

JavaScript Introduction

Our Team

- **Carolina Acosta Acosta**
carolina.acosta.15@ull.edu.es



- **Paulo Padilla Domingues**
padilla.domingues.37@ull.edu.es



- **Laura Ramallo Pérez**
laura.ramallo.27@ull.edu.es



INDEX

01

JavaScript
Introduction

02

Code
Style

03

Google
Guide Style

04

JSDoc

05

ESLint



How to run it on the server

- **Running Node.js on a .js file in which you have written the code**

```
○ ○ ○           withFile  
$ node hello-world.js
```

- **Using Node.js in interactive mode**

```
○ ○ ○           interactiveMode  
$ node  
Welcome to Node.js v23.6.1.  
Type ".help" for more information.  
> console.log('Hello world');  
Hello world
```

The mailman has arrived

package.json

- Creation -> npm init -y

- Importance on dependencies

```
○ ○ ○          { } package.json

{
  "name": "frontend-project",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```



Out with the old, in with the new

- Directive “use strict” or ‘use strict’
- It can be enabled in a function alone
- Make sure that it’s at the top of your script so it is enabled
- Beware, there’s no going back

Differences from what we know

Semicolons and when to insert them

● What is a statement

Syntax constructs and commands that performs actions

```
○ ○ ○      JS statement.js

console.log('Hello');
console.log('World');
```

● In JavaScript we can omit a semicolon

When a line break exists JavaScript interprets it as an “implicit” semicolon

```
○ ○ ○      JS semicolons.js

console.log('Hello')
console.log('World')
```

● Not such a good idea though!

There are cases when a newline does not mean a semicolon

```
○ ○ ○      JS error.js

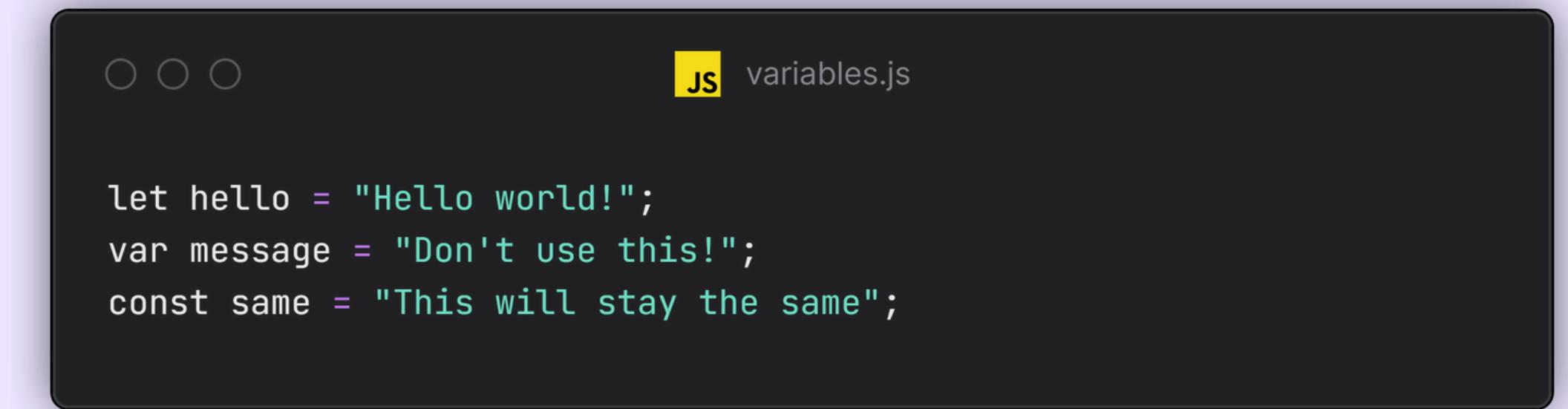
console.log("Hello");
[1, 2].forEach(console.log);

console.log("Hello")
[1, 2].forEach(console.log);
```



The art of declaring a variable

- Remember: JS is weakly and dynamically typed
- So...
- **let, var, const keywords**



variables.js

```
let hello = "Hello world!";
var message = "Don't use this!";
const same = "This will stay the same";
```

What is this?

typeof

● 7 primitive types

- Boolean
- Number
- String
- Symbol
- Null
- Undefined

● **typeof: an amazing operator...**

...or not, let's see it

No strings attached

○ ○ ○

JS strings.js

