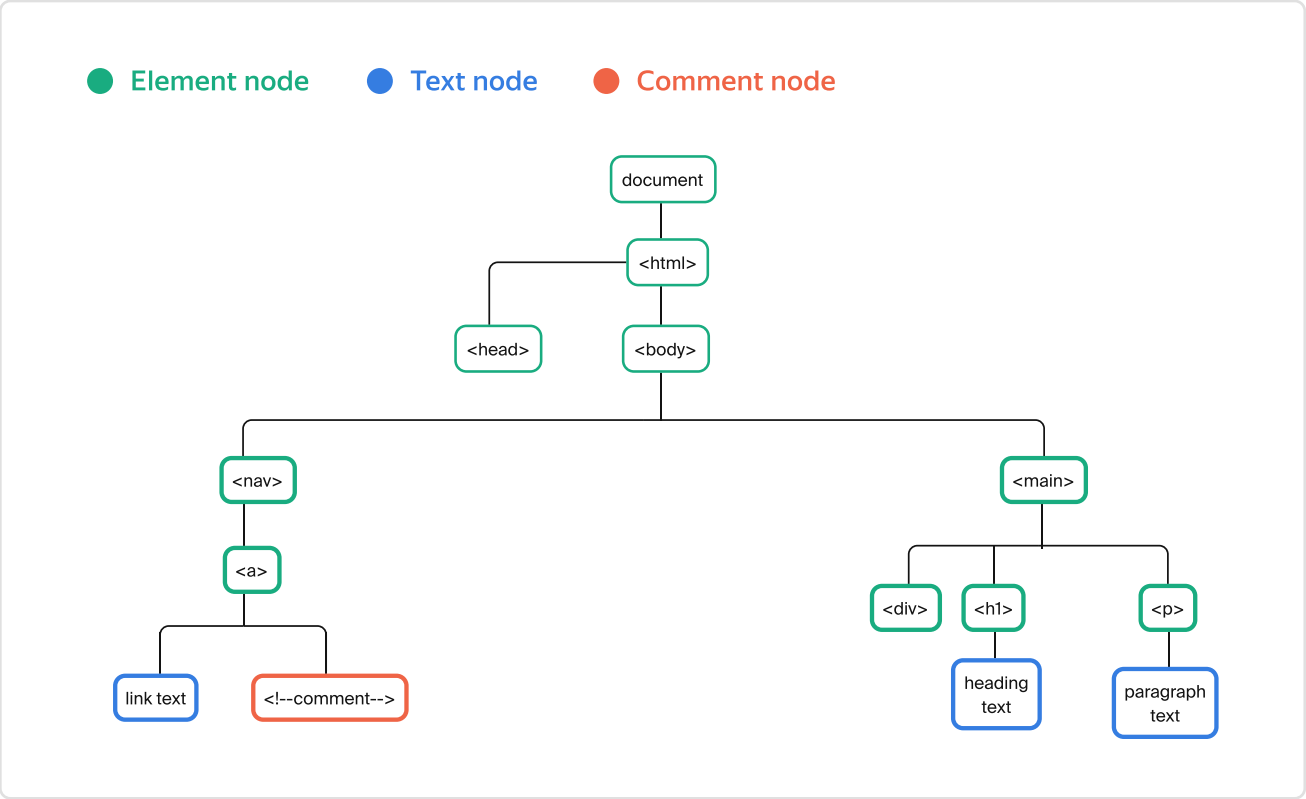
DOM Basics

Sprint 4

The Document Object Model (DOM)

An abstract representation of an HTML Document. Consists of three types of nodes: **elements**, **text**, and **comments**.



Selecting HTML Elements

`document.querySelector` selects the first element with a matching CSS selector string:

```
// calling it on the document returns the first matching element on the page
const element = document.querySelector(selectorString);

// calling it on another element returns the first matching element inside that element
const element = someOtherElement.querySelector(selectorString);
```

`document.querySelectorAll` selects all elements with a matching CSS selector string:

```
// select all matching elements on the page
const elements = document.querySelector(selectorString);
```

These methods can be used on any HTML element. For example:

```
// select first element with class `parent`
const parent = document.querySelector(".parent");

// select first element contained in `parent` that has the `child` class
const child = container.querySelector(".child");
```

In general, you should use `".class"` selectors with `querySelector` and `querySelectorAll`

Manipulating DOM Elements

Sprint 4

Manipulating an Element's Attributes

Use `element.setAttribute()` to set an element's attribute:

```
// sets the element's attribute to the supplied value
element.setAttribute(attribute, value);

// example - setting the `src` attribute of the first element with the class `image`
document.querySelector(".image").setAttribute("src", "path/to/image");

// example - setting the Boolean `disabled` attribute
document.querySelector(".button").setAttribute("disabled", "");

// example - setting inline CSS styles via the `style` attribute
document.querySelector(".button").setAttribute("style", "background-color: #000");
```

