DOM

DOM = Abkürzung für **Document Object Model**
ist eine Sammlung von Objekten, Methoden, Funktionen und Attributen

Browser stellt uns so Möglichkeit zur Verfügung, HTML-Seiten dynamisch zu verändern

DOM unterscheidet zwischen **Knoten** (Node) im Allgemeinen und speziellen Knotentypen, wie Element-Knoten (Element), Text-Knoten, Kommentar-Knoten usw.

für Praxis normalerweise nur Element-Knoten relevant

Element-Knoten (Element) = Spezialisierung eines Knotens (Node)
auf Element-Knoten stehen neben Element-Eigenschaften (z. B. children) auch alle Node-Eigenschaften (z. B. parentNode) zur Verfügung

document-Knoten = Spezialfall eines Knotens
repräsentiert das komplette HTMLDokument und stellt Methoden wie createElement bereit

Eigenschaften des document-Objektes

Eigenschaft	Beschreibung
document.title	Titel des aktuellen Dokumentes
document.lastModified	Datum der letzten Änderung des aktuellen Dokumentes
document.URL	URL des aktuellen Dokumentes
document.cookie	Liste aller Cookies des aktuellen Dokumentes
document.forms	Liste aller Formulare des aktuellen Dokumentes
document.images	Liste aller Bilder des aktuellen Dokumentes
document.links	Liste aller Links des aktuellen Dokumentes
Document.anchors	Liste aller Links des aktuellen Dokumentes, die id oder name haben

DOM-Manipulation

Funktion document.querySelector Auswahl von HTML-Elemente innerhalb HTML-Dokumentes
Parameter = beliebiger CSS-Selektor
document.querySelector('h1') liefert erstes h1 -Element als JS-Objekt zurück - sogenanntes Element-Objekt

Objekte in JS haben Eigenschaften (engl. properties)
wichtige Eigenschaft, speziell von Element-Objekten = innerHTML

document.querySelector() liefert Element-Objekt zurück, dessen Eigenschaft innerHTML wir abfragen

Wert von innerHTML = String mit innerem HTML des jeweiligen Elementes (tags)

Eigenschaften lassen sich wie Variablen verwenden – lassen sich an Variablen binden und können neue Werte zugewiesen bekommen: document.querySelector('h1').innerHTML = "Hello again!";

Suchen und Finden im DOM

ganzes HTML-Dokument = Sammlung von HTML-Elementen, die in Baumstruktur angeordnet
Äquivalent dazu sind passenden JS-Objekte ebenfalls im Baum angeordnet - Document Object Model, kurz DOM

document.querySelector() selektiert immer erstes Element, das übergebenen CSS-Selektor entspricht, und gibt es zurück - Selektor kann beliebiger valider CSS-Selektor sein

siehe auch: https://www.w3schools.com/css/css_selectors.asp

Wichtigste CSS-Selektoren

Name	Beispiel	Beschreibung
Universal selector	*	Selektiert ein beliebiges Element. Diesen Selektor sollten Sie nicht ohne weitere Einschränkungen verwenden, da das Selektieren aller Elemente einer Webseite sehr viel Rechenzeit in Anspruch nehmen kann.
Type selector	h1, p, ul, li	Selektiert alle Elemente eines bestimmten Typs.
ID selector	#id	Selektiert genau ein Element mit der gegebenen id.
Class selector	.class	Selektiert alle Elemente, denen die angegebene Klasse zugewiesen wurde. Beachte, dass ein Element mehrere Klassen haben kann.
-	h1, p, .go	Das Komma ist selbst kein Selektor, sondern erlaubt es, mehrere Selektoren zu kombinieren. Im Beispiel würden alle h1-Elemente, alle p-Elemente und alle Elemente mit der Klasse go selektiert.

können mit querySelectorAll() mehrere Elemente gleichzeitig auslesen oder verändern

querySelectorAll() liefert alle zum Selektor passenden Elemente als NodeList

NodeList = Objekt, das mehrere Elemente (Knoten/Nodes) in Liste zusammenfasst

Begriff Knoten ist im Sinne des HTML-Baums gemeint
Knoten hat immer nur ein Eltern-Element, kann aber beliebig viele (oder keine) Kind-Elemente haben