```
let name = "John";
let str = 'Single quotes are ok too';
let phrase = can embed another ${str};

// embed a variable
alert(`Hello, ${name}!`);

// embed an expression
alert(`the result is ${1 + 2}`);
```

- **Simple quotes**

- **Double quotes**

- **Backticks**

Numbers

○ ○ ○

JS nan.js

```
console.log("not a number" / 2);
```

○ ○ ○

JS propagation.js

```
console.log(NaN + 1);
console.log(3 * NaN);
console.log("not a number" / 2 - 1);
```

Functions

- **Function Declaration**
- **Function Expression**
- **What's different?**

○ ○ ○

 expression.js

```
let sayHi = function() {  
  console.log("Good morning, PAI!");  
};
```

Arrow functions

○ ○ ○

JS multiline.js

```
let sum = (a, b) => {
  let result = a + b;
  return result;
};
```

```
console.log(sum(1, 2));
```

○ ○ ○

JS sum.js

```
let sum = (num1, num2) => num1 + num2;

console.log(sum(1, 2));
```

Syntax and Formatting Rules

Curly Braces

Should be on the same line as the declaration

Incorrect example

```
○ ○ ○          JS incorrect.js
const sumDigits = function(num)
{
    let result = 0;
    while(num > 0)
    {
        result += num % 10;
        num = Math.floor(num / 10);
    }
    return result;
};
```

Correct example

```
○ ○ ○          JS correct.js
const sumDigits = function(num) {
    let result = 0;

    while (num > 0) {
        result += num % 10;
        num = Math.floor(num / 10);
    }

    return result;
};
```

When to Use Curly Braces

Incorrect use

○ ○ ○

JS incorrect.js

```
if (n < 0) {alert(`Power ${n} is not supported`);}
```

○ ○ ○

JS incorrect.js

```
if (n < 0)
    alert(`Power ${n} is not supported`);
```

When to Use Curly Braces

Correct use

○ ○ ○

JS correct.js

```
if (n < 0) alert(`Power ${n} is not supported`);
```

○ ○ ○

JS correct.js

```
if (n < 0) {  
    alert(`Power ${n} is not supported`);  
}
```

Indents

There are two types of indentations.

Horizontal indentation

Use **2 or 4 spaces** or the horizontal **tab symbol**

Vertical indentation

empty lines for splitting code into **logical blocks**.

Example

```
○ ○ ○          JS correct.js

function pow(x, n) {
  let result = 1;
  // ←
  for (let i = 0; i < n; i++) {
    result *= x;
  }
  // ←
  return result;
}
```

Nesting Levels

○ ○ ○

JS incorrect.js

```
for (let i = 0; i < 10; i++) {  
    if (cond) {  
        // ... one more nesting level  
    }  
}
```

○ ○ ○

JS correct.js

```
for (let i = 0; i < 10; i++) {  
    if (!cond) continue;  
    // ... no extra nesting level  
}
```

JS incorrect.js

```
for (let i = 0; i < 10; i++) {  
    if (i % 2 === 0) {  
        console.log(i);  
    }  
}
```

JS correct.js

```
for (let i = 0; i < 10; i += 2) {  
    console.log(i);  
}
```

Function Placement

There are three ways to organize helper functions and their usage.

Above the code.

```
○ ○ ○      JS above-the-code.js

// function declarations
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}

// the code which uses them
let elem = createElement();
setHandler(elem);
walkAround();
```

Below the code.

```
○ ○ ○      JS below-the-code.js

// the code which uses the functions
let elem = createElement();
setHandler(elem);
walkAround();

// --- helper functions ---
function createElement() {
  ...
}

function setHandler(elem) {
  ...
}

function walkAround() {
  ...
}
```

Anonymous or arrow functions

```
○ ○ ○      JS example.js

const sumar = (a, b) => a + b;
console.log(sumar(2, 3));
```

Google Style Guide

Source File Basics

File Name

- Must be in **lower case**.
- Underscores (_) or hyphens (-) may be used, with no other punctuation.
- The extension must be **.js**.

File Encoding

- All files must be **UTF-8** encoded.

