DWA_02.8 Knowledge Check_DWA2

---

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

ES5, ES6, and ES2015 are all different versions of the ECMAScript standard, which is a standardized scripting language specification that JavaScript is based on. Here's a breakdown of what each term means and the differences between them:

1. ES5 (ECMAScript 5): It was released in 2009 and introduced significant updates to the language. Some notable features and improvements in ES5 include strict mode, which enforces stricter syntax and error handling, native JSON support, new array methods (such as `forEach`, `map`, `filter`, etc.), and improvements to the `Object` and `Function` prototypes. ES5 is widely supported by modern web browsers and is considered the baseline for JavaScript development.

2. ES6 (ECMAScript 2015): ES6, also known as ECMAScript 2015, was released in June 2015. It brought several major changes and additions to JavaScript, making the language more powerful and expressive. Some notable features introduced in ES6 include block-scoped variables (`let` and `const`), arrow functions, template literals, classes, modules, destructuring assignment, and enhanced object literals. ES6 also introduced Promises for handling asynchronous operations, as well as generators and iterators. ES6 marked a significant milestone in the evolution of JavaScript and introduced many features that are now widely used.

3. ES2015: ES2015 is simply another name for ES6. It refers to the same version of the ECMAScript standard but is often used to avoid confusion with the incremental versioning that came after ES6. The ECMAScript standard began using a year-based versioning scheme starting from ES2015, with new editions released each year, incorporating new features and improvements. So, ES2015 is essentially synonymous with ES6.

It's worth noting that subsequent versions of ECMAScript, such as ES2016, ES2017, and so on, introduced additional features and enhancements to the language. These newer versions built upon the foundation of ES6/ES2015 and extended JavaScript's capabilities further. Developers often use transpilers like Babel to convert code written in newer versions to ES5-compatible code, ensuring broader browser compatibility.

---

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

JScript, ActionScript, and ECMAScript are all scripting languages that are closely related to JavaScript. Here's a brief explanation of each and their relationship to JavaScript:

1. JScript: JScript is a scripting language developed by Microsoft. It is essentially Microsoft's implementation of the ECMAScript standard. JScript is very similar to JavaScript and shares many of the same syntax and features. In fact, JScript is often considered a dialect of ECMAScript, specifically tailored for use in Microsoft technologies such as Internet Explorer and Windows Script Host.

2. ActionScript: ActionScript is a scripting language primarily used for creating interactive content and applications within Adobe Flash (now deprecated) and Adobe AIR platforms. ActionScript is also based on the ECMAScript standard and shares similarities with JavaScript. It has its own specific features and syntax for working with multimedia, animations, and the Flash platform.

3. ECMAScript: ECMAScript is the standardized specification for scripting languages that JavaScript, JScript, and ActionScript are based on. ECMAScript defines the syntax, semantics, and behavior of these languages. JavaScript is the most widely known and used implementation of ECMAScript, often used for client-side scripting in web browsers. JScript is Microsoft's implementation of ECMAScript, and ActionScript is the implementation used in Adobe Flash and AIR.

In summary, JavaScript, JScript, and ActionScript are all implementations of the ECMAScript standard, with each implementation having its own specific features and target platforms. JavaScript is the most commonly used implementation and is often used for web development, while JScript and ActionScript have been used primarily in Microsoft and Adobe environments, respectively.

---

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, which defines the standard for the JavaScript language. The specification describes the syntax, semantics, and behavior of JavaScript in detail.

The ECMAScript Language Specification is maintained by the Ecma International organization, specifically Technical Committee 39 (TC39). TC39 is responsible for evolving and updating the ECMAScript standard.

You can find the ECMAScript Language Specification in its official form on the Ecma International website. The specification is available as a document in PDF format, and you can access it by visiting the following URL:

https://www.ecma-international.org/publications/standards/Ecma-262.htm

On the page, you will find links to download the latest version of the ECMAScript specification, as well as previous editions. The specification provides in-depth information about the JavaScript language, including its syntax, data types, operators, control flow, object model, and more. It serves as a comprehensive reference for understanding the intricacies of JavaScript.

