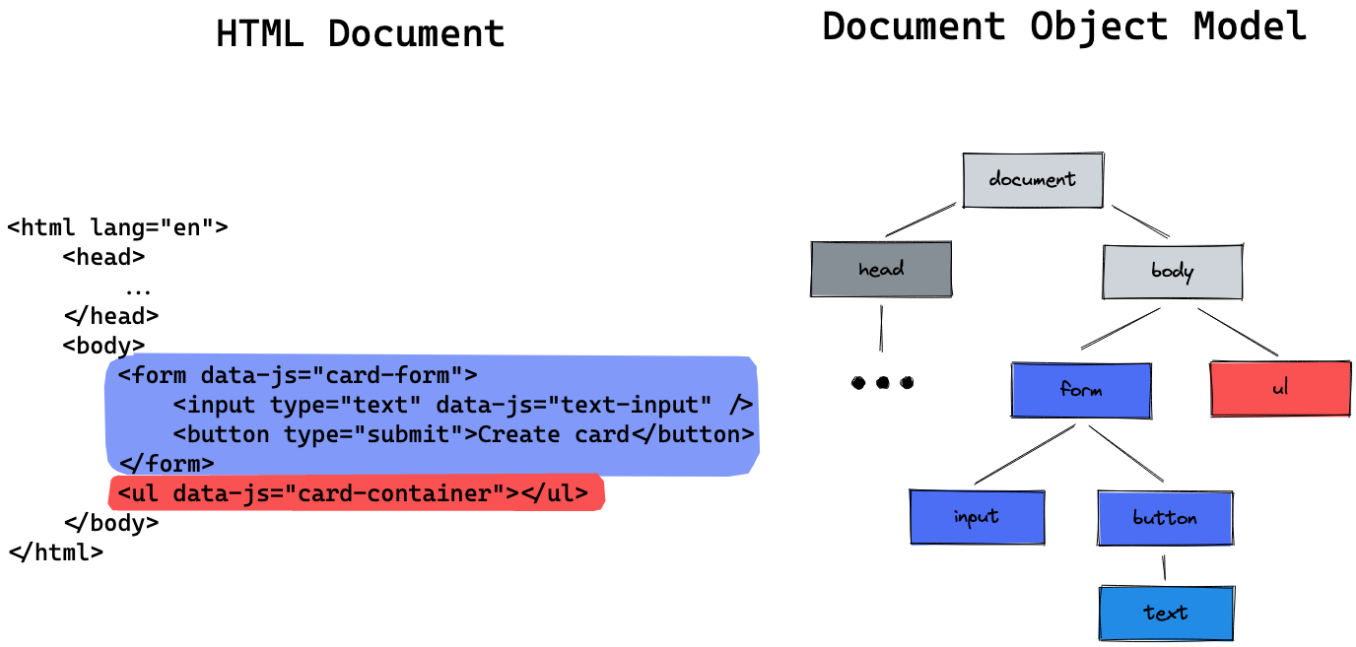
JS createElement

Lernziele

- Wissen, was das DOM ist
- Lernen, wie man HTML in JavaScript generiert
- Verwenden von HTML-Element-Objekteigenschaften und -methoden
- Lernen, wie man `.innerHTML` verwendet

Das DOM

Das **Document Object Model** ist eine Darstellung des HTML-Dokuments. Jeder HTML-Tag wird als **Knoten** in einer Baumstruktur modelliert, die zeigt, wie HTML-Elemente verschachtelt sind. Ein Computerprogramm wie Ihre JavaScript-Datei kann auf die HTML-Website zugreifen und sie manipulieren, indem es das DOM über das `document`-Objekt ändert.