Use `element.removeAttribute()` to remove an element's attribute:

```
// example - removing an element's attribute to the supplied value
element.removeAttribute("disabled");
```

Many attributes can be set by directly changing the corresponding property of an element:

```
// example - setting an element's `disabled` attribute
element.disabled = true;

// example - removing an element's `disabled` attribute
element.disabled = false;

// example - setting an element's inline styles, make sure to use camelCase instead of kebab-case
element.style.backgroundImage = "url(path/to/image)";
```

Manipulating an Element's Classes via the `classList` Property

```
// access an element's list of classes
element.classList;

// add a class to an element's class list
element.classList.add("some-class");
```

```
// remove a class from an element's class list
element.classList.remove("some-class");

// remove a class if it exists, add it if it doesn't
element.classList.toggle("new-class");
```

Replacing an Element's Content

Entirely replaces text content with `element.textContent`:

```
element.textContent = "replacement text content";
```

Entirely replaces text content with `element.innerHTML`:

```
element.innerHTML = "<span>replacement HTML content</span>";
```

Appending to an Element's Content

`element.insertAdjacentText()` **inserts new text content:**

```
element.insertAdjacentText(where, "insert this text");
```

`element.insertAdjacentHTML()` **inserts new HTML content:**

```
element.insertAdjacentHTML(where, "<span>insert this content</span>");
```

The first parameter of these functions describes where the new content should be inserted:

- `"beforebegin"` — before the element itself
- `"afterbegin"` — before the element's first child
- `"beforeend"` — after the element's last child
- `"afterend"` — after the element itself

Events and the Event Object

Sprint 4

Events

Things that happen on a webpage.

Common event types include:

- `"click"` — triggered when a user clicks on an element
- `"mouseover"` — triggered when the cursor hovers over an element
- `"mouseout"` — triggered when the cursor hover ends and it moves away from an element
- `"scroll"` — triggered when the user scrolls an element
- `"submit"` — triggered when the user clicks a submit `<button>` element or presses Enter while editing an input field in a form

Listen for events and react to them using element.addEventListener():

```
// example - removing an element's "disabled" attribute
element.removeAttribute("disabled");
```

Listen for events and react to them using element.addEventListener():

```
// example - logs a string to the console when a button is clicked
document.querySelector(".button").addEventListener("click", function() {
  console.log("Button has been clicked");
});
```

The `event` Object

Contains useful info about the event and the element that triggered it.

You can access the event object as the first argument of `addEventListener()`:

```
// by convention, we call it `event`, `evt`, or `e`
element.addEventListener("click", function (event) {
  console.log(event); // logs the event object to the console
});
```

Properties and methods include:

- The `event.target` property stores the element on which the event occurred
- `event.preventDefault()` prevents the default browser behavior from occurring when the event is triggered
  - Often used to prevent page reloads on submit events

Template Elements

Sprint 4

`<template>`

Elements that contain reusable blocks of HTML markup:

```
<template id="user">    <!-- use an id to select the template -->
  <div class="user">
    <img class="user__avatar" alt="avatar">   <!-- the src attribute will be set using JavaScript -->
    <p class="user__name"></p>   <!-- the textContent will be set using JavaScript -->
  </div>
</template>
```

Note: A `<template>` and it's content will not be rendered on the page.

Rendering `<template>` elements with JavaScript

```
// parent is the container where the users will be rendered
const userList = document.querySelector(".users");

// select the template
const userTemplate = document.querySelector("#user")
                             .content              // access the content of the template tags
                             .cloneNode(true);     // clone the node and all of its contents

// insert content as needed
userElement.querySelector(".user__avatar").src = "tinyurl.com/v4pfzwy";
userElement.querySelector(".user__name").textContent = "Duke, mayor of Cormorant";

// make it appear on the page
userList.append(userElement);
```

Adding Elements to the DOM

Each of these functions accepts a comma-separated list of parameters.

The parameters can be DOM nodes or strings:

- `node.append(firstParam, secondParam, ...)` adds the parameters after the last child of the `node`
- `node.prepend(firstParam, secondParam, ...)` adds the parameters before the first child of the `node`
- `node.after(firstParam, secondParam, ...)` inserts the parameters after the `node`
- `node.before(firstParam, secondParam, ...)` inserts the parameters before the `node`
- `node.replaceWith(firstParam, secondParam, ...)` replaces `node` with the parameters