

# DWA\_02.8 Knowledge Check\_DWA2

---

1. What do ES5, ES6 and ES2015 mean - and what are the differences between them?

These are all versions of ECMAScript, the standardized language specification that JavaScript is based on. The main difference between these specifications is the set of new features and syntax introduced in follow up specification/ version. ES5 introduced new features such as JSON support and strict mode. The next version ES6, introduced arrow functions, classes, modules, and more. ES2015 is simply another name for ES6, indicating that it was finalized in 2015.

---

2. What are JScript, ActionScript and ECMAScript - and how do they relate to JavaScript?

JScript and ActionScript are both scripting languages that are similar to JavaScript but have specific contexts in which they are used. JScript is a scripting language developed by Microsoft and primarily used in their Internet Explorer web browser. ActionScript is a scripting language used in Adobe Flash, a multimedia software platform. Both JScript and ActionScript are derived from ECMAScript and have similarities with JavaScript, but they have their own unique features and capabilities.

ECMAScript is the standardized specification for scripting languages, including JavaScript. JavaScript is the most widely used implementation of ECMAScript and is commonly used for web development. JavaScript is essentially the implementation of the ECMAScript specification in web browsers and other environments.

---

3. What is an example of a JavaScript specification - and where can you find it?

An example of a JavaScript specification is the ECMAScript Language Specification. It defines the syntax, semantics, and behavior of the ECMAScript scripting language. The ECMAScript Language Specification provides the guidelines and rules that developers and implementers need to follow to ensure compatibility and interoperability. You can

find the ECMAScript Language Specification on the Ecma International website or by searching for "ECMAScript Language Specification" online.

---

#### 4. What are v8, SpiderMonkey, Chakra and Tamarin? Do they run JavaScript differently?

V8, SpiderMonkey, Chakra, and Tamarin are all JavaScript engines. JavaScript engines are responsible for executing JavaScript code in web browsers and other environments. Each engine has its own implementation of the ECMAScript specification, and they may have differences in performance, optimization techniques, and supported features.

V8 is the JavaScript engine developed by Google, and it powers the Chrome web browser. It is known for its high-performance execution and is also used in other projects like Node.js.

SpiderMonkey is the JavaScript engine developed by Mozilla, and it is used in the Firefox web browser. SpiderMonkey has a long history and was one of the first JavaScript engines.

Chakra was the JavaScript engine developed by Microsoft for their Edge web browser. However, Microsoft has replaced Chakra with a new engine called "ChakraCore," which is used in the newer versions of Microsoft Edge.

Tamarin was a JavaScript engine developed by Adobe, primarily for use in the Flash Player. However, Adobe announced the end of life for Flash Player, and Tamarin is no longer actively maintained.

While these engines may have differences in their internal workings and optimizations, they aim to provide compatibility with the ECMAScript specification and execute JavaScript code correctly.