```
O O O          JS example-file-name.js
'use strict';

/* Suma dos números. */
const add = (a, b) => a + b;

console.log(`Resultado: ${add(5, 10)}`);
```

When to Use Braces

Control structures (if, else, for, while, etc.) **must use** braces, even for a single instruction.



incorrect.js

```
if (isValid())
    doSomething();

for (let i = 0; i < items.length; i++)
    processItem(items[i]);
```



correct.js

```
if (isValid()) {
    doSomething();
}

for (let i = 0; i < items.length; i++) {
    processItem(items[i]);
}
```

When to Use Braces

Exception: A simple if that fits on a **single line and no else** can be omitted.

○ ○ ○

JS exception.js

```
if (isReady()) doSomething();
```

Position of the Opening Brace

According to K&R style, the opening brace { must be placed on the **same line** as the statement (e.g., if, for, function, class, etc.).

○ ○ ○ **JS** incorrect.js

```
if (condition)
{
    // block content
}
```

○ ○ ○ **JS** correct.js

```
if (condition) {
    // block content
}
```

Line Break After the Opening Brace

Once the opening brace is written, the **block's content should start on a new line**. This improves readability and clearly separates the declaration from its content.

○ ○ ○

JS incorrect.js

```
if (condition) { doSomething();  
}
```

○ ○ ○

JS correct.js

```
if (condition) {  
  doSomething();  
}
```

Line Break Before the Closing Brace

Before writing the closing brace, ensure that the **block's content ends on its own line**. This way, the closing brace is clearly separated from the block content.

○ ○ ○

JS incorrect.js

```
if (condition) {  
    doSomething(); }
```

○ ○ ○

JS correct.js

```
if (condition) {  
    doSomething();  
}
```

Line Break After the Closing Brace

Insert a line break after a **closing brace** unless it's followed by else, catch, while, or characters such as a comma, semicolon, or right parenthesis.

○ ○ ○

JS incorrect.js

```
if (condition) {  
  doSomething();  
} doAnotherThing();
```

○ ○ ○

JS correct.js

```
if (condition) {  
  doSomething();  
}  
doAnotherThing();
```

Exceptions: Cases with else, catch, etc.

Do **not insert a line break** if { is followed by **else, catch, while**, or characters such as a **comma, semicolon**, or **right parenthesis**.

```
○ ○ ○          JS incorrect.js
if (condition) {
    doSomething();
}
else {
    doSomethingElse();
}
```

```
○ ○ ○          JS correct.js
if (condition) {
    doSomething();
} else {
    doSomethingElse();
}
```

Empty Blocks May Be Concise

Close empty blocks immediately `({})` unless part of a multi-block statement like if/else or try/catch/finally.

Use an empty function as the default for a callback parameter to avoid errors if no callback is provided.



JS correct.js

```
function doNothing() {}
```



JS empty-block.js

```
function doSomething(callback = function() {}) {
  // Perform operations
  callback();
}
```

Block Indentation: 2 Spaces

Indent by **2 spaces** when opening a block; return when closing. Applies to **code** and **comments**.



incorrect.js

```
if (condition) {  
doSomething();  
}
```



correct.js

```
if (condition) {  
    // Code indented with +2 spaces  
    doSomething();  
}
```

Switch Statements

● Indentation

Switch block contents are indented +2.

● Switch labels

Add a newline and increase indentation +2, like opening a block.

● Next label

Resets indentation, as if closing a block.

● Blank line

Optional after break.

○ ○ ○ JS incorrect.js

```
switch (animal) {  
  case Animal.BANDERSNATCH:  
    handleBandersnatch();  
    break;  
  case Animal.JABBERWOCK:  
    handleJabberwock();  
    break;  
  default:  
    throw new Error('Unknown animal');  
}
```

○ ○ ○ JS correct.js

```
switch (animal) {  
  case Animal.BANDERSNATCH:  
    handleBandersnatch();  
    break;  
  
  case Animal.JABBERWOCK:  
    handleJabberwock();  
    break;  
  
  default:  
    throw new Error('Unknown animal');  
}
```

One Statement Per Line

Each statement must be on **its own line**.

○ ○ ○

JS incorrect.js

