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 versions of the ECMAScript standard, which is the standardized specification for JavaScript. Each version introduces new features, syntax, and functionality to the language. Here's a breakdown of what they mean and their differences:

ES5 (ECMAScript 5): ES5 was released in 2009 and is widely supported by all modern web browsers. It introduced significant improvements to the language, including new methods for manipulating arrays (forEach, map, reduce, etc.), additional object manipulation methods (Object.keys, Object.defineProperty, etc.), and support for JSON (JavaScript Object Notation). ES5 is the version of JavaScript that was commonly used before the introduction of ES6.

ES6 (ECMAScript 2015): ES6, also known as ECMAScript 2015, was released in 2015 and brought several major enhancements to JavaScript. It introduced new features such as block-scoped variables (let and const), arrow functions, classes, template literals, modules, and destructuring assignments. ES6 introduced a more modern syntax and made JavaScript development more efficient and expressive. However, not all ES6 features are supported in older web browsers, so developers often use transpilers like Babel to convert ES6 code into ES5-compatible code for broader compatibility.

ES2015: ES2015 is another name for ES6. It refers to the year of its release, and it's commonly used interchangeably with ES6.

To summarize, ES5 is an earlier version of the ECMAScript standard, while ES6 (or ES2015) introduced significant improvements and new features to JavaScript. Developers now commonly use ES6 (and subsequent versions) for modern web development, leveraging the newer syntax and features. However, when targeting older browsers, it's important to transpile or polyfill the code to ensure compatibility with ES5.

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

JScript, ActionScript, and ECMAScript are scripting languages that are related to JavaScript in different ways.

JScript: JScript is a scripting language developed by Microsoft. It is essentially Microsoft's implementation of the ECMAScript standard. JScript shares a lot of similarities with JavaScript because it was based on ECMAScript, particularly ECMAScript 3 (ES3). JScript was primarily used in Microsoft's Internet Explorer browser and Windows scripting environments. While JScript and JavaScript are similar, there may be some slight differences in their implementations and features due to the specific environments they target.

ActionScript: ActionScript is a scripting language primarily used for developing interactive applications and multimedia content in Adobe Flash. ActionScript is also based on the ECMAScript standard, specifically ECMAScript 4 (ES4). It extends the ECMAScript syntax to include features specific to Flash, such as timeline-based animation and manipulation of vector graphics. ActionScript has a strong resemblance to JavaScript, but it has its own unique features and capabilities tailored for the Flash platform.

ECMAScript: ECMAScript is a standardized scripting language specification. It defines the syntax, semantics, and behavior of scripting languages like JavaScript, JScript, and ActionScript. ECMAScript serves as the foundation for these languages, providing a set of rules and guidelines for their implementation. JavaScript is the most well-known and widely used implementation of the ECMAScript standard, often used in web development. JScript and ActionScript are specific implementations of ECMAScript by Microsoft and Adobe, respectively, with some modifications and additions specific to their respective platforms.

In summary, JScript and ActionScript are implementations of the ECMAScript standard by Microsoft and Adobe, respectively, with some platform-specific features. JavaScript is the most commonly used implementation of ECMAScript, known for its broad adoption and versatility in web development. ECMAScript is the standardized specification that provides the foundation for these scripting languages, defining their syntax and behavior.

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

An example of a JavaScript specification is the ECMAScript specification. It defines the syntax, semantics, and behavior of the JavaScript programming language. The ECMAScript specification provides a detailed description of how JavaScript should work, including its core features, data types, control structures, objects, and more.

The ECMAScript specification is maintained and updated by the Ecma International organization, specifically the Technical Committee 39 (TC39). TC39 is responsible for evolving and standardizing ECMAScript. The latest edition of the ECMAScript specification, as of my knowledge cutoff in September 2021, is ECMAScript 2022.

You can find the ECMAScript specification on the Ecma International website. The specific URL for accessing the ECMAScript specification may change over time.

On the Ecma International website, you can access the latest version of the ECMAScript specification, as well as previous versions. The specification documents provide in-depth details about the language's features, syntax, and behavior, making it a valuable resource for understanding and implementing JavaScript.

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

V8, SpiderMonkey, Chakra, and Tamarin are all JavaScript engines, which are responsible for executing JavaScript code. Each engine is developed by different organizations and may have unique approaches to running JavaScript. Here's an overview of each engine:

V8: V8 is an open-source JavaScript engine developed by Google. It is used in the Google Chrome web browser and the Node.js runtime environment. V8 is known for its high-performance and efficient execution of JavaScript code. It uses just-in-time (JIT) compilation to optimize code execution, including techniques like inline caching and hidden class transitions. V8 also introduced the concept of the JavaScript engine as a standalone component, making it possible to embed V8 in other applications beyond web browsers.

