# 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 Style Guide

04 JSDoc

05 ESLint

# The mailman has arrived

## package.json

- Creation -> npm init
- Importance on dependencies
- Download -> npm install

{} package.json

```
{
  "name": "Práctica  XX de PAI",
  "version": "1.0.0",
  "description": "Bouncing Balls",
  "private": true,
  "main": "index.js",
  "scripts": {
    "push": "rm *~; git pull; git add . ; git commit -m 'Code
improvement'; git push",
    "pull": "git pull",
    "test": "jest",
    "lint:ci": "eslint . --ext .tsx,.ts",
    "start": "tsc-watch --onSuccess \"node dist/index.js\""
  },
  "author": "F. de Sande <fsande@ull.es>",
  "license": "MIT",
  "devDependencies": {
    "@exercism/babel-preset-typescript": "^0.1.0",
    "@exercism/eslint-config-typescript": "^0.4.1",
    "@typescript-eslint/eslint-plugin": "^5.50.0",
    "babel-jest": "^27.5.1",
    "core-js": "^3.21.0",
    "eslint": "^8.33.0",
    "eslint-config-standard-with-typescript": "^34.0.0",
    "eslint-plugin-import": "^2.27.5",
    "eslint-plugin-n": "^15.6.1",
    "eslint-plugin-promise": "^6.1.1",
    "jest": "^29.4.3",
    "tsc-watch": "^6.0.0",
    "typedoc": "^0.23.25",
    "typescript": "^4.9.5"
  }
}
```

# Key fields

- main
- scripts
  - "start"
  - “test”
  - “pull” and “push”
- dependencies
- devDependencies
  - Compilación
  - Pruebas
  - Calidad de código
  - Documentación

# README

- **Markdown**

README.md

```
# Project X
This is an amazing project.

## Installation
```sh
npm install
```

- **ReStructuredText**

README.rst

```
Project X
==========

This is an amazing project.

Installation
------------
 .. code-block:: bash

pip install package
```

- **HTML**

README.rst

```
<h1>Project X</h1>
<p>This is an amazing project.</p>
<h2>Installation</h2>
<pre><code>npm install</code></pre>
```
```

# **Out with the old,** in with the new

- **Directive ‘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 contructs and commands that performs actions

- **In JavaScript we can omit a semicolon**

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

- **Not such a good idea though!**

  There are cases when a newline does not mean a semicolon

statement.js

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

semicolons.js

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

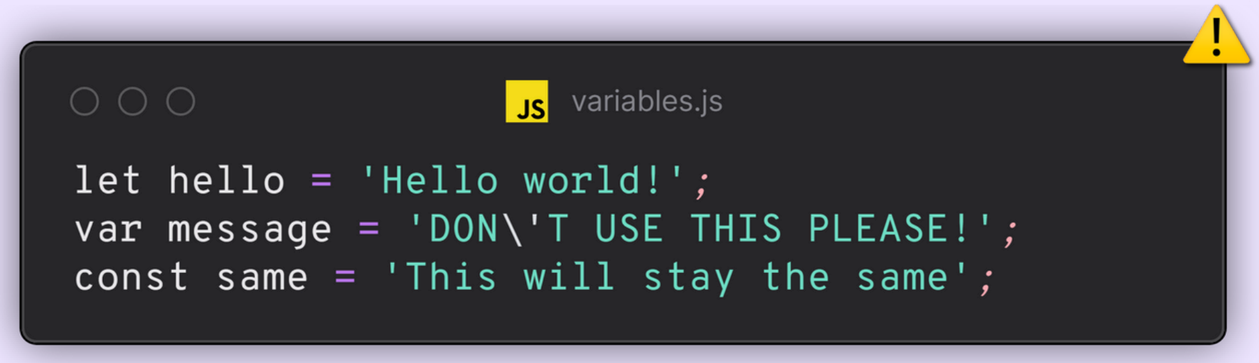
error.js

```
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**



# Conversions

- **Numbers**

conversions1.js

