

DWA_02.8 Knowledge Check_DWA2

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

Each programming language has its own standardised way of writing code. And JavaScript follows both ES5 and ES6 scripting language specification but when writing a JavaScript code using these standards, there are slight differences between ES5 and ES6.

ECMAScript 5 (ES5) which was released in 2009, is used to create dynamic web pages and is the basis of several programming languages, including JavaScript. It is a function contractors focused on how the objects are created. For ES5 you have to write a function keyword and return, to be used to define the function, like normal general JavaScript language.

ES6, also known as ES2015, was released in 2015 and introduced many new features such as let and const declarations, arrow functions, classes, modules, and template literals, and destructuring assignment.

Here are some differences between ES5 and ES6:

Variables - In ES5, there is a one way of defining variables using var keyword. But ES6 has introduced two more ways of defining variables using keywords let and const. However, let and const use block scope while var uses function scope. Also, the variables that are defined using const are immutable.

```
> //ES5
var age = 20;
//ES6
let days = 5;
const pi = 3; // immutable
```

Define Objects - ES6 provides an easy way to define objects when the keys and the variable names are the same.

```
> var fullName = 'John Moore';  
  var age = 25;  
  var gender = 'Male';  
  var city = 'London'  
  // ES5  
  var student = { fullName: fullName, age: age, gender: gender, city: city };  
  // ES6  
  var student = { fullName, age, gender, city };
```

Object Destructuring - In ES5, Objects are extracted one by one manually. But ES6 has made it to one line of code.

```
> var student = { fullName: 'John Moore', age: 25, gender: 'Male', city: 'London' };  
  //ES5  
  var fullName = student.fullName;  
  var age = student.age;  
  var gender = student.gender;  
  var city = student.city;  
  //ES6  
  var { fullName, age, gender, city } = student;
```

String Interpolation - In ES5, Concatenation operator is used to join strings. But ES6 has introduced Template Literal() which allows to perform the string interpolation in a convenient way.

```
> var fullName = 'Malebo Legodi';  
  var age = 33;  
  var gender = 'Female';  
  var city = 'Johannesburg'  
  //ES5  
  var intro = 'Hello, I am ' + fullName + '. I am ' + age + ' years old ' + gender + '  
  student from ' + city + '.';  
  //ES6  
  var intro = `Hello, I am ${fullName}. I am ${age} years old ${gender} student from  
  ${city}.`;
```

Import Module - ES5 uses require keyword while ES6 uses import keyword. ES6 allows to import child modules as well.

```
> //ES5  
  var App = require('./App');  
  //ES6  
  import App from './App';  
  import { Header, Sticky } from '@primer/components'; //child modules
```

Export Module - ES6 allows to export child modules and variables as well.

```
> var Student = { fullName: 'John Moore', age: 25, gender: 'Male', city: 'London' };  
//ES5  
module.exports = Student;  
//ES6  
export default Student;  
export const fullName = 'John Moore'; //child variables  
export const age = 25;
```

Arrow Function - ES6 has introduced the arrow function. It does not require the keyword function.

```
> //ES5  
function sayHello(name) {  
  console.log('Hello, ' + name);  
}  
//ES6  
const sayHello = (name) => {  
  console.log(`Hello, ${name}`);  
}  
const sayHello = name => console.log(`Hello, ${name}`); //You can ignore parenthesis, if  
the function contains a single parameter and only one statement
```

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

- ECMAScript (ES) is a scripting language specification standardized by ECMA International. It is used to create dynamic web pages and is the basis of several programming languages, including JavaScript. ES versions have been abbreviated to ES1, ES2, ES3, ES5, and ES6. Since 2016, versions are named by year (ECMAScript 2016, 2017, 2018, 2019, 2020).
- JavaScript is a high-level programming language that conforms to the ECMAScript specification. It was developed by Netscape and first released in 1995. JavaScript is used primarily for creating interactive web pages and user interfaces.
- JScript is Microsoft's implementation of the ECMAScript specification. It was first released in 1996 and is used primarily for server-side scripting in Microsoft's Internet Information Services (IIS).
- ActionScript is a scripting language based on ECMAScript that is used primarily for creating interactive applications and video games in Adobe Flash Player.

In summary, ECMAScript is a standard for scripting languages that JavaScript, JScript, and ActionScript are all based on. JavaScript is the most widely used implementation of ECMAScript and is used primarily for creating interactive web pages. JScript is Microsoft's implementation of ECMAScript and is used primarily for server-side scripting in IIS. ActionScript is a scripting language based on ECMAScript that is used primarily for creating interactive applications and video games in Adobe Flash Player.

