JS.2

▼ How to manipulate generic attributes on elements?

The element.getAttribute(key) method is used to get the value a certain attribute by its key.

The element.removeAttribute(key) is used to remove an attribute.

The element.setAttribute(key, value) is used to write a new attribute (or update the value of an old one that already exists).

The element.hasAttribute(key) method is used to check whether an attribute exists or not.

```
<div id="banner">
    <button disabled="disabled" id="login">Login</button>
</div>

const banner = document.querySelector("#banner");
banner.getAttribute("id"); // "banner"

const button = document.querySelector("#login");
button.removeAttribute("disabled");

banner.setAttribute("id", "navbar");

button.hasAttribute("disabled"); // false
```

▼ Should you update classes over styles?

Yes

▼ Why use .style property?

Sometimes you need to compute a dynamic value based on some variables

```
const banner = document.querySelector("#banner");
banner.style.backgroundColor = "red";

// hide element
banner.style.display = "none";

// show element by resetting it's display
banner.style.display = ""; //or "initial"
```

▼ How to remove an element from the DOM?

```
<h1 id="headline">Welcome</h1>

const headline = document.querySelector("#headline");
headline.remove();
```

▼ What does document.documentElement return?

the Element that is the root element of the document (for example, the <html> element for HTML documents).

```
const element = document.documentElement
```

▼ How to access the <body> element of the page?

```
document.body // the <body> of the page
```

▼ What is a data attribute?

the HTML spec recommends that developers prefix their own custom attributes with data-.

```
<-- This is recommended -->
<form id="payment-form" data-currency="EUR">
    ...
</form>
```

▼ What is a dataset object?

To read a data attribute, you can access the dataset object on an element.

```
const form = document.querySelector("#payment-form");
console.log(form.dataset); // {currency: "EUR"}
const currency = form.dataset.currency; // "EUR"

console.log(form.dataset)
/*
{
    userId: "2",
    currency: "EUR"
}
*/
```

▼ How to write data attributes on a dataset object?

```
const navbar = document.querySelector("#navbar");
navbar.dataset.userId = 43;
navbar.dataset.rememberMe = false;

<div id="navbar" data-user-id="43" data-remember-me="false"></div>
```

▼ How to get the parent element of the current element?

The element.parentElement property returns the parent element of the current element.

```
<div class="article">
    <h1>Hello World</h1>
    <p>Lorem ipsum</p>
</div>

const h1 = document.querySelector("h1");
console.log(h1.parentElement); // <div class="article">...</div>
```

▼ How to get the **closest** parent element of the current element?

The element.closest("CSS-selector") method returns the closest parent that matches the CSS-selector you specified. It searches for parent elements and goes up one by one.

```
<div class="main">
    <div class="banner">
        <h1>Hello World</h1>
    </div>
</div>

const h1 = document.querySelector("h1");
```

```
h1.closest(".main");
console.log(h1); // <div class="main">...</div>
```

▼ How to append HTML?

element.insertAdjacentHTML("beforeend", htmlString)

```
<div id="job-positions">
    <div class="position">2009-2013</div>
    <div class="position">2013-2015</div>
</div>

const positions = document.querySelector("#job-positions");
positions.insertAdjacentHTML("beforeend", `<div class="position">2015-2020</div>`);

// after
<div id="job-positions">
    <div class="position">2009-2013</div>
    <div class="position">2013-2015</div>
    <div class="position">2015-2020</div>
</div>
```

▼ How to prepend HTML?

element.insertAdjacentHTML("afterbegin", htmlString)

```
<div id="job-positions">
    <div class="position">2009-2013</div>
    <div class="position">2013-2015</div>
</div>

const positions = document.querySelector("#job-positions");
positions.insertAdjacentHTML("afterbegin", `<div class="position">2007-2009</div>`);

// after
<div id="job-positions">
    <div class="position">2007-2009</div>
    <div class="position">2009-2013</div>
    <div class="position">2013-2015</div>
</div>
```

▼ How to insert an array of items into the DOM?

The insertAdjacentHTML method presents the same security risk as innerHTML.

So, you should not use it if the variables you're interpolating might be coming from the user.

```
<ul id="apps-list"></ul>

const apps = ["Calculator", "Phone", "Messages"];
const list = document.querySelctor("#apps-list");

apps.forEach(app => {
    list.insertAdjacentHTML("beforeend", `<li>${app}</li>`);
});

// after
<ul id="apps-list">
    <li>Calculator</li>
    <li>Phone</li>
    <li>Messages</li>
</ul>
```

▼ What does the innerHTML method do?

write HTML and overwrite all the previous values

▼ What does the element.addEventListener(eventType, callback) method do?

allows you to wait for an event (let's say click for now) to happen on an element

Once that event occurs (the user clicks on the button), the callback function will execute.