```
console.log(Number('123')); // 123
console.log(Number('123z')); // NaN (error reading a number at "z")
console.log(Number(true)); // 1
console.log(Number(false)); // 0
```

- **Booleans**

- **Careful with ‘0’**

conversions2.js

```
console.log(Boolean(1)); // true
console.log(Boolean(0)); // false

console.log(Boolean('hello')); // true
console.log(Boolean('')); // false

console.log(Boolean('0')); // true
console.log(Boolean(' ')); // spaces, also true
```

# Object Wrapper

wrapper.js

```
let pi = 3.1415;
console.log(pi.toFixed(2)); // '3.14'
```

- **Autoboxing**
- **Container that wraps a primitive numeric value**

# Comparisons

## == or ===

comparisons.js

```
console.log(0 == false); // true
console.log('' == false); // true

console.log(0 === false); // false
```

- In == there's a conversion
- In === no conversion, false if it's a differente type

# Nullish coalescing operator ‘??’

- Returns firstValue if it’s not null or undefined
- Returns secondValue otherwise
- To put default values

nullish.js

```
result = firstValue ?? secondValue
result = (firstValue !== null && firstValue !== undefined) ? firstValue : secondValue;
```

default.js

```
let firstName;
let lastName;
let nickName = 'Supercoder';

/** shows the first defined value: */
console.log(firstName ?? lastName ?? nickName ?? 'Anonymous');
```

# No strings attached

strings.js

```
// Single quotes
const singleQuotes = 'This is a string with single quotes';
// Double quotes
const doubleQuotes = "This is a string with double quotes";

// Backticks (Template literals)
const userName = 'John';
const backTicks = `Hello, my name is ${userName}.`;

// Examples of usage
console.log(singleQuotes);
console.log(doubleQuotes);
console.log(backTicks);
```

- Simple quotes
- Double quotes
- Backticks

splice.js

```
const sentence = ['I', 'study', 'JavaScript'];
sentence.splice(1, 1, 'love'); // Removes 'study', inserts 'love'
console.log(sentence); // Output: ['I', 'love', 'JavaScript']
```

map.js

```
const names = ['Bilbo', 'Gandalf', 'Nazgul'];
const nameLengths = names.map((name) => name.length);
console.log(nameLengths); // Output: [5, 7, 6]
```

splice.js

```
const numbers = [1, 2, 5];
numbers.splice(-1, 0, 3, 4); // Inserts 3 and 4
console.log(numbers); // Output: [1, 2, 3, 4, 5]
```

reduce.js

```
const sum = [1, 2, 3, 4, 5].reduce((total, num) => total + num, 0);
console.log(sum); // Output: 15
```

# **Essential** Array Methods

**Manipulation:**

- Add/Remove items: push(), pop(), splice()

**Searching:**

- Find elements: indexOf(), includes(), find(), filter()

**Transformation & Sorting:**

- Modify items: map()
- Sort/Reverse: sort(), reverse()

**Reduction:**

- Aggregate values: reduce()

map.js

```
const myMap = new Map();
myMap.set('first key', 1)
     .set(1, 'one')
     .set(true, 'bool');

console.log(myMap.get(1)); // 'one'
console.log(myMap.get('first key')); // 1
console.log(myMap.size); // 3
```

set.js

```
const mySet = new Set();
mySet.add(1);
mySet.add(2);
mySet.add(1); // Will not be added

console.log(mySet.size); // 2

for (const value of mySet) {
  console.log(value); // Prints 1 and then 2
}
```

# Map and Set

- **Map**
  - **Keys** can be of **any type**.
  - Preserves **insertion order**.
  - Allows **chaining calls** with .set().
- **Set**
  - Collection of unique values.
  - Preserves **insertion order**.

# Functions

- Function Declaration
- Function Expression
- What’s different?

expression.js

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

# Arrow functions

sum.js

```
const calculateSum = (firstAddend, secondAddend) => firstAddend + secondAddend;

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

multiline.js

```
const calculateSum = (firstAddend, secondAddend) => {
  const totalSum = firstAddend + secondAddend;
  return totalSum;
};

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

# **Syntax and** Formatting Rules

# Curly Braces

# Brace Placement Styles in JavaScript

## Most Common Convention

most-common.js

