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);
    }
}
```

▼ What is a package manager?

a tool that helps us find, install & update packages

▼ What is a package?

a JavaScript project created by another person, group of people, company, or organization

▼ What is Node (or NodeJs)?

a JavaScript runtime environment

The JavaScript that you write in Node is more or less the same as the JavaScript you learned here, except that there's no DOM (because it's running outside of the browser).

This means, that you do not have variables such as document and window.

▼ What is npm or Node Package Manager?

the package manager that comes bundled with Node

▼ What is a runtime environment?

an environment in which a program or application is executed

▼ When you install a package locally, where is it downloaded?

into a node_modules folder

▼ What is npx?

a command that comes with npm that lets you **execute** packages installed locally

e.g. npx webpack-dev-server

▼ Why is it recommended to install packages locally over globally (generally)?

you risk the problem of breaking certain projects

▼ When you install a package locally, its name and version will be saved in a file called what?

package.json

▼ What does this version mean? "webpack": "~2.6.1"

It accepts 2.6.0 - 2.6.9

▼ How are package releases formatted?

major.minor.patch

▼ What does this version mean? ^2.6.1

it accepts new patch releases and new minor releases but not a major release

it won't accept 3.0.0 or newer

▼ What does npm init do?

it will generate a package.json

▼ What questions do you need to answer when you create a package.json?

- What is the entry point? This will point to a javascript file that will be run when someone uses your package.
- What is the test command? This will be the command that runs all of your **tests**. Tests are used to make sure that your project's functionality does not break.

▼ What is a .gitignore file?

you can ignore node_modules/ in this file

▼ What is package-lock.json?

it holds the exact versions of the packages that are installed

the file helps make sure that all the team members have the exact same packages (& package versions) on their machines

▼ What is the yarn package manager and why use it?

it has a flat mode that guarantees that there will be a single version of a package installed in all scenarios.

That's because sometimes you've got the same package required multiple times but in different versions.

npm would install all versions requested, whereas yarn will make sure that only one of them is installed

▼ When you link a JavaScript file to your HTML file, you do that with a script tag

<script src="index.js"></script>

▼ If you've ever seen an error in your browser saying that the import is causing a syntax error, it because why?

import/export does not work in traditional scripts

▼ What is JavaScript Modules?

a script that supports import/export syntax

```
<script type="module">
    import {something} from "./file.js";
    // works as expected
</script>
```

▼ What is a bare import?

when you import a package rather than a file name

import {Component} from "react";

▼ What is a module bundler (e.g. webpack or parcel)?

a build tool that understands your imports and is able to efficiently merge your files together into a single or multiple file set that you can deploy to a web server

▼ What does resolve your dependencies mean?

when you import a library e.g. 'lodash' then webpack will resolve that import and find the actual file that needs to be imported (for example, './node_modules/lodash/dist/lodash.min.js'

▼ What is route-based code splitting?

You can also merge files based on the URL.

For example, the homepage will have a homepage.js file, the settings will have a settings.js file, etc.

▼ Tools that generate service workers need to run when?

after a build has been completed, which can also be run using the module bundler

▼ What is cache busting?

a technique that allows you to generate files with a hash in their name to maximize the caching capability of the browser

For example, your app.js becomes: app-9d0bc8147e2da823.js after building with the module bundler.

Then, only when the content of app.js changes, the hash (9d0bc8147e2da823) would change.

This allows the browser to cache app-9d0bc8147e2da823.js for a long period of time knowing that its content won't update.

And that **if its content has been updated, it will have a new URL**. For example, app-3cd24fb0d6963f7d.js.

▼ How to setup webpack?

1. npm install webpack webpack-cli webpack-dev-server --save-dev
2. Create index.html index.js webpack.config.js
    1. touch index.html index.js webpack.config.js
    2. index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>My App</title>
</head>
<body>
  <script src="bundle.js"></script>
</body>
</html>
```

    3. webpack.config.js

```
const path = require('path');

const config = {
  entry: './index.js',
  output: {
    path: path.resolve(__dirname, 'dist'),
    filename: 'bundle.js'
  }
};

module.exports = config;
```

3. Run the server
    1. npx webpack serve
    2. You can now open http://localhost:8080 in your browser and make changes to your HTML & JS files which **will automatically update in the browser**

▼ What is webpack-dev-server?

a package that allows you to have a local web server

▼ What is webpack.config.js?

a file used to store the configuration for webpack

▼ In what environment does the following syntax work: require and module.exports?

only works in the NodeJS environment

They don't work in the browser.

▼ What is Lodash?

a JavaScript library that makes it easier to work with collections (arrays of objects), objects, strings, etc

It also provides commonly used functionality such as the ability to debounce a function

▼ What is debouncing a function?

To debounce a function is to make it wait a certain amount of time before running again.

The goal of debouncing a function is to reduce overhead by preventing it from being called several times in succession.

debounce(callback, waitMilliseconds)

```
import {debounce} from "lodash-es";

window.addEventListener("scroll", debounce(() => {
    console.log("User has scrolled");
}, 30));
```

▼ What date library is recommended over moment.js?

date-fns is a collection of functions that make it easier to work with dates in JavaScript

▼ Whenever you import a file in your JavaScript code, the import will happen how?

synchronously

This is a static import.

The imports are processed synchronously meaning that the browser will wait until the file has been imported before running the rest of the code. More imports = slower load

▼ What is a dynamic import?

it only loads the module when the dynamic import() function is executed

When you use import() as a function (dynamic import), it will return a promise.

