the shortcut

Fullstack Web Development Tutorial Lesson 11

Today’s lesson will cover

- DOM Properties
- HTML Attributes
- Styles

the shortcut

JavaScript fundamentals

DOM properties and HTML attributes

- When the browser reads or parses the HTML, it generates DOM objects from it
- DOM nodes are regular JavaScript objects. We can alter them.
- We can create new property and methods just like any other Javascript objects
- HTML, tags may have attributes. When the browser parses the HTML to create DOM objects for tags, it recognizes *standard* attributes and creates DOM properties from them.
    - Standard attribute for one element can be unknown for another one. For instance, `"type"` is standard for `<input>` (HTMLInputElement), but not for `<body>` (HTMLBodyElement).
- Standard attributes can be accessed just like any property but in general all attributes are accessible by using the following methods:
    - `elem.hasAttribute(name)` – checks for existence.
    - `elem.getAttribute(name)` – gets the value.
    - `elem.setAttribute(name, value)` – sets the value.
    - `elem.removeAttribute(name)` – removes the attribute.
- You can read all attributes using `elem.attributes`

Non-standard attributes, dataset

- Non-standard attributes are used to pass custom data from HTML to JavaScript, or to “mark” HTML-elements for JavaScript
- Also they can be used to style an element because an attribute is more convenient to manage. The state can be changed easily
- Using non-standard attributes however can raise possibility of conflicts in case standard attributes with same name is introduced in HTML
    - Hence, **All attributes starting with “data-” are reserved for programmers’ use. They are available in the `dataset` property.**

Modifying DOM: Creating and inserting DOM nodes

- DOM modification is the key to creating “live” pages
- To create DOM nodes meaning for creating HTML elements, there are two methods:
  - **`document.createElement(tag)`**: Creates a new *element node* with the given tag
  - **`document.createTextNode(text)`**: Creates a new *text node* with the given text
- After creating DOM nodes, they need to inserted, using a special method `append`
- This set of methods provides many ways to insert DOM nodes or text pieces:
  - `node.append(...nodes or strings)` – append nodes or strings at the end of `node`,
  - `node.prepend(...nodes or strings)` – insert nodes or strings at the beginning of `node`,
  - `node.before(...nodes or strings)` –- insert nodes or strings before `node`,
  - `node.after(...nodes or strings)` –- insert nodes or strings after `node`,
  - `node.replaceWith(...nodes or strings)` –- replaces `node` with the given nodes or strings.

Modifying DOM: Inserting HTML

- To insert HTML “as html”, with all tags and stuff working, like `elem.innerHTML`?
- To insert full HTML, we use the method: `elem.insertAdjacentHTML(where, html)`.
  - The first parameter is a code word, specifying where to insert relative to `elem`. Must be one of the following:
    - `"beforebegin"` – insert `html` immediately before `elem`,
    - `"afterbegin"` – insert `html` into `elem`, at the beginning,
    - `"beforeend"` – insert `html` into `elem`, at the end,
    - `"afterend"` – insert `html` immediately after `elem`.
  - The second parameter is an HTML string, that is inserted “as HTML”.
  - The method has two brothers:
    - `elem.insertAdjacentText(where, text)` – the same syntax, but a string of `text` is inserted “as text” instead of HTML,
    - `elem.insertAdjacentElement(where, elem)` – the same syntax, but inserts an element.

Node modification

- `node.remove()`
  - To remove a node, there's a method `node.remove()`.
  - **All insertion methods automatically remove the node from the old place**.
- Cloning nodes: cloneNode
  - The call elem.cloneNode(true) creates a "deep" clone of the element – with all attributes and subelements. If we call elem.cloneNode(false), then the clone is made without child elements.
- DocumentFragment
  - DocumentFragment is a special DOM node that serves as a wrapper to pass around lists of nodes.
  - We can append other nodes to it, but when we insert it somewhere, then its content is inserted instead.

Exercise

Keep track of which books you read and which books you want to read!