▼ How to change a button to 'Loading' and disable it?

```
const button = document.querySelector("#app-button");

button.addEventListener("click", () => {
    button.setAttribute("disabled", "disabled");
    button.textContent = "Loading...";
});
```

▼ Web accessible click event?

make your website accessible for users with screen readers, you should only add a click event listener on <button> and <a> elements

An easier solution would be to wrap any item with a <button> element and then adjust the styles of that button to not make it look like a button.

```
<button class="clickable">
    <img src="..." alt="...">
</button>

.clickable {
    all: unset;
}
```

▼ Within an event listener, what argument does the callback receive from the browser?

event details

You can add the event parameter to be able to access the event details:

```
button.addEventListener("click", event => {
    // event callback
});
```

▼ What is event.currentTarget?

The event.currentTarget refers to the element to which the event listener has been attached.

```
button.addEventListener("click", event => {
    console.log(event.currentTarget); // same as the variable 'button'
    console.log(event.currentTarget.textContent); // text of button
});
```

▼ Why use debugger in an event listener?

you can add a debugger; statement which will pause the execution of your code and thus allowing you to see the event.currentTarget in the dev tools console:

```
button.addEventListener("click", event => {
    console.log(event);
    debugger;
});
```

▼ How to add an event listener on multiple elements?

```
const buttons = document.querySelectorAll("button");

const handleButtonClick = event => {
    console.log(event.currentTarget); // the 'button' that was clicked.
}

buttons.forEach(button => {
    button.addEventListener("click", handleButtonClick);
});
```

▼ How to trigger a submit event for a form?

1. have a form element
2. have at least 1 input or textarea inside of the form
3. have a button with type="submit"

The button could either be an <input type="submit" value="..."> or a <button type="submit">...</button>.

```
<form id="address-form">
    <input type="text" placeholder="Enter your address">
    <button type="submit">Submit Form</button>

    // or you could do an input for submit event
    // <input type="submit" value="Save">
</form>
```

▼ How to listen for the submit event on a form?

```
const form = document.querySelector("#address-form");

form.addEventListener("submit", event => {
    // event callback (when the form is submitted)
});
```

▼ When you submit a form, the browser will take all the values your user has written and...

send them to the backend of your website

▼ Why do we use event.preventDefault() for forms?

We don't want to reload the page every time the user sends some information.

We have to prevent the default behavior of the submit event by calling the preventDefault method on the event details.

```
const form = document.querySelector("#address-form");

form.addEventListener("submit", event => {
    event.preventDefault();
    // the form will not reload anymore on submit
});
```

▼ How to read the user's input with inputElement.value?

you have to make sure that you access value inside the submit event. Otherwise, the value will return an empty string ("")

```
const form = document.querySelector("#weather-form");
const city = document.querySelector("#city");

form.addEventListener("submit", event => {
    event.preventDefault();
```

```
    // read the user's city once has been filled out on submit
    console.log(city.value); // see in the console to make sure it's working
});
```

▼ How to remove an existing event listener on an element?

element.removeEventListener(eventType, callback)

The callback has to be the same one as added originally and should be named:

```
button.removeEventListener("click", handleClick);
```

▼ How to add an event listener that only runs once?

```
const button = document.querySelector("button");

button.addEventListener("click", () => {
    console.log("button clicked");
}, {
    once: true
});
```

▼ How to get a click event to work on mobile?

The click event also works on mobile (as long as you have the <meta name="viewport"> defined with a content such as "width=device-width,initial-scale=1").

▼ What is a focus event?

when a user focuses (put the cursor inside of it) on a textbox

The word focus means that the element is selected to receive user input from the keyboard. If you write something on your keyboard, it will be written inside the element that is focused.

```
name.addEventListener("focus", () => {
    console.log("user focused inside the name");
});
```

▼ What is a blur event?

When you remove the focus, then this will dispatch a blur event.

```
name.addEventListener("blur", () => {
    console.log("user removed focus from the name");
});
```

▼ What is a DOMContentLoaded event?

This event fires on the document element only. It signifies that the HTML has been loaded successfully by the browser.

This means that the browser has finished reading all of the content of your HTML file. It doesn't mean however that images and other assets have finished loading.

```
document.addEventListener("DOMContentLoaded", () => {
    console.log("DOM is ready");
});
```

▼ What is a scroll event?

The scroll event triggers on any element that scrolls.

However, adding a scroll event will most likely slow down your page. Its usage is discouraged as it makes scrolling slow, especially for scroll-based animations.

```
window.addEventListener("scroll", () => {
    console.log("page scrolled");
});
```

▼ What is a change event?