NodeList verhält sich ähnlich wie Array - können mit Index-Operator darauf zugreifen:
document.querySelectorAll('h2')[0].innerHTML liefert Text der ersten h2 -Überschrift zurück

kann mit Eigenschaft length abfragen, wie viele (wenn überhaupt) Knoten der Selektor gefunden hat:
document.querySelectorAll('h2').length

Attribut-Selektoren

Selektor	Beispiel zu 	Beschreibung
[attr]	[src]	Selektiert Elemente mit dem angegebenen **Attribut**. Der Attributwert spielt dabei keine Rolle (attribute presence).
[attr=val]	[src='bild.png']	Selektiert Elemente, die das angegebene Attribut (src) mit **genau** dem angegebenen Wert (bild.png) enthalten (exact attribute match).
[attr*=val]	[src*='bild']	Selektiert Elemente, bei denen das angegebene Attribut (src) den Wert (bild) als **Teilstring** enthält (substring attribute match).
[attr^=val]	[src^='bild']	Selektiert Elemente, bei denen das angegebene Attribut (src) mit dem Wert (bild) **beginnt**.
[attr$=val]	[src$='png']	Selektiert Elemente, bei denen das angegebene Attribut (src) auf den angegebenen Wert (png) **endet**.

Combinator Selectors

Name	Beispiel	Beschreibung
Descendant combinator (Leerzeichen)	li span	Selektiert Elemente, die die Nachfahren eines anderen angegebenen Elements sind - völlig unabhängig davon, wie viele Ebenen dazwischen liegen.
Adjacent sibling combinator (+)	li#listItem1 + li	Element auswählen, das im HTML-Quelltext direkt nach dem angegeben Element auf der gleichen Ebene (Geschwister-Element) folgt. Falls das im Dokument nachfolgende Element nicht dem hinter dem + Symbol angegebenen entspricht, wird nichts selektiert.
General sibling combinator (~)	li#**listItem2** ~ li	Alle Elemente auswählen, die im Dokument nach dem angegebenen Element auf der gleichen Ebene (Geschwister-Element) folgen.
Child combinator (>)	li > span	Mit Kind-Selektoren Elemente auswählen, die direkte »Kinder« von übergeordneten Eltern-Elementen sind. Im Beispiel werden nur span-Elemente selektiert, die sich direkt innerhalb eines li-Elements befinden. Elemente, die sich innerhalb des span-tags, dort aber in einem anderen Element befinden, werden nicht selektiert.

Pseudo Classes

Name	Beispiel	Beschreibung
:first-child	ul li:first-child	Pseudo-Klasse spricht das erste Kind-Element eines übergeordneten Containers an, falls es sich um ein Element vom angegeben Tag-Typ handelt. Sollte das erste Kindelement nicht den richtigen Tag-Typ haben, wird nichts selektiert.
:nth-child(x)	ul li:nth-child(5)	Die Pseudo-Klasse spricht das x-te Kind-Element an - im Beispiel das fünfte Element innerhalb von ul falls es vom Tag-Typ li ist.
:last-child	ul li:last-child	Äquivalent zu :first-child wird hier das letzte Kind-Element selektiert, sofern es vom richtigen Tag-Typ ist.
:only-child	#article p:only-child	Diese Klasse selektiert ein Element nur, wenn es sich um das einzige Kind eines angegebenen Eltern-Elementes handelt.
:empty	div:empty	Diese Pseudo-Klasse spricht nur Elemente an, die ohne Kind-Elemente sind. Inhalt in Form von Text wird dabei ebenfalls als Kind-Element betrachtet.
:not(selector)	:not(span)	Negation. Wählt Elemente aus, wenn sie nicht dem in Klammern angegebenen Selektor entsprechen. Im Beispiel würde alles außer span-Elementen ausgewählt.