---

5. Show a practical example using [caniuse.com](https://caniuse.com) and the MDN compatibility table.

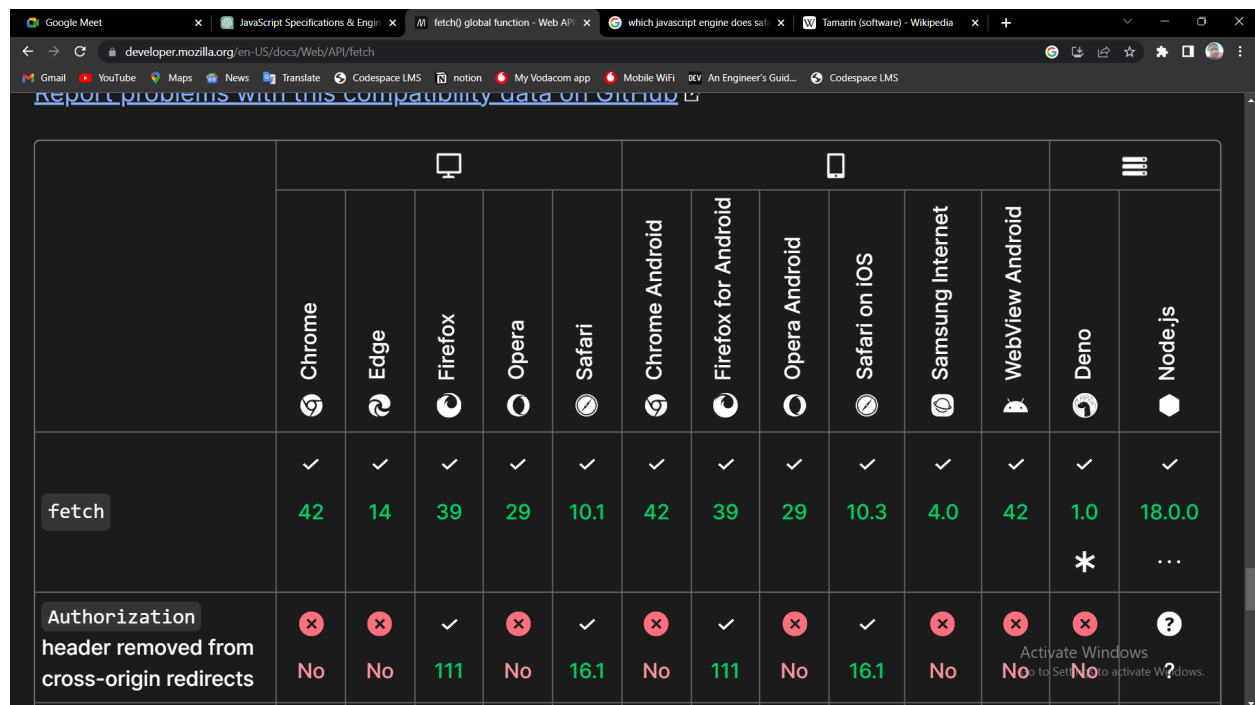
CHROME - v8

Medge - Chakra

Firefox - SpiderMonkey

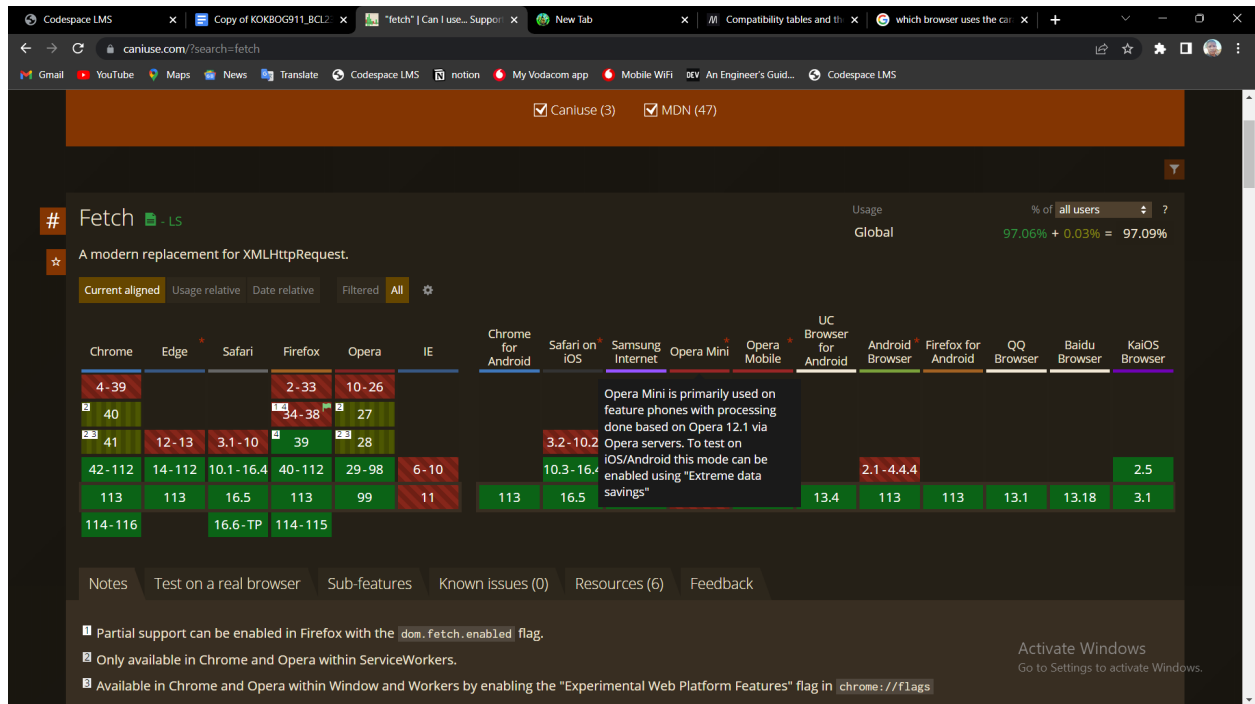
Opera - Carakan

Safari - Webkit



	Desktop					Mobile					Other		
	Chrome	Edge	Firefox	Opera	Safari	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet	WebView Android	Deno	Node.js
fetch	✓ 42	✓ 14	✓ 39	✓ 29	✓ 10.1	✓ 42	✓ 39	✓ 29	✓ 10.3	✓ 4.0	✓ 42	✓ 1.0	✓ 18.0.0
Authorization header removed from cross-origin redirects	✗ No	✗ No	✓ 111	✗ No	✓ 16.1	✗ No	✓ 111	✗ No	✓ 16.1	✗ No	✗ No	✗ No	? ?

MDN shows that it offers all browsers support the fetch feature. For Node.js which uses the v8 engine, From version 16.15.0: this feature is behind the `--experimental-fetch` runtime flag and also From version 17.5.0: this feature is behind the `--experimental-fetch` runtime flag.



"Partial support can be enabled in Firefox with the `dom.fetch.enabled` flag":

This means that Firefox has partial support for the fetch API, but it is not enabled by default.

"Only available in Chrome and Opera within ServiceWorkers":

The fetch API is supported in Chrome and Opera, but only within the context of Service Workers.

"Available in Chrome and Opera within Window and Workers by enabling the 'Experimental Web Platform Features' flag in `chrome://flags`":

In Chrome and Opera, the fetch API is available not only within Service Workers but also within the Window and Workers contexts.

"Firefox <40 is not completely conforming to the specs and does not respect the `<base>` tag for relative URIs in fetch requests":

This information pertains to an issue in earlier versions of Firefox (before version 40). In those versions, Firefox did not fully conform to the specifications of the fetch API.

Overall, this information highlights the varying levels of support and behavior for the fetch API across different browsers and versions, and it underscores the importance of considering these differences when using the fetch API in your JavaScript code.

