeutung
nodeType	Knotentyp als Zahl (z.B. 1=Elementknoten, 3=Textknoten)
nodeName	Name des Knotens, bei einem HTML-Element z.B. der Tag-Name in Großbuchstaben (z.B. STRONG)
nodeValue	Bei Textknoten der Inhalt des Knotens, bei anderen Knotentypen wie z.B. Elementknoten null

# document-OBJEKT

- Über das document-Objekt kann per JavaScript auf das DOM zugegriffen werden
- document bietet Eigenschaften zum Zugriff auf bestimmte Informationen und Knoten, z.B.:

Eigenschaft	Bedeutung
<code>document.body</code>	body-Element des Dokuments
<code>document.forms</code>	Liste aller form-Elemente des Dokuments
<code>document.images</code>	Liste aller Bilder des Dokuments
<code>document.title</code>	title-Element des Dokuments
<code>document.URL</code>	URL des Dokuments

# document-OBJEKT (2)

- Weiterhin können über das document-Objekt Elemente im DOM selektiert werden:

Methode	selektiert...
<code>document.getElementById(id)</code>	Element mit einer bestimmten ID
<code>document.getElementsByTagName(name)</code>	Liste aller Elemente mit bestimmtem Tag-Namen
<code>document.getElementsByClassName(class)</code>	Liste aller Elemente mit bestimmtem class-Attribut
<code>document.getElementsByName(name)</code>	Liste aller Elemente mit bestimmtem name-Attribut
<code>document.querySelector(selector)</code>	das erste Element, auf das der CSS-Selektor <code>selector</code> zutrifft
<code>document.querySelectorAll(selector)</code>	Liste aller Elemente, auf die der CSS-Selektor <code>selector</code> zutrifft

# document-OBJEKT: BEISPIELE

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <section>
    <h2>API</h2>
    <p>
      DOM bietet eine API für den Zugriff.
    </p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
// Ausgabe: Mein Titel
console.log(document.title);

// Ausgabe: http://localhost/seite.html
console.log(document.URL);

// Ausgabe: BODY
console.log(document.body.nodeName);
```

# document-OBJEKT: BEISPIELE (2)

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <section>
    <h2>API</h2>
    <p>
      DOM bietet eine API für den Zugriff.
    </p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
document.getElementById("ueberschrift")
```

! Die gelb markierten HTML-Elemente werden durch die oben stehende JavaScript-Anweisung selektiert.

# document-OBJEKT: BEISPIELE (3)

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <section>
    <h2>API</h2>
    <p>
      DOM bietet eine API für den Zugriff.
    </p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
document.getElementsByTagName("h2")
```

! Die gelb markierten HTML-Elemente werden durch die oben stehende JavaScript-Anweisung selektiert.

# document-OBJEKT: BEISPIELE (4)

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <section>
    <h2>API</h2>
    <p>
      DOM bietet eine API für den Zugriff.
    </p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
document.querySelector("#ueberschrift")
```

! Die gelb markierten HTML-Elemente werden durch die oben stehende JavaScript-Anweisung selektiert.

# document-OBJEKT: BEISPIELE (5)

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <section>
    <h2>API</h2>
    <p>
      DOM bietet eine API für den Zugriff.
    </p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

script.js

```
document.querySelectorAll("section p:last-child")
```

! Die gelb markierten HTML-Elemente werden durch die oben stehende JavaScript-Anweisung selektiert.



# DOM-MANIPULATION

In Kombination mit den kennengelernten Möglichkeiten zum Navigieren und Durchsuchen kann das DOM mittels JavaScript manipuliert werden, z.B.:

1. Elemente ändern
2. Elemente hinzufügen
3. Elemente ersetzen oder löschen

! Wir betrachten weiterhin HTML DOM, also sind hier (und im Folgenden) HTML-Elemente gemeint.