DOM, Zugriff auf Text

Text lesen/schreiben/löschen

(Element).firstChild.nodeValue

(Element).textContent (evtl. vorhandenes Markup wird unterdrückt)

(Element).innerText (gibt "gerenderte" Text-Inhalte – also so, wie es der User im Browser sehen / von dort kopieren könnte – ausgeblendete Texte werden bsw. nicht angezeigt, Leerstellen ebenso)

(Element).innerHTML (evtl. vorhandenes Markup bleibt erhalten)

Document vs. Element

querySelector() und querySelectorAll(() lassen sich auf document global und auf einzelnen Elementen lokal verwenden

global - Selektor durchsucht gesamtes Dokument
lokal - nur Kinder eines Element-Knotens werden durchsucht

Zusammenfassung

Methode	Beschreibung	Interface/API
[element].querySelector()	gibt das erste Element zurück, das zum angegebenen Selektor passt. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement querySelector() aufgerufen wurde.	Element
[element].querySelectorAll()	gibt alle Elemente zurück, die zum angegeben Selektor passen. Durchsucht nur Elemente innerhalb des HTML-Teilbaums, auf dessen Wurzelelement querySelectorAll() aufgerufen wurde.	Element
document.querySelector()	gibt das erste Element zurück, das zum angegeben Selektor passt. Durchsucht das gesamte HTML-Dokument.	Document
document.querySelectorAll()	gibt alle Elemente zurück, die zum angegeben Selektor passen. Durchsucht das gesamte HTML-Dokument.	Document
document.body	gibt das body-Element zurück.	Document

DOM, Zugriff auf class-Attribut

className

Eigenschaft className gibt am Element-Objekt anliegende Klassen als String zurück
Änderungen funktionieren nur über Zeichenkettenverarbeitung

classList

NodeObject hat neben Eigenschaft innerHTML auch Eigenschaft classList, die CSS-Klassen des Objekts als DOMTokenList zurückgibt

DOMTokenList verhält sich ähnlich wie Array - können z. B. Eigenschaft length verwenden, um Anzahl der CSS-Klassen zu ermitteln

Methoden von classList

add() zum hinzufügen, remove() zum entfernen, toggle() zum umschalten und contains() zum prüfen

Klasse mit add() hinzufügen:
[element].classList.add('klassenname');

vorhandene Klasse lässt sich nicht nochmals hinzufügen, da in DOMTokenList gleicher Wert nicht mehrfach vorkommen kann

Anzahl der am Element anliegenden Klassen bestimmen:
[element].classList.length

Klasse mit remove() wieder entfernen:
[element].classList.remove('klassenname');

Klasse ersetzen mit replace():
[element].classList.replace("oldClassName","newClassName");

toggle() schaltet zwischen Hinzufügen und Entfernen hin und her
folgenden Code mindestens zweimal eingeben, um Effekt zu beobachten:
[element].classList.toggle('klassenname');

contains() prüft, ob bestimmte Klasse vorhanden ist (true/false):
[element].classList.contains('klassenname'));

Attribute

Name des Attributs	Elemente, für die Attribut oft verwendet wird	Beschreibung
alt	area, img, input	Alternativer Text, z. B. falls ein Bild nicht dargestellt werden kann
checked	input	Gibt an, ob ein Radiobutton gewählt oder eine Checkbox angehakt ist
cols	textarea	Anzahl der Spalten
disabled	button, input, option, select, textarea	Gibt an, ob ein Element deaktiviert ist
href	a, link	Die URL einer verlinkten Ressource
max	progress, input	Der Maximalwert — bei input-Elementen ist das nur für type="number" und type="range" sinnvoll
maxlength	input, textarea	Maximal erlaubte Zeichen in einem Element
min	input	Der Minimalwert — bei input-Elementen ist das nur für type="number" und type="range" sinnvoll
multiple	input, select	Erlaubt bei true mehrere Eingaben für input type="email" oder input type="file". Bei select können mit strg/cmd mehrere Optionen gewählt werden
name	button, form, fieldset, input, select, textarea	Name des Elements. Dieses Attribut wird hauptsächlich für serverseitige Anwendungen zur Identifizierung genutzt
placeholder	input, textarea	Voreingestellter Text, der dem Anwender einen Hinweis zu den erwarteten Eingaben gibt
readonly	input, textarea	Gibt an, ob ein Element bearbeitet werden kann
rel	a, link	Definiert die Beziehung zwischen dem Ziel- und dem Link-Objekt. Kann auch für eigene Beziehungstypen verwendet werden, z. B. interne Links für eine JS-Bildergalerie
required	input, select, textarea	Gibt an, ob ein Element ein Pflichtfeld darstellt.
rows	textarea	Anzahl der Zeilen
selected	option	Gibt an, ob eine option selektiert ist
size	select	Anzahl der sichtbaren Optionen bei einer select-Box
src	img	URL der Bilddatei
start	ol	Definiert die Startnummer einer Liste, wenn etwas anderes als 1 gewünscht ist
step	input	Die Schrittweite — bei input-Elementen ist das nur für type="number" und type="range" sinnvoll
type	button, input	Typ des Elements