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

An example of a JavaScript specification is the ECMA-262 specification. It is the most in-depth, detailed, and formalized information about JavaScript. It defines the language and is the most trustworthy source of information about the language details.



```
index.html JS scripts.js x
Mod02 > JS scripts.js > getES
1 function getES() {
2   var array = []
3   if (!Array.isArray) {
4     return 3
5   } else if (!window.Promise) {
6     return 5
7   } else if (!array.includes) {
8     return 6
9   } else if (!"".padStart) {
10    return 7
11   } else if (!Promise.prototype.finally) {
12    return 8
13   } else if (!window.BigInt) {
14    return 9
15   } else if (!Promise.allSettled) {
16    return 10
17   } else if (!"".replaceAll) {
18    return 11
19   } else if (array.at) {
20    return 12
21   } else if (array.with) {
22    return 13
23   } else {
24     return 14
25   }
26 }
27
28 var ecmaScript = getES()
29 var year = 1999
30 if (ecmaScript !== 3) {
31   year = ecmaScript + 2009
32 }
33 console.log("Version " + ecmaScript + ", Year " + year)
```

The latest version of the specification is ECMAScript 2023. You can find the latest draft of the specification at <http://www.ecma-international.org/ecma-262/5.1/ECMA-262.pdf>.

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

- **V8** is a JavaScript compiler developed by Google for the Chrome browser. It compiles JavaScript code to machine code for faster execution.
- **SpiderMonkey** is a JavaScript compiler developed by Mozilla for the Firefox browser. It compiles JavaScript code to bytecode, which is then interpreted by the compiler.
- **Chakra** is a JavaScript compiler developed by Microsoft for the Edge browser. It compiles JavaScript code to bytecode, which is then interpreted by the compiler.
- **Tamarin** is a JavaScript compiler developed by Adobe for the Flash Player. It compiles ActionScript (a language based on ECMAScript) to bytecode, which is then interpreted by the compiler.

All of these compilers run JavaScript differently. They use different techniques to optimise and execute JavaScript code. For example, V8 compiles JavaScript code to machine code, while SpiderMonkey and Chakra compile it to bytecode. This difference in compilation can lead to differences in performance and memory usage between compilers.

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

The screenshot shows the CanIUse website interface. At the top, there's a navigation bar with 'Home' and 'News' links. Below that, a search bar with the text 'Can I use' and a 'Settings' button. The main content area is divided into several sections:

- Index of features**: A dropdown menu to filter features.
- Latest features**: A list of recent features including 'zstd (Zstandard) content-encoding', 'Scoped Styles: the @scope rule', 'View Transitions API (single-document)', 'Passkeys', and 'CSS text-wrap: balance'.
- Most searched features**: A list of popular features including 'WebP image format', 'Flexbox', 'gap property for Flexbox', 'CSS Grid', and 'WebGL'.
- Test a feature**: A section for testing features, mentioning a partnership with BrowserStack and providing a list of browsers to test on: 'IE 11', 'Safari 13', 'Safari on iPhone 11 Pro', and 'Chrome on Galaxy S20'.
- Browser scores**: A section showing scores for 'Current version' and 'Dev version' of browsers, with a bar chart for 'Chrome 118: 414', 'Firefox 118: 393', and 'Safari 17.1: 390'.
- Did you know?**: A section providing tips on how to use the site, such as submitting research or using the command line tool.
- Third party tools**: A list of tools like 'CanIUse Embed', 'CanIUse Component', 'CanIUse command line tool', 'Doluse...? — Lint your CSS to check what features work', and 'I want to use — Select multiple features and see what % of users can use them'.

At the bottom, there's a 'See full list' link.

Can I use **ES6 MODULE** ? Settings

3 results found

JavaScript modules via script tag - Ls

Usage: Global 95.67% + 0.05% = 95.72%

Loading JavaScript module scripts (aka ES6 modules) using `<script type="module">` includes support for the `nomodule` attribute.