SpiderMonkey: SpiderMonkey is the JavaScript engine developed by Mozilla, the organization behind the Firefox web browser. It was one of the first JavaScript engines ever created and is written in C++. SpiderMonkey has evolved over time and incorporates various performance optimizations. It uses a combination of interpreted and JIT compilation techniques to execute JavaScript code efficiently. SpiderMonkey has a long history and has played a significant role in the development of the JavaScript language.

Chakra: Chakra, also known as ChakraCore, is the JavaScript engine initially developed by Microsoft for its Edge web browser. ChakraCore is the open-source version of Chakra and can be used independently of the Edge browser. It supports both just-in-time (JIT) and ahead-of-time (AOT) compilation strategies. ChakraCore has been designed with a focus on performance and includes features like profile-guided optimization to improve execution speed.

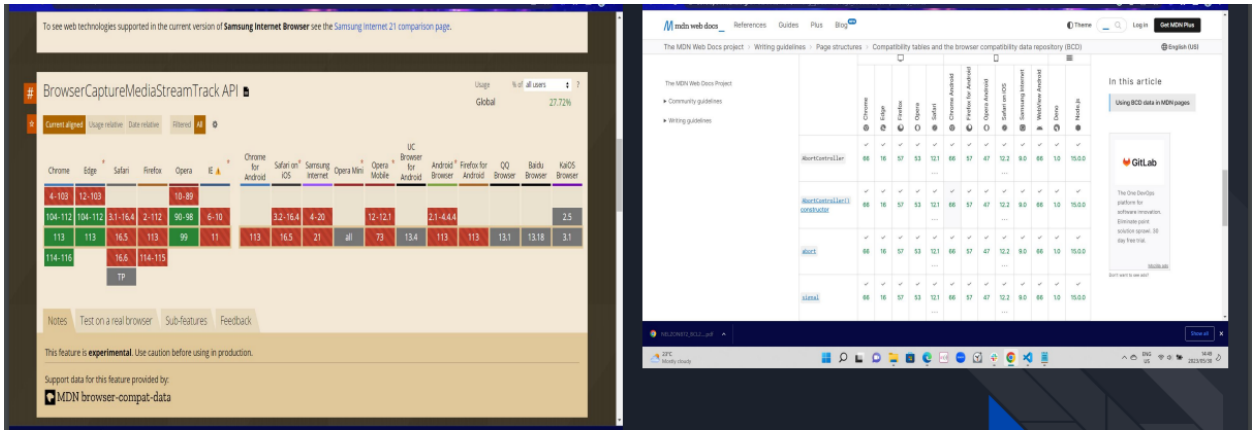
Tamarin: Tamarin is a JavaScript engine that was initially developed by Adobe Systems for the Adobe Flash platform. It is no longer actively maintained. Tamarin aimed to execute JavaScript and ActionScript (based on ECMAScript) code efficiently. Tamarin used a technique called trace-based just-in-time compilation to optimize code execution.

While all these JavaScript engines are designed to execute JavaScript code, they may have different performance characteristics, optimization strategies, and additional features. Each engine may have its own unique implementation details, but they all adhere to the ECMAScript specification to ensure compatibility with JavaScript code.

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

MDN has a standard format of tables that illustrate compatibility of shared technology such as DOM, HTML, CSS, JavaScript, SVG, etc while **caniuse** provides up-to-date browser support of front-end web technologies on desktop and mobile web browsers

BELOW are the Compatibility Tables between MDN & CANIUSE



MDN (Mozilla Developer Network) and **caniuse** are both reliable sources of information about web technologies and their compatibility across different browsers. While both platforms provide compatibility tables, there are some differences in terms of the data they present and the way they present it. Here are some key points to consider when comparing the compatibility tables from MDN and Can I use:

Data Sources: MDN's compatibility tables are based on information provided by browser vendors, web developers, and contributors to MDN itself. **caniuse** collects data from a variety of sources, including browser vendors, documentation, and community contributions.

Scope of Coverage: Both MDN and **caniuse** cover a wide range of web technologies, including HTML, CSS, JavaScript, and various APIs. However, MDN often provides more detailed and extensive documentation, including comprehensive information about the features, syntax, and usage of web technologies.

Browser Support: MDN's compatibility tables primarily focus on the support provided by major browser engines (e.g., Blink, Gecko, WebKit), detailing which versions of each engine support a particular feature. **caniuse**, on the other hand, offers a broader perspective by providing information on the support of specific features in individual browser versions.

Level of Detail: MDN's compatibility tables often provide detailed notes and explanations about the implementation and usage of a feature in different browsers. **Canluse** offers concise and visually appealing tables, highlighting the support status of a feature across browsers using color-coded indicators.