the shortcut

Fullstack Web Development Tutorial Lesson 10

Today’s lesson will cover

- **DOM**

the shortcut

JavaScript fundamentals

DOM and BOM

- Depending on host environment, you get specific objects and additional functions on top of language core
- Browser provides a “root” object called `window`
- Document Object Model, or DOM for short, represents all page content as objects that can be modified
- `document` object is the main “entry point” to the page. We can change or create anything on the page using it
- DOM Properties and methods are described in the specification:
    - DOM Living Standard at https://dom.spec.whatwg.org

- **Browser Object Model (BOM)** represents additional objects provided by the browser (host environment) for working with everything except the document.
    - Functions `alert/confirm/prompt` are also a part of BOM: they are directly not related to the document, but represent pure browser methods of communicating with the user.
    - BOM is the part of the general HTML specification

DOM Nodes

- As per DOM, every HTML tag is an object
- All these objects and contents are accessible using Javascript and can be modified
- Everything in your HTML body including comments becoming nodes that can be accessible
    - An anomaly which is part of autocorrection by DOM: <table> adds <tbody> node which may not be present in original HTML
- There are 12 node types. In practice we usually work with 4 of them:
    - `document` – the "entry point" into DOM.
    - element nodes – HTML-tags, the tree building blocks.
    - text nodes – contain text.
    - comments – sometimes we can put information there, it won't be shown, but JS can read it from the DOM.
- **Child nodes (or children)** – elements that are direct children. In other words, they are nested exactly in the given one. For instance, `<head>` and `<body>` are children of `<html>` element.
- **Descendants** – all elements that are nested in the given one, including children, their children and so on.
- Properties `firstChild` and `lastChild` give fast access to the first and last children
- There's also a special function `elem.hasChildNodes()` to check whether there are any child nodes

DOM Nodes *(Contd.)*

- `childNodes` looks like an array. But actually it’s not an array, but rather a *collection* – a special array-like iterable object. Array methods won’t work because nodes aren’t arrays, but we can create arrays and then use those methods if needed
- Use `for..of` to iterate over it not `for..in`
- Siblings and the parent: Siblings are nodes that are children of the same parent.
    - `<body>` is said to be the “next” or “right” sibling of `<head>`,
    - `<head>` is said to be the “previous” or “left” sibling of `<body>`.
    - The next sibling is in `nextSibling` property, and the previous one – in `previousSibling`.
    - The parent is available as `parentNode`.
- For many tasks we don’t want text or comment nodes. We want to manipulate element nodes that represent tags and form the structure of the page in such links are similar to those given above, just with `Element` word inside:
    - `children` – only those children that are element nodes.
    - `firstElementChild`, `lastElementChild` – first and last element children.
    - `previousElementSibling`, `nextElementSibling` – neighbor elements.
    - `parentElement` – parent element.

Exercise

1. Log the first names from the following HTML doc on the console
2. Log name of all the nodes inside the HTML body in lowercase on the console

```
<html>
<body>
  <ul> First Names
    <li>John</li>
    <li>Pete</li>
    <li>Anne</li>
    <li>Mary</li>
  </ul>
  <ul> Last Names
    <li>Walker</li>
    <li>Smith</li>
    <li>Badan</li>
    <li>Anne</li>
  </ul>
</body>
</html>
```

```
// Output 1
First Names
John
Pete
Anne
Mary

// Output 2
#text
ul
#text
ul
#text
script
```

Targeting arbitrary elements

- If element has the `id` attribute, we can get the element using the method `document.getElementById(id)`, no matter where it is
- `querySelectorAll()` and `querySelector()` method as returning a NodeList representing a list of elements matching the specified group of selectors which are descendants of the object on which the method was called
  - `Element.querySelectorAll()`, `Document.querySelectorAll()`, and `DocumentFragment.querySelectorAll()`
- Refer to all possible CSS selectors whenever targeting certain elements
  - https://developer.mozilla.org/en-US/docs/Web/CSS/CSS_Selectors

Modifying contents

- **innerHTML**: The innerHTML property allows to get the HTML inside the element as a string. We can also modify it. So it's one of the most powerful ways to change the page. Does full overwrite.
- Outer html: Replaces full HTML, doesn't change element content
- **nodeValue and data properties**: innerHTML valid for elements but for other node types, need to use nodeValue or data properties; both of which functions similarly with minor differences
- **"hidden" attribute:** specifies whether the element is visible or not. `hidden` works the same as `style="display:none"`. But it's shorter to write.
- You can access any attributes such as `href,` `type` `or` `value` or any modify them

Exercise

- Write script to make the following changes
- Change the body style so it has a font-family of "Arial, sans-serif".
- Replace each of the spans (nickname, favorites, hometown) with your own information.
- Iterate through each li and change the class to "listitem". Add a `style` tag that sets a rule for "listitem" to make the color red - Do this manually by editing original HTML.
- Use `setinterval()` to and to toggle `hidden` attribute for My Page in the bottom every 1 second

```
<!DOCTYPE html>
<html>
 <head>
  <meta charset="utf-8"/>
  <title>About Me</title>
</head>
<body>
  <h1>About Me</h1>

  <ul>
    <li>Nickname: <span id="nickname"></span>
    <li>Favorites:  <span id="favorites"></span>
    <li>Hometown: <span id="hometown"></span>
   </ul>
   <aside>
      <h1>My page</h1>
   </aside>
 </body>
</html>
```

Summary: DOM node methods

There are 6 main methods to search for nodes in DOM:

Method	Searches by...	Can call on an element?
`querySelector`	CSS-selector	✓
`querySelectorAll`	CSS-selector	✓
`getElementById`	`id`	-
`getElementsByName`	`name`	-
`getElementsByTagName`	tag or `'*'`	✓
`getElementsByClassName`	class	✓

By far the most used are `querySelector` and `querySelectorAll`, but `getElementBy*` can be sporadically helpful or found in the old scripts.

Besides that:

- There is `elem.matches(css)` to check if `elem` matches the given CSS selector
- There is `elem.closest(css)` to look for the nearest ancestor that matches the given CSS-selector. The `elem` itself is also checked.

Summary: Main DOM node properties

- **`nodeType:`** We can use it to see if a node is a text or an element node. It has a numeric value: `1` for elements,`3` for text nodes, and a few others for other node types. Read-only.
- **`nodeName/tagName:`** For elements, tag name (uppercased unless XML-mode). For non-element nodes `nodeName` describes what it is. Read-only.
- **`innerHTML:`** The HTML content of the element. Can be modified.
- **`outerHTML:`** The full HTML of the element. A write operation into `elem.outerHTML` does not touch `elem` itself. Instead it gets replaced with the new HTML in the outer context.
- **`nodeValue/data:`** The content of a non-element node (text, comment). These two are almost the same, usually we use `data`. Can be modified.
- **`textContent:`** The text inside the element: HTML minus all `<tags>`. Writing into it puts the text inside the element, with all special characters and tags treated exactly as text. Can safely insert user-generated text and protect from unwanted HTML insertions.
- **`Hidden:`** When set to `true`, does the same as CSS `display:none`.
- DOM nodes also have other properties depending on their class. For instance, `<input>` elements (`HTMLInputElement`) support `value`, `type`, while `<a>` elements (`HTMLAnchorElement`) support `href` etc. Most standard HTML attributes have a corresponding DOM property.

the shortcut

Self Study Assignments

To Dos

- Continue freecodecamp Javascript. Ideally finish before we resume after summer.
- Continue with FCC HTML, CSS lessons. Ideally finish all the lessons by end of this month.
- If you need help pushing your HTML CSS project on GIthub and using Github pages let me know right away.