**Q. Js hoisting**

=> JavaScript **Hoisting** refers to the process whereby the interpreter appears to move the declaration of functions, variables or classes to the top of their scope, prior to execution of the code.

**Ref** : <https://developer.mozilla.org/en-US/docs/Glossary/Hoisting>

* **Variable Hoisting**

In terms of variables and constants, keyword **var** is hoisted and let and const dose not allow hoisting.

Ex: // program to display value

a = 5;

console.log(a);

var a; // 5

* However in javascript, initializations are not hoisted.

// program to display value

console.log(a);

var a = 5;

Output : undefined.

* **Function Hoisting**

A function can be called before declaring it. For example,

Ex : // program to print the text

greet();

function greet() {

console.log('Hi, there.');

}

Output : Hi, there

**Ref :** <https://www.programiz.com/javascript/hoisting>

**Q. ECMA , ES6**

**Ecma International** (formally European Computer Manufacturers Association) is a non-profit organization that develops standards in computer hardware, communications, and programming languages.

**An organization that creates standards for technologies.**

To illustrate an example of “standard” (though not one created by Ecma), think of all the keyboards you have ever used. Did the vast majority have letters in the same order, and a space bar, an Enter key, arrow keys, with numbers displayed in a row at the top? This is because most keyboard manufacturers base their keyboard design on the [QWERTY](https://en.wikipedia.org/wiki/QWERTY)layout standard.

On the web it is famous for being the organization which maintain [the ECMA-262 specification](https://www.ecma-international.org/publications-and-standards/standards/ecma-262/) (aka. [ECMAScript](https://developer.mozilla.org/en-US/docs/Glossary/ECMAScript)) which is the core specification for the [JavaScript](https://developer.mozilla.org/en-US/docs/Glossary/JavaScript) language.

**ECMA-262**

This is a standard published by Ecma International. It contains the specification for a general purpose scripting language.

### A scripting language

**A programming language designed specifically for acting on an existing entity or system.**

For a general idea of what makes a programming language a scripting language, consider the commands “walk”, “run”, and “jump.” These actions require something to carry them out, perhaps a person, a dog, or a video game character. Without an actor to perform these commands, “walk”, “run”, and “jump” wouldn’t make sense. This set of actions is analogous to a scripting language that focuses on manipulating an external entity.

### ECMAScript

**The specification defined in ECMA-262 for creating a general purpose scripting language.**  
**Synonym:**ECMAScript specification

ECMA-262 is the name of the standard, it represents the scripting language specification ECMAScript.

#### ECMAScript 6 (ES6)

**It is the sixth edition of the ECMA-262 standard, and features major changes and improvements to the ECMAScript specification.**

**Synonyms**: ES6, ES2015, and ECMAScript 2015

This edition of ECMAScript changed its name from ES6 to ES2015 because in 2015 Ecma International decided to switch to annual releases of ECMAScript. Accordingly, Ecma International also started to name new editions of the ECMAScript specification based on the year they are released. In short, ES6 and ES2015 are two different names for the same thing.

**Ref :** <https://developer.mozilla.org/en-US/docs/Glossary/ECMA>

**Ref :** <https://www.freecodecamp.org/news/whats-the-difference-between-javascript-and-ecmascript-cba48c73a2b5/>

**Advantages of ES6 modules:**

* They provide a standardized way to organize and share code between different files, making it easier to manage large codebases.
* They allow for more modular code, which can make code easier to read and maintain.
* They allow for better control over the visibility of variables and functions, reducing the risk of naming collisions.

**Disadvantages of ES6 modules:**

* They are not spported in all browesers, so developers may need to use a transpiler or polyfill to ensure compatibility with older browsers.
* They can be more difficult to understand and use than traditional script tags or other methods of including code.

In summary, hoisting is a mechanism in javascript that moves variable and function declarations to the top of their scope, while ECMAScript is a standard that defines the syntax and core features of javascript. ES6 modules are a new way of organizing and sharing code in javascript, providing more modularity and better control over the visibility of variables and functions, but may not be compatible with all browsers and can be more difficult to use than tranditional methods.

**Q. Features/ functionalities which was introduced in last 2 years and are most commonly used with example and code for it.**

ECMAScript 2021, the 12th edition, introduced the **replaceAll** method for Strings; **Promise.any**, a Promise combinator that short-circuits when an input value is fulfilled; **AggregateError**, a new Error type to represent multiple errors at once; logical assignment operators (**??=**, **&&=**, **||=**); **WeakRef**, for referring to a target object without preserving it from garbage collection, and **FinalizationRegistry**, to manage registration and unregistration of cleanup operations performed when target objects are garbage collected; separators for numeric literals (**1\_000**); and **Array.prototype.sort** was made more precise, reducing the amount of cases that result in an [implementation-defined](https://tc39.es/ecma262/multipage/overview.html#implementation-defined) [sort order](https://tc39.es/ecma262/multipage/indexed-collections.html#sort-order).

ECMAScript 2022, the 13th edition, introduced top-level **await**, allowing the [keyword](https://tc39.es/ecma262/multipage/ecmascript-language-lexical-grammar.html#sec-keywords-and-reserved-words) to be used at the top level of modules; new class elements: public and private instance fields, public and private static fields, private instance methods and accessors, and private static methods and accessors; static blocks inside classes, to perform per-class evaluation initialization; the **#x in obj** syntax, to test for presence of private fields on objects; regular expression match indices via the **/d** flag, which provides start and end indices for matched substrings; the **cause** property on **Error** objects, which can be used to record a causation chain in errors; the **at** method for Strings, Arrays, and TypedArrays, which allows relative indexing; and **Object.hasOwn**, a convenient alternative to **Object.prototype.hasOwnProperty**.

**Ref :** https://tc39.es/ecma262/multipage/#sec-intro

**Commonly used feature / functionality :**

1. **Async/ await :**

MDNweb docs: <https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Asynchronous/Async_await>

**Example Code :**

async function getData() {

const response = await fetch('https://api.example.com/data');

const data = await response.json();

return data;

}

getData().then(data => console.log(data)).catch(error => console.error(error));

1. **ES6 modules:**

**MDN web Docs:** <https://developer.mozill.org/en-US/docs/Web/JavaScript/Guide/Modules>

**Example Code :**

// Module 1

export const hello = () => console.log('Hello!');

// Module 2

import { hello } from './module1.js';

hello();

1. **Template literals :**

**MDN web docs**: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference /Template\_literals

Example code:

const name = 'John';

console.log(`Hello, ${name}!`);

1. **Arrow functions:**

**Mdn web docs** : <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Functions/Arrow_functions>

**Example Code:**

const numbers = [1, 2, 3, 4, 5];

const squares = numbers.map(number => number \* number);

console.log(squares);

1. **Spread Syntax :**

**Mdn web docs :** <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/Spread_syntax>

**Example code:**

const numbers1 = [1, 2, 3];

const numbers2 = [4, 5, 6];

const combined = [...numbers1, ...numbers2];

console.log(combined);

1. **Promises :**

**Mdn web docs :** https://developer.mozilla.org/en-US/docs/Web/JavaScript /Reference/Global\_Objects/Promise

**Example code :**

function getData() {

return new Promise((resolve, reject) => {

const request = new XMLHttpRequest();

request.open('GET', 'https://api.example.com/data');

request.onload = () => resolve(request.responseText);

request.onerror = () => reject(request.statusText);

request.send();

});

}

getData().then(data => console.log(data)).catch(error => console.error(error));