```
const sumOfDigits = function(number) {
  let totalSum = 0;
  while (number > 0) {
    const lastDigit = number % 10;
    totalSum += lastDigit;
    number = Math.floor(number / 10);
  }
  return totalSum;
};
```

## Less Common

less-common.js

```
const sumOfDigits = function(number)
{
  let totalSum = 0;
  while (number > 0)
  {
    const lastDigit = number % 10;
    totalSum += lastDigit;
    number = Math.floor(number / 10);
  }
  return totalSum;
};
```

# **Obligatory Inclusion** in Structures of Control

Always include braces, even for **single-line blocks**, to **avoid future errors** when adding more sentences.

JS correct.js

```
if (number > 0) {
  console.log('number is positive');
  console.log('This line executes only when number > 0');
}
```

JS incorrect.js

```
if (number > 0)
  console.log('number is positive');
  console.log('This line executes regardless of the condition');
```

# Efficient Arrow Function Syntax

❌

incorrect.js

```
const add = (addend1, addend2) => {
  addend1 + addend2;
};
```

UNDEFINED

✅

correct.js

```
const add = (addend1, addend2) => {
  return addend1 + addend2;
};
```

correct.js

```
const add = (addend1, addend2) => addend1 + addend2;
```

# Keep Your Functions **Short and Focused**

Martin, R. C. *Clean Code: A Handbook of Agile Software Craftsmanship.* Prentice Hall, 2009.


- An ideal function should **not take up more than one screen**.
- **Single Responsibility:** Each function should perform one clear task or responsibility.

JS incorrect.js

```
function processUserData(user) {
  // Validate input
  if (!user.name || !user.email) {
    throw new Error('Invalid user data');
  }

  // Process user data
  const formattedName = user.name.trim().toUpperCase();
  const emailDomain = user.email.split('@')[1];

  // Log the result
  console.log(`User ${formattedName} uses email from ${emailDomain}`);

  // Additional processing could go here ...
}
```

JS correct.js

```
function validateUserData(user) {
  if (!user.name || !user.email) {
    throw new Error('Invalid user data');
  }
}

function formatUserName(name) {
  return name.trim().toUpperCase();
}

function getEmailDomain(email) {
  return email.split('@')[1];
}

function logUserInfo(name, domain) {
  console.log(`User ${name} uses email from ${domain}`);
}

function processUserData(user) {
  validateUserData(user);
  const formattedName = formatUserName(user.name);
  const domain = getEmailDomain(user.email);
  logUserInfo(formattedName, domain);
}
```

# Encapsulation, Entry Point and Use of Global Variables

- No Global Pollution.
- **Modularity** & Reusability.
- **Centralized** Execution.

incorrect.js

```
// Code and variables executed directly in the global scope

let result = 0;

function calculateSum(firstAddend, secondAddend) {
  result = firstAddend + secondAddend;
}

calculateSum(2, 3);
console.log('Sum is:', result);
```

correct.js

```
// Functions defined in the global scope without direct execution

function calculateSum(firstAddend, secondAddend) {
  return firstAddend + secondAddend;
}

function main() {
  const result = calculateSum(2, 3);
  console.log('Sum is:', result);
}

if (require.main === module) {
  main();
}
```

# Direct Boolean Returns: Conciseness and Clarity

not-recommended.js

```
function isMinor(age) {
  if (age < 18) {
    return true;
  } else {
    return false;
  }
}
```

recommended.js

```
function isAdult(age) {
  return age >= 18;
}
```

✅

# Indents

# **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

horizontal-indentation.js

```
function calculatePower(base, exponent) {
••let powerResult = 1;
  // ←
••for (let i = 0; i < exponent; i++) {
••••powerResult *= base;
  }
  // ←
••return powerResult;
}
```

# Nesting Levels

incorrect.js

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

correct.js

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

incorrect.js

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

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.

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.

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

example_filename.js

```
const add = (firstAddend, secondAddend) =>
    firstAddend + secondAddend;