When the promise fulfills, it will resolve with a module object that contains the exported data.

```
import("lodash-es").then(module => {
    const debounce = module.default; // because debounce is a default export
    // use debounce inside this function...
}).catch(error => console.error(error)); // handle network/import errors

import("./helpers.js").then(module => {
    const Helpers = module.default; // because Helpers is a default export
    const getDate = module.getDate; // because getDate is a named export
    // use Helpers and getDate inside this function...
}).catch(error => console.error(error));
```

▼ How to dynamically import with async/await?

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

button.addEventListener("click", async () => {
    try {
      const module = await import("chat-library");
      module.init();
    } catch (error) {
        console.error(error); // handle network/import errors
    }
});
```

▼ How to dynamically import with destructuring?

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

button.addEventListener("click", async () => {
    try {
      const {init} = await import("chat-library");
      init();
    } catch (error) {
        console.error(error); // handle network/import errors
    }
});
```

▼ What is EcmaScript?

JavaScript is an implementation of EcmaScript.

EcmaScript is the specification, and JavaScript is the language.

▼ EcmaScript is a specification for a scripting language managed by who?

an organization called Ecma (Ecma International) which is headquartered in Geneva, Switzerland

▼ Vanilla JavaScript?

JavaScript without a framework

▼ What is the unofficially referred to as the "old version of JavaScript"?

ES5 (EcmaScript version 5)

▼ ES6 or ES2015 was a big update

- let/const
- spread operator
- destructuring
- arrow functions
- lexical this

▼ What is EcmaScript stage 4?

This is the final stage! Once your proposal reaches stage 4, it means it will be included in this year's EcmaScript release.

When a new JavaScript proposal reaches stage 4, it does not mean you can start using it in production because it takes a while for other browsers to include it and it takes time for users to upgrade their browsers.

▼ What JS features are generally safe to use?

It's generally safe to use new language features from the previous year.

▼ Why use Babel for the latest features?

If you really want to use the latest features, then you can set up babel which will "transpile" modern syntax into older syntax in a way that will work on old browsers.

Most libraries/frameworks (such as React, Angular, etc.) already include babel.

You will only have to include the plugin responsible for making that syntax work.

▼ What scope is let/const?

block scope

which means variables are only accessible in the nearest block

▼ What scope is var?

function scope

When you define a variable with var, it will be scoped to the nearest function.

▼ What is hoisting (only applies when you use var)?

Hoisting in JavaScript is when the variables you define inside a function are moved to the top of the function. This happens every time you define a variable using var.

▼ What is the Temporal Dead Zone (TDZ) as related to let/const ?

The Temporal Dead Zone is a fancy way of saying that variables defined with let and const cannot be accessed before they are initialized.

Any line of code that uses the variable name before it was defined, is called Temporal Dead Zone.

```
console.log(name); // this is the Temporal Dead Zone
let name = "Sam";
```

▼ How to convert old code with var?

In most cases, you can swap var with let and things will keep on working as expected.

▼ Functions defined with let and const are not

hoisted

▼ Only call functions when?

after they were defined.

▼ What is the window object?

it represents the current page and includes the DOM (document) as well as all the global variables and functions

▼ Every time you create a function in JavaScript, you create what?

a closure

▼ What is a closure?

A closure is where an inner function has access to the outer function's variables.

▼ What is the scope chain for a closure? (where does JavaScript starts looking for where to find a variable)

1. Variables in their own scope (variables defined between the curly braces of a function).
2. Variables defined in the outer function's scope. - scope outside of the current function
3. Variables defined in the global scope.

▼ Why use a for loop?

When incrementing, decrementing, skipping numbers in iteration, counting

```
for (let counter = 1; counter <= 10; counter++) {
    console.log(counter);
}
```

▼ for...of has the added benefit of what keywords?

break and continue keywords

```
const people = ["Sam", "Alex"];

for (const person of people) {
    console.log(person);
}
```

▼ What does continue keyword do?

allows you to skip the remaining body of the iteration and continuing with the next iteration

```
const items = [1, 2, 3, 4];

for (const item of items) {
    if (item % 2 === 0) {
        console.log("Even number, skip the iteration");
        continue;
    }
    console.log("Odd number", item);
}
```

▼ What does break keyword do?

The break keyword allows you to quit the iteration completely early on

```
let sum = 0;

for (const number of numbers) {
    sum += number;
    if (sum >= 500) {
        break;
    }
}
```

▼ What does for...of let you iterate over?

Strings, Arrays, NodeList (the result of document.querySelectorAll), Map, and Set.

▼ How can you loop over objects with for...in?

```
const person = {
    id: 1,
    name: "Alex"
};

for (const key in person) {
    console.log(key);
    console.log(person[key]);
}
```

▼ Why is JavaScript single-threaded?

it can process instructions one instruction at a time

▼ What is the call stack?

an array where the JavaScript interpreter keeps track of the code that you're running

▼ What does the call stack store?

it stores what needs to be called/executed aka instructions

▼ Why do Web APIs such as setTimeout work differently?

due to the design of JavaScript and the way it interacts with the visual component of the Browser (think about the DOM, user interaction with the DOM, network requests, etc.)

▼ What stores the callbacks from WebAPIs?

the callback queue

▼ When are items from the callback queue added to the call stack?

when the call stack is empty

▼ When the call stack is empty, the browser does what?

takes a task from the callback queue and executes it

▼ What is the event loop?

a task that keeps checking if there are any items in the callback queue and forwards them to the call stack but only when the call stack is empty