---

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

V8, SpiderMonkey, Chakra, and Tamarin are all JavaScript engines, each used by different web browsers or platforms to execute JavaScript code. While they all serve the same purpose of interpreting and executing JavaScript, they have different design choices and implementation details, leading to variations in performance and behavior. Here's a brief overview of each JavaScript engine:

1. V8: V8 is an open-source JavaScript engine developed by Google. It is primarily used in the Google Chrome browser, but it has also been adopted by other browsers such as Opera and Brave. V8 compiles JavaScript code into machine code using a just-in-time (JIT) compilation technique called "Ignition" for optimizing execution speed. V8 is known for its high-performance capabilities and has introduced various innovative features and optimizations.

2. SpiderMonkey: SpiderMonkey is the JavaScript engine developed by the Mozilla Foundation. It is used in the Firefox web browser and other Mozilla projects. SpiderMonkey was the first JavaScript engine ever created and has a long history. It uses a combination of interpretation and JIT compilation to execute JavaScript code. SpiderMonkey has been instrumental in driving the development of JavaScript standards and has implemented several advanced features.

3. Chakra: Chakra is the JavaScript engine developed by Microsoft. It was initially used in the Internet Explorer browser and is now primarily used in Microsoft Edge. Chakra has undergone significant transformations over time. In its original form, known as Chakra Classic, it used an interpreter. However, with the release of Microsoft EdgeHTML, Chakra was redesigned and re-implemented as ChakraCore, utilizing a JIT compilation technique. ChakraCore has been open-sourced and is used beyond web browsers in various Microsoft platforms.

4. Tamarin: Tamarin was a JavaScript engine developed by Adobe Systems. It was designed for the Adobe Flash Player and its virtual machine, allowing ActionScript (a dialect of ECMAScript) to be executed. Tamarin used a JIT compilation technique called "nanojit" for optimizing performance. However, Adobe officially discontinued development of Tamarin in 2012.

While these JavaScript engines have differences in their implementation, they all strive to execute JavaScript code efficiently. They continuously evolve to incorporate new language features, improve performance, and adhere to the ECMAScript

specifications. JavaScript developers benefit from these advancements, as they can write code that runs consistently across different platforms and browsers, leveraging the optimizations provided by each JavaScript engine.

---

5. Show a practical example using **caniuse.com** and the MDN compatibility table.

Certainly! Let's use the `caniuse.com` website and the MDN compatibility table to check the browser compatibility of the `fetch()` function in JavaScript.

Step 1: Visit caniuse.com
Go to the caniuse website by typing "caniuse.com" in your browser's address bar and pressing Enter.

Step 2: Search for `fetch()`
In the search bar on the caniuse website, type "fetch" and press Enter or click the search icon.

Step 3: View browser compatibility
The search results will display information about the `fetch()` function. Look for the "Global" support table, which shows the percentage of users who can use the feature in each browser version. The table also indicates if a polyfill or prefixed version of `fetch()` is required for certain browsers.

Step 4: Visit MDN compatibility table
To gather more detailed information about browser compatibility, open a new tab and navigate to the Mozilla Developer Network (MDN) website. In the MDN search bar, type "fetch()" and press Enter.

Step 5: View compatibility table
The search results will display the documentation for the `fetch()` function on MDN. Scroll down to the "Browser compatibility" section, where you'll find a table listing the compatibility of `fetch()` across various browsers. The table provides detailed

information about which browser versions fully support `fetch()` and if any specific considerations or workarounds are needed.

By combining the information from caniuse.com and the MDN compatibility table, you can assess the browser compatibility of the `fetch()` function and determine the need for any additional steps or fallbacks to ensure your code works across different browsers.