```
let a = 5; let b = 10; console.log(a + b);
```

○ ○ ○

JS correct.js

```
let a = 5;  
let b = 10;  
console.log(a + b);
```

Semicolons are Required

Each statement **must end with a semicolon.**

○ ○ ○

JS incorrect.js

```
let a = 5  
console.log(a)
```

○ ○ ○

JS correct.js

```
let a = 5;  
console.log(a);
```

Column Limit

80 Characters

When wrapping lines, every subsequent line must be indented at least **4 spaces** beyond the original line.

○ ○ ○ JS correct.js

```
const example = someFunction(  
    argument1,  
    argument2,  
    argument3  
);
```

Vertical Whitespace

✓ When to Use a Single Blank Line

- Between consecutive methods in a class or object literal.
- Within a method, to logically group statements.

Exception

- Between properties in an object literal: optional for logical grouping.

✗ When to Avoid Blank Lines

- At the start or end of a method.
- Multiple consecutive blank lines: allowed but not recommended.

```
○ ○ ○          JS incorrect.js

class MyClass {
  methodOne() {

    console.log("Start of method one");

    console.log("End of method one");

  }

  methodTwo() {

    console.log("Method two with extra blank lines");

  }
}
```

```
○ ○ ○          JS correct.js

class MyClass {
  methodOne() {
    console.log("Start of method one");
    // Logical grouping of statements without blank lines at the beginning or end
    console.log("End of method one");
  }

  methodTwo() {
    console.log("Method two without unnecessary blank lines");
  }
}
```

Horizontal Whitespace

- **Trailing Whitespace:** Forbidden

Internal Whitespace: Use exactly one ASCII space only:

- **After reserved words:** e.g., if (
- **Around binary/ternary operators:** e.g., a === b
- After commas, semicolons, and colons.
- **Before** an opening { (with exceptions)

O O O JS incorrect.js
if (a==b){doSomething();}

O O O JS correct.js
if (a == b) { doSomething(); }

Issues:

- Extra space after if
- No spaces around ===
- Missing space before {

O O O JS correct.js
function logData(data) {
 console.log(data);
}

logData({ key: "value" }); // No space before '{'
exception



Features: Local Variables

- Use only **const** and **let**.

Use **const** by default unless the variable needs to be reassigned.

Use **let** for variables whose values will change.

- **One** variable per declaration.

Avoid declaring multiple variables in the same statement.

- Declare variables as **close** as possible to their first use.



JS correct.js

```
const value1 = 10; // Use const by default
const value2 = 20;
const result = value1 + value2; // Declare variables near usage
```

Naming

General Rules for Identifiers

Use only ASCII **letters** and **digits**.

Underscores are allowed, and **\$** is permitted in **very specific cases** (e.g., for Angular).

Names must be as descriptive as possible!

○ ○ ○ **JS** incorrect.js

```
let n = 0;          // Too generic.
let nErr = 0;        // Ambiguous abbreviation.
let cstmrId = 12345; // Removed letters.
let kSecondsPerDay = 86400; // Hungarian notation is not allowed.
```

○ ○ ○ **JS** correct.js

```
let errorCode = 0;
let dnsConnectionIndex = 1;
let referrerUrl = "https://example.com";
let customerId = 12345;
```

Constant Names

Use CONSTANT_CASE for **deeply immutable global constants**, and **lowerCamelCase for local variables**, even if declared with const.

```
○ ○ ○          JS correct.js
// CONSTANT_CASE for global constants
const MAX_USERS = 100;
const HTTP_STATUS_OK = 200;

// lowerCamelCase for local variables
const result = calculateSum(10, 20);
console.log(result);
```

Class Names

Format: **UpperCamelCase**.

Names are typically **nouns** or **noun phrases** (e.g., Request, ImmutableList)

```
○ ○ ○      JS incorrect.js
class request {
    // ...
}

class immutableview {
    // ...
}
```

```
○ ○ ○      JS correct.js
class Request {
    // ...
}

class ImmutableList {
    // ...
}
```

Method Names

Format: **lowerCamelCase**.

Names should be **verbs** or **verb phrases** (e.g., `sendMessage`).

Private methods may end with an **underscore** (`_`).