The change event is often used on the select element.

It lets you know when the user has selected a new choice.

```
<select id="countries">
    <option value="">Select a country</option>
    <option value="NL">Netherlands</option>
    <option value="BR">Brazil</option>
</select>

const countries = document.querySelector("#countries");

countries.addEventListener("change", () => {
    console.log(countries.value);
});
```

▼ What are keyup/keydown events?

keydown triggers while the user starts pressing the button and before the character is being typed

keyup fires after the character has been typed

```
document.addEventListener("keydown", event => {
    console.log(event.key);
});

document.addEventListener("keyup", event => {
    console.log(event.key);
});
```

▼ What is a fab button?

floating action button. think a plus (+) button on a page to open a modal

▼ How to capitalize a word?

word.charAt(0).toUpperCase() + word.slice(1)

▼ Every HTML element in your page implements what?

a certain interface which in turn inherits from a base interface called HTMLElement

```
const div = document.querySelector("div");
const p = document.querySelector("p");

div.toString(); // [object HTMLDivElement]
// means div gets access to all the properties and methods on HTMLDivElement

p.toString(); // [object HTMLParagraphElement]
// means p gets access to all the properties and methods on HTMLParagraphElement
```

▼ What is an interface?

An interface is very similar to the concept of class, except that you can only inherit it

(you're not allowed to create a new instance of it directly).

▼ How to check the type of an HTML element?

by converting it to a string

element.toString()

▼ What are some properties and methods you can find on a button element that implements the HTMLButtonElement interface?

Properties

- `accessKey`
- `autofocus`
- `disabled`
- `form`
- `formAction`

Methods

- `checkValidity()`
- `reportValidity()`
- `setCustomValidity()`

▼ Every time you create a new function, what new value will be defined inside of it?

this

▼ What solution should you use instead of var that = this or .bind(this)?

Use an arrow function. `this` inside function same as outside function

▼ What is lexical scope?

Arrow functions have lexical scope which means the `this` inside of them, is the same as the outer function.

▼ How can you pass a reference to a function?

```
const registerUser = (user, callback) => {
    if (!user.id) {
        return false;
    }

    console.log("registering user");
    //call the "callback" function passed as an argument
    callback();
}
```

▼ How to throw an exception manually?

```
const sayHello = name => {
    if (!name) {
        throw new Error("name must be provided");
    }
    console.log(`Hello ${name}`);
}
```

▼ The async/await keywords are syntactic sugar on top of what?

promises

▼ What can an async keyword do?

will make a function return a promise automatically

```
const getNumber = () => {
    return new Promise(resolve => {
        resolve(42);
    });
}

// the same as a promise
const getNumber = async () => {
    return 42;
}

// or can define like so
async function getNumber() {
    return 42;
}

// usage
getNumber().then(value => {
    console.log(value); // 42
});
```

▼ When a function is async it will always return what?

a promise

▼ How do you read a promise value?

you resolve the promise

▼ What does the await keyword do?

it "pauses" the execution of a function, until the promise resolves

will automatically call .then() on the promise and give you its result

```
async function getValue() {
    return 42;
}

// we made this function async
const init = async () => {
    // this will pause execution of init until promise returned from
    // getValue() resolves or rejects
    const result = await getValue();
    console.log(result); // 42
}

init();
```

▼ Why does the function that uses await need to be async as well?

- Whenever you use await, it means that you're working with a promise that needs to be resolved somewhere in the future.
- This means that the current function cannot return a value immediately and has to return a promise itself.

▼ How to async/await with the fetch() web API?

```
const getNotificationsCount = async () => {
    const response = await fetch("https://jsdemo-3f387-default-rtdb.europe-west1.firebasedatabase.app/notifications/new.json");
    // await for this promise then(response) => response.json to resolve
    const data = await response.json();
    console.log(data); // visualize response
    return data.count;
}

// Sample usage
getNotificationsCount().then(data => {
    console.log(data); // 3
});
```

▼ Why use a try/catch when using await?

If the promise being awaited moves from pending to rejected, the error message will be thrown as an error (using the throw operator).

You can recover from such an error by wrapping the await call with a try/catch.

```
const getUnreadCount = async () => {
    try {
        const response = await fetch("https://jsdemo-3f387-default-rtdb.europe-west1.firebasedatabase.app/notifications/new.json");
        const data = await response.json();
        return data.count;
    } catch (error) {
        // this throws an error (thus, works as expected)
        throw "An error has occurred";
        // or console.error(error);
    }
}
```

▼
▼
▼
▼
▼
▼
▼
▼
▼
▼
▼
▼
▼