value	button, option, input, progress	Wert des Elements
tabindex	auf allen Elementen möglich	Überschreibt die herkömmliche Reihenfolge der angesteuerten (Formular-)Elemente mit Hilfe der Tab-Taste
title	auf allen Elementen möglich	auf allen Elementen möglich Kleine Text-Box (Tooltip), die beim »Hover« angezeigt wird

Attribute auslesen und ändern

Element-Objekte haben Methoden getAttribute() und setAttribute()
zum Auslesen und Verändern von Attributwerten

[element].getAttribute('value')

[element].setAttribute('value', 8) // Änderung oder setzen eines Attributes value auf 8

mit Eigenschaft attributes, Rückgabe der Liste aller verfügbaren Attribute:
[element].attributes // liefert NamedNodeMap mit Eigenschaften

Rückgabewert = sogenannte NamedNodeMap (Objekt, das alle Attribute und deren Werte enthält)

IDL-Attribut vs. Content-Attribut

können mit Methoden getAttribute() und setAttribute() nahezu beliebige Attribute auslesen und ändern

oft stellt DOM Attribute auch direkt als JS-Properties auf HTML-Element-Objekt zur Verfügung

können Attribute von Elementen auch auslesen mit: [element].attribut
und ändern mit: [element].attribut = wert

Attribute, die sich mit Zugriffs-Methoden auslesen lassen, heißen Content-Attribute
Attribute, die direkt über Eigenschaften anzusprechen, heißen IDL-Attribute
IDL-Attribute greifen auf zugrundeliegenden Content-Attribute

Wesentlicher Unterschied:

- Content-Attribute immer vom Datentyp string
- IDL-Attribute können unterschiedliche Datentypen haben

boolesche Attribute

bei booleschen Attributen etwas aufpassen

boolesche Attribute sind bsw. required, checked, selected, defer, async, disabled

(geht bei booleschen Attributen nur um Frage, ob Attribut vorhanden ist oder nicht)

IDL-Attribut hat Typ boolean: typeof [element].disabled // => boolean

Attribute setzen durch Zuweisung true, deaktivieren/entfernen mit false
[element].attribute = true/false

für Content-Attribute, ist korrekte Schreibweise zum setzen:
[element).setAttribute(attribut, attribut)

für Content-Attribute, ist korrekte Schreibweise zum entfernen:
[element).removeAttribute(attribut)

[element].toggleAttribute(attribut) // boolsche Attribute toggeln

IDL-Attribut

wenn Wert eines Attributes sich nicht ändert, IDL- Attribut = Content-Attribut
Unterschied dann, wenn durch JS oder User eine Änderung des Wertes erfolgt!

IDL-Attribut liefert aktuelle Werte – Änderungen werden berücksichtigt

Content-Attribut bezieht sich auf Wert aus HTML-Quellcode – sozusagen dem Standardwert im Input-Element - User-Eingabe wird vernachlässigt

Empfehlung

Verwende immer die IDL-Attribute - außer es wird gezielt ein Originalwert aus HTML-Code benötigt

Zugriff auf IDL-Attribute aus mehreren Gründen vorzuziehen:

- erhalten einen genaueren Datentyp und nicht nur String
- erhalten aktuelle, möglicherweise geänderte Werte (z. B. durch Eingaben)
- Verhalten ist konsistenter, wir müssen weniger Sonderfälle beachten