```
○ ○ ○           JS incorrect.js

class Messenger {
  SendMessage(message) {
    // ...
  }

  Stop(message) {
    // ...
  }
}
```

```
○ ○ ○           JS correct.js

class Messenger {
  sendMessage(message) {
    // Send message
  }

  stop_() {
    // Private method
  }
}
```

Enum Names

Format: **UpperCamelCase** (singular).

Enum items use **CONSTANT_CASE** (uppercase with underscores).

○ ○ ○ **JS** incorrect.js

```
enum userRoles {  
  GuestUser,  
  admin_role,  
}
```

○ ○ ○ **JS** correct.js

```
enum UserRole {  
  ADMIN,  
  GUEST_USER,  
}
```

Parameters and Local Variables

Format: **lowerCamelCase**.

○ ○ ○ **JS** incorrect.js

```
function processUserData(uName, user_age) {
  let r = [];

  if (user_age > 18) {
    r.push({ name: uName, age: user_age });
  }
  return r;
}
```

○ ○ ○ **JS** correct.js

```
function processUserData(userName, userAge) {
  let activeUsers = [];

  if (userAge > 18) {
    activeUsers.push({ name: userName, age: userAge });
  }
  return activeUsers;
}
```

Single quotes “”, rather than
double quotes “” for **strings**



JSDoc

JSDoc Key Rules

- Used in **classes, methods, and fields.**
- Format:
 - Multi-line: `/** ... */`
 - Single-line: `/** ... */`
- Use Markdown for lists and text formatting.
- Allowed **tags**: `@param`, `@return`, `@private`, `@const`, etc.
- Do **not** combine **multiple tags on a single line.**
- Indentation: +4 spaces when wrapping lines.

○ ○ ○ **JS** incorrect.js

```
/** @param {number} a @param {number} b @return {number} */
function add(a, b) { return a + b; }
```

○ ○ ○ **JS** correct.js

```
/**
 * Calculates the sum of two numbers.
 * @param {number} a First number to be added.
 * @param {number} b The second number, which
 *   will be added to the first one.
 * @return {number} The total sum of `a` and `b`.
 */
function add(a, b) {
  return a + b;
}
```

ESLint JavaScript linter.

```
eslint.config.mjs
```

```
rules: {
  ...googleConfig.rules, // Reglas de Google
  'quotes': ['error', 'single'], // ✓ Fuerza comillas simples
  'semi': ['error', 'always'], // ✓ Punto y coma obligatorio
  'indent': ['error', 2, { 'SwitchCase': 1 }]
},
```

what is a Linter?

Get better habits

Correct, coherent and
consistent code

Detects possible
problems and errors



How to Use ESLint?

○ ○ ○

```
// Initialize ESLint in our project by running  
npx eslint --init
```

○ ○ ○

```
// Analyze our code  
npx eslint nombre-del-archivo.js
```

○ ○ ○

```
// Automatically fix errors  
npx eslint nombre-del-archivo.js --fix
```

ESLint Examples

1:1	error	Unexpected var, use let or const instead	no-var
1:14	error	Strings must use singlequote	quotes
1:20	error	Missing semicolon	semi
3:21	error	'city' is assigned a value but never used	no-unused-vars
5:1	error	Missing JSDoc comment	require-jsdoc
5:10	error	'greetUser' is defined but never used	no-unused-vars
5:21	error	Missing space before opening brace	space-before-blocks
6:1	error	Expected indentation of 2 spaces but found 0	indent
6:59	error	Missing semicolon	semi
9:1	error	Expected space(s) after "if"	keyword-spacing
11:7	error	'numbers' is assigned a value but never used	no-unused-vars
11:17	error	A space is required after ','	comma-spacing
11:19	error	A space is required after ','	comma-spacing
11:21	error	A space is required after ','	comma-spacing
11:23	error	A space is required after ','	comma-spacing



References

- [package.json](#)
- [JavaScript Tutorial](#)
- [Eloquent JavaScript](#)
- [Gemini \(dudas puntuales\)](#).
- [Coding Style](#)
- [Google Style Guide](#)
- [ESLint](#)
- [ChatGPT](#)

Thank You.

Get In Touch With Us



carolina.acosta.15@ull.edu.es



padilla.domingues.37@ull.edu.es



laura.ramallo.27@ull.edu.es