Current aligned Usage relative Date relative Filtered All

Chrome	Edge	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-59	12-14	3.1-10	2-53	10-46			3.2-10.2	4-7.4								
60	16-18	10.1	54-59	47			10.3	8.2-21								
61-117	79-117	11-17.0	60-117	48-102	6-10		11-17.0	12-12.1				2.1-4.4.4				2.5
118	118	17.1	118	103	11	118	17.1	22	all	73	15.5	118	118	13.1	13.18	3.1
119-121		17.2-TP	119-121				17.2									

Notes Test on a real browser Sub-features Known issues (2) Resources (11) Feedback

- 1 Support can be enabled via `about:flags`
- 2 Support can be enabled via `about:config`
- 3 Does not support the `nomodule` attribute
- 4 A [polyfill is available](#) for Safari 10.1/iOS Safari 10.3
- 5 `nomodule` scripts are not executed, but they're still fetched

Can I use **JavaScript modules: dynamic import()** - OTHER

Usage: Global 95.64%

Loading JavaScript modules dynamically using the `import()` syntax

Current aligned Usage relative Date relative Filtered All

Chrome	Edge	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-62	12-18	3.1-11	2-65	10-49			3.2-10.3	4-7.4								
63-117	79-117	11.1-17.0	67-117	50-102	6-10		11-17.0	8.2-21				2.1-4.4.4				2.5
118	118	17.1	118	103	11	118	17.1	22	all	73	15.5	118	118	13.1	13.18	3.1
119-121		17.2-TP	119-121				17.2									

Notes Test on a real browser Known issues (1) Resources (7) Feedback

- 1 Support can be enabled via `javascript.options.dynamicImport` flag. See [bug 1517546](#).

Can I use **ECMAScript 2015 (ES6)** - OTHER

Usage: Global 95.9% + 1.25% = 97.16%

Support for the ECMAScript 2015 specification. Features include Promises, Modules, Classes, Template Literals, Arrow Functions, Let and Const, Default Parameters, Generators, Destructuring Assignment, Rest & Spread, Map/Set & WeakMap/WeakSet and many more.

Current aligned Usage relative Date relative Filtered All

Chrome	Edge	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android	QQ Browser	Baidu Browser	KaiOS Browser
4-20	12-14	3.1-7	2-5	10-12.1			3.2-6.1	4								
21-50	15-18	7.1-9.1	6-53	15-37			7-9.3	5-21				2.1-4.3				
51-117	79-117	10-17.0	54-117	38-102	6-10		10-17.0	12-12.1				4.4-4.4.4				2.5
118	118	17.1	118	103	11	118	17.1	22	all	73	15.5	118	118	13.1	13.18	3.1
119-121		17.2-TP	119-121				17.2									

Notes Test on a real browser Sub-features Known issues (0) Resources (4) Feedback

As ES6 refers to a huge specification and browsers have various levels of support, "Supported" means at least 95% of the spec is supported. "Partial support" refers to at least 10% of the spec being supported. For full details see the [Kangax ES6 support table](#).

- 1 Notable partial support in IE11 includes (at least some) support for `const`, `let`, block-level function declaration, typed arrays, `Map`, `Set` and `WeakMap`.
- 2 Does not support [tail call optimization](#)
- 3 Only has partial Symbol support and partial support for `RegExp.prototype` properties

HomeNews

August 19, 2023 · New feature: zstd (Zstandard) content-encoding

Compare browsersAbout

Can I use

let

50 results found

☒ Caniuse (7)☒ MDN (43)

#letOTHER

Usage: Global

96.06% + 0.58% = 96.64%

Declares a variable with block level scope

Current aligned

Usage relative

Date relative

Filtered: All

Chrome	Edge	Safari	Firefox	Opera	IE	Chrome for Android	Safari on iOS	Samsung Internet	Opera Mini	Opera Mobile	UC Browser for Android	Android Browser	Firefox for Android	QQ Browser	Baidu Browser	KaIOS Browser
4-18				10-12.1												
19-40		3.1-9.1		15-22				3.2-9.3								
41-48		10-10.1	2-43	28-35				10-10.3	4							
49-117	12-117	11-17.0	44-117	36-102	6-10		11-17.0	5-21		12-12.1		2.1-4.4.4				2.5
118	118	17.1	118	103	11	118	17.1	22	all	73	15.5	118	118	13.1	13.18	3.1
119-121		17.2-TP	119-121				17.2									

Notes

Test on a real browser

Known issues (0)

Resources (4)

Feedback

¹

Supports a non-standard version that can only be used in script elements with a type attribute of `application/javascript;version=1.7`. As other browsers do not support these types of `script` tags this makes support useless for cross-browser support.

²

Requires the 'Experimental JavaScript features' flag to be enabled

³

Only supported in strict mode

⁴

`let` bindings in for loops are incorrectly treated as function-scoped instead of block-scoped.

⁵

`let` variables are not bound separately to each iteration of for loops