`document.createElement`

Sie können ein HTML-Element mit JavaScript erstellen, indem Sie die `document.createElement`-Methode verwenden. Diese Methode erwartet den Typ des Elements als Argument.

```
const article = document.createElement("article");
const button = document.createElement("button");
```

Nachdem Sie ein Element erstellt haben, müssen Sie das Element in das DOM einfügen. Dafür können Sie die `.append`-Methode verwenden. Sie platziert das Element als **last child** in das jeweilige Element.

```
document.body.append(article); // das erstellte article-Element am Ende des body platzieren
article.append(button); // das erstellte button-Element in das article einfügen
```

Das Ergebnis sieht so aus:

```
<body>
  ...
  <article>
    <button></button>
  </article>
</body>
```

Eigenschaften und Methoden von Elementen

Wie bei abgefragten HTML-Elementen (über `querySelector`) können wir Klassen hinzufügen, Event-Listener und mehr zu den erstellten HTML-Elementen hinzufügen.

```
article.classList.add("card");

button.addEventListener("click", () => {
  console.log("Es funktioniert!");
});
```

Der Text eines Elements kann geändert werden, indem die `.textContent`-Eigenschaft neu zugewiesen wird:

```
button.textContent = "Klick mich!";
```

Häufig verwendete Eigenschaften und Methoden von Elementen

Property	Effect
`classList`	Klassen zum Element hinzufügen, toggeln oder entfernen
`textContent`	Text im Element abrufen oder setzen
`style`	Inline-Styles definieren, z. B. `element.style.backgroundColor = "red"`
`hidden`	Boolean, ob das Element ausgeblendet ist oder nicht
`focus()`	Fokussiert das Element auf der Website

Property	Effect
`hasAttribute()`	Gibt true zurück, wenn das Element das angegebene Attribut hat
`querySelector()`	Gibt das erste Kind zurück, das dem angegebenen CSS-Selektor entspricht

> 💡 Sie können HTML-Attribute durch die Verwendung der Eigenschaften des Elements zuweisen. Besuchen Sie die MDN Docs für eine umfassende Liste von Element-Eigenschaften.

`.innerHTML``

> ❗ innerHTML kann unsicher sein, wenn Benutzereingaben in das Template-Literal eingefügt werden. Verwenden Sie es mit Vorsicht. Lesen Sie diesen Artikel für weitere Informationen darüber.

Die `innerHTML`-Eigenschaft kann verwendet werden, um das gesamte HTML-Layout eines Elements zu erstellen, indem der HTML-Code als String übergeben wird. Durch die Verwendung von **Template-Literals** kann der Inhalt des HTMLs dynamisch erstellt werden.

```
const cityName = "Lissabon";

article.innerHTML = `
    <h2> ${cityName} </h2>
    <p class="card__content">
        ${cityName} ist eine sehr schöne Stadt in Portugal.
        Gehen Sie hin und genießen Sie den Aufenthalt!
    </p>
    <button type='button' class="card__booking-button">
        Reise buchen
    </button>
`;
```

Dieser HTML-Code wird dann **innerhalb** des article-Elements gerendert:

```
<body>
  ...
  <article>
    <h2>Lissabon</h2>
    <p class="card__content">
      Lissabon ist eine sehr schöne Stadt in Portugal. Gehen Sie hin und
      genießen Sie den Aufenthalt!
    </p>
    <button type="button" class="card__booking-button">Reise
buchen</button>
  </article>
</body>
```

Zurücksetzen des Inhalts eines Elements

`.innerHTML` kann auch verwendet werden, um den Inhalt eines Elements, z. B. eines Containers, **zurückzusetzen**:

HTML vorher:

```
<ul data-js="cardContainer">
  <li class="card">...</li>
  <li class="card">...</li>
  <li class="card">...</li>
</ul>
```

Durch das Setzen von innerHTML auf einen leeren String wird der Inhalt gelöscht:

```
const cardContainer = document.querySelector('[data-js="cardContainer"]');
cardContainer.innerHTML = "";
```

Das Ergebnis:

```
<ul data-js="cardContainer"></ul>
```

Ressourcen

Element Properties

MDN Docs about element Properties

innerHTML

MDN Docs about securtiy risks with innerHTML