- Create a webpage with an `h1` of "My Book List".
- Add a script tag to the bottom of the page, where all your JS will go.
- Copy the array of books from:

```
var books = [
  {title: 'The Design of EveryDay Things',
   img: 'http://ecx.images-amazon.com/images/I/41j2ODGkJDL._AA115_.jpg',
   author: 'Don Norman',
   alreadyRead: false
  },
  {title: 'The Most Human Human',
   img: 'http://ecx.images-amazon.com/images/I/41Z56GwEv9L._AA115_.jpg',
   author: 'Brian Christian',
   alreadyRead: true

  }];
```

- Iterate through the array of books. For each book, create a `p` element with the book title and author and append it to the page.
- **Bonus**: Use a `ul` and `li` to display the books.
- **Bonus**: add a property to each book with the URL of the book cover, and add an `img` element for each book on the page.
- **Bonus**: Change the style of the book depending on whether you have read it or not

Styles: className and classList

- Changing a class is one of the most often used actions in scripts.
- We can operate both on the full class string using `className` or on individual classes using `classList`s.
    - `elem.className`, it replaces the whole string of classes
    - `elem.classList` is a special object with methods to `add/remove/toggle` a single class
- Methods of `classList`:
    - `elem.classList.add/remove("class")` – adds/removes the class.
    - `elem.classList.toggle("class")` – adds the class if it doesn't exist, otherwise removes it.
    - `elem.classList.contains("class")` – checks for the given class, returns `true/false`.
    - `classList` is iterable, so we can list all classes with `for..of`

Styles: Element style

- `elem.style` is an object that corresponds to what's written in the `"style"` attribute.
- Sometimes we want to assign a style property, and later remove it.
  - In such cases instead of for instance `delete elem.style.display` we should assign an empty string to it: `elem.style.display = ""`.
- Mind the CSS units to values
  - For instance, we should not set `elem.style.top` to `10`, but rather to `10px`. Otherwise it wouldn't work
- For multiword property, you have to use camelCase
  - `background-color` => `elem.style.backgroundColor`
  - `border-left-width` => `elem.style.borderLeftWidth`

Exercise

- Grab the files from Github: https://github.com/itistheshortcut/fullstack-june-2020/tree/master/lesson%2011
- Write a function `showNotification(options)` that creates a notification: `<div class="notification">` with the given content. The notification should automatically disappear after 1.5 seconds.
- The options are:

```
// shows an element with the text "Hello" near the right-top of the window
showNotification({
  top: 10, // 10px from the top of the window (by default 0px)
  right: 10, // 10px from the right edge of the window (by default 0px)
  html: "Hello!", // the HTML of notification
  className: "welcome" // an additional class for the div (optional)
});
```

Summary: DOM Properties and HTML Attributes

- Attributes – is what's written in HTML.
- Properties – is what's in DOM objects
- Methods to work with attributes are:
  - `elem.hasAttribute(name)` – to check for existence.
  - `elem.getAttribute(name)` – to get the value.
  - `elem.setAttribute(name, value)` – to set the value.
  - `elem.removeAttribute(name)` – to remove the attribute.
  - `elem.attributes` is a collection of all attributes.
- For most situations using DOM properties is preferable. We should refer to attributes only when DOM properties do not suit us, when we need exactly attributes, for instance:
- We need a non-standard attribute. But if it starts with `data-`, then we should use `dataset`.
- We want to read the value "as written" in HTML. The value of the DOM property may be different, for instance the `href` property is always a full URL, and we may want to get the "original" value.

Summary: Modifying the document

- Methods to create new nodes:
  - `document.createElement(tag)` – creates an element with the given tag,
  - `document.createTextNode(value)` – creates a text node (rarely used),
  - `elem.cloneNode(deep)` – clones the element, if `deep==true` then with all descendants.
- Insertion and removal:
  - `node.append(...nodes or strings)` – insert into `node`, at the end,
  - `node.prepend(...nodes or strings)` – insert into `node`, at the beginning,
  - `node.before(...nodes or strings)` –- insert right before `node`,
  - `node.after(...nodes or strings)` –- insert right after `node`,
  - `node.replaceWith(...nodes or strings)` –- replace `node`.
  - `node.remove()` –- remove the `node`.
- All these methods return `node`. Given some HTML in `html`, `elem.insertAdjacentHTML(where, html)` inserts it depending on the value of `where`:
  - `"beforebegin"` – insert `html` right before `elem`,
  - `"afterbegin"` – insert `html` into `elem`, at the beginning,
  - `"beforeend"` – insert `html` into `elem`, at the end,
  - `"afterend"` – insert `html` right after `elem`.

Summary: Styles

- To manage classes, there are two DOM properties:
  - `className` – the string value, good to manage the whole set of classes.
  - `classList` – the object with methods `add/remove/toggle/contains`, good for individual classes
- To change the styles:
  - The `style` property is an object with camelCased styles. Reading and writing to it has the same meaning as modifying individual properties in the `"style"` attribute. To see how to apply `important` and other rare stuff – there's a list of methods at MDN.
  - The `style.cssText` property corresponds to the whole `"style"` attribute, the full string of styles.

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.