# ELEMENTE ÄNDERN

- Elemente (die z.B. zuvor über die Methoden von `document` selektiert wurden) können über verschiedene Eigenschaften modifiziert werden:

Eigenschaft	Bedeutung
<code>innerHTML</code>	HTML-Inhalt des Elements
<code>textContent</code>	Textinhalt des Elements
<i>Attributname</i>	Jedes Attribut eines Elements steht als gleichnamige Eigenschaft zur Verfügung, z.B. <code>href</code> , <code>src</code> <sup>*</sup>
<code>style.Eigenschaft</code>	Jede CSS-Eigenschaft eines Elements steht als gleichnamige Eigenschaft zur Verfügung, z.B. <code>color</code> , <code>fontSize</code> <sup>*</sup>

<sup>\*</sup> Namen mit Bindestrich müssen in Camel-Case-Notation umgeformt werden, um Verwechslung mit einer arithmetischen Subtraktion zu vermeiden (z.B. `font-size` wird `fontSize`).

# ELEMENTE ÄNDERN: BEISPIELE

script.js

```
// Erstes p-Element selektieren und Inhalt ändern
document.querySelector("p").innerHTML = "Anderer Text";
/* Element mit ID "ueberschrift" selektiert und
   CSS-Eigenschaften ändern */
document.getElementById("ueberschrift").style.color
    = "red";
document.getElementById("ueberschrift").style.backgroundColor
    = "blue";
// Erstes img-Element selektieren und "src"-Attribut ändern
document.getElementsByTagName("img")[0].src="picture-small.jpg";
```

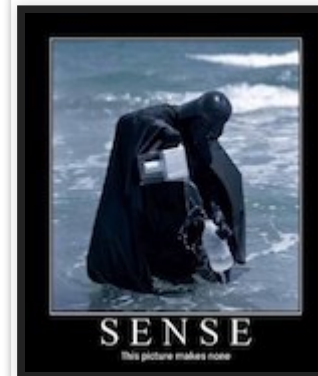
seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
    
  </section>
  <script src="script.js"></script>
</body>
</html>
```

**DOM**

## Beschreibung

Anderer Text



# ELEMENTE HINZUFÜGEN

- Das document-Objekt bietet Methoden zum Erzeugen von Knoten, z.B.:

Methode	Bedeutung
<code>document.createElement ( tagName )</code>	Neuen Elementknoten mit Name <code>tagName</code> erzeugen
<code>document.createTextNode ( content )</code>	Neuen Textknoten mit Inhalt <code>content</code> erzeugen

# ELEMENTE HINZUFÜGEN (2)

- Elemente bieten Methoden, um neue Knoten in das DOM einzufügen z.B.:

Methode	Bedeutung
<code>element.prepend(nodes)</code>	Füge einen oder mehrere Knoten vor dem ersten Kind von <code>element</code> ein
<code>element.append(nodes)</code>	Füge einen oder mehrere Knoten nach dem letzten Kind von <code>element</code> ein
<code>element.before(nodes)</code>	Füge einen oder mehrere Knoten vor <code>element</code> ein
<code>element.after(nodes)</code>	Füge einen oder mehrere Knoten nach <code>element</code> ein

# ELEMENTE ERSETZEN ODER LÖSCHEN

Methode	Bedeutung
<code>element.replaceWith(nodes)</code>	<code>element</code> durch einen oder mehrere andere Knoten ersetzen ( <code>element</code> wird dabei gelöscht)
<code>element.remove()</code>	<code>element</code> löschen

# ELEMENTE HINZUFÜGEN UND LÖSCHEN: BEISPIELE

script.js

```
// Neuen Abschnitt erzeugen und unterhalb der Überschrift einfügen
let abschnitt = document.createElement("p");
abschnitt.append(document.createTextNode("Neuer Abschnitt!"));
document.getElementById("ueberschrift").after(abschnitt);

// h2-Element löschen
document.querySelector("h2").remove();
```

seite.html

```
<!DOCTYPE html>
<html>
<head>
  <title>Mein Titel</title>
  <meta charset="utf-8">
</head>
<body>
  <h1 id="ueberschrift">DOM</h1>
  <section>
    <h2>Beschreibung</h2>
    <p>Das DOM ist eine Baumstruktur.</p>
  </section>
  <script src="script.js"></script>
</body>
</html>
```

## DOM

Neuer Abschnitt!

Das DOM ist eine Baumstruktur.