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

# 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.

example_filename.js

```
'use strict';

/* Adds two numbers. */
const add = (firstAddend, secondAddend) => firstAddend + secondAddend;

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.

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.).

incorrect.js

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

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.

incorrect.js

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

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.

incorrect.js

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

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** ({}).

correct.js

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

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

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**.

JS incorrect.js

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

JS correct.js

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

⚠️

// Always leave a **space after //** for better readability.

# Switch Statements

- **Indetation**

  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.

incorrect.js

```
const animal = 'cat';

switch (animal) {
case 'dog':
console.log('Woof! It\'s a dog.');
break;
case 'cat':
console.log('Meow! It\'s a cat.');
break;
default:
console.log('Unknown animal.');
break;
}
```

correct.js

```
const animal = 'cat';

switch (animal) {
  case 'dog':
    console.log('Woof! It\'s a dog.');
    break;

  case 'cat':
    console.log('Meow! It\'s a cat.');
    break;

  default:
    console.log('Unknown animal.');
    break;
}
```

# One Statement Per Line

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

incorrect.js

```
const firstNumber = 5; const secondNumber = 10; console.log(firstNumber + secondNumber);
```

correct.js

```
const firstNumber = 5;
const secondNumber = 10;
console.log(firstNumber + secondNumber);
```

# Semicolons are Required

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

incorrect.js

```
const userAge = 22
console.log(`The age of the user is ${userAge}`)
```

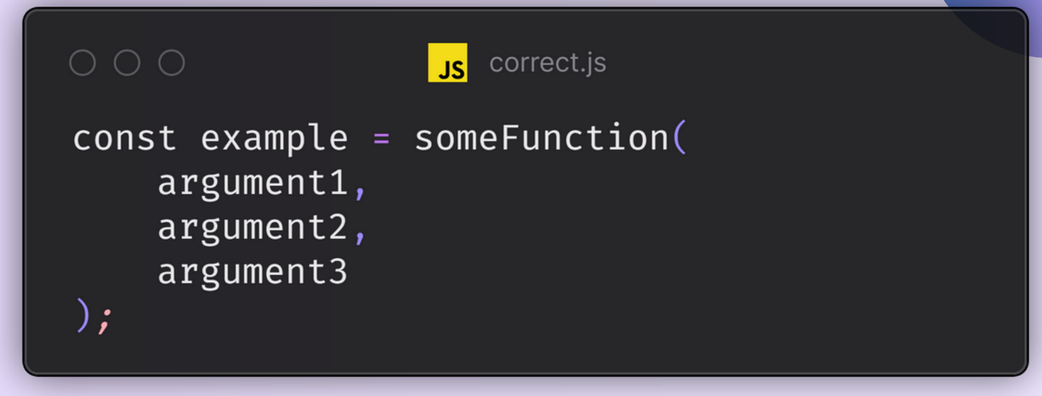
correct.js

```
const userAge = 22;
console.log(`The age of the user is ${userAge}`);
```

# Column Limit

**80 Characters**

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



# 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.');

  }
}
```

JS correct.js

```
function calculateSum() {
  const firstNumber = 5;
  const secondNumber = 10;

  console.log(firstNumber + secondNumber);
}
```

JS correct.js

```
const config = {
  host: 'localhost',

  // Database related properties
  databaseUser: 'user',
  databasePassword: 'secret',

  port: 8080,
};
```

# 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)

incorrect.js

```
if   (a===b){doSomething();
```

correct.js

```
if (a === b) { doSomething(); }
```

correct.js

```
function logData(data) {
  console.log(data);
}

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

*exception*

*Issues:*

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

# **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).

😈 ⚠️

Give as **descriptive a name as possible**, within reason. **Do not worry about saving horizontal space** as it is far more important to make your code immediately understandable by a new reader.

JS incorrect.js

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

JS correct.js

```
const errorCount = 0;
const dnsConnectionIndex = 1;
const referrerUrl = 'https://example.com';
const customerId = 12345;
```

# Constant Names

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

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, ImmutableView)

incorrect.js

```
class request {
  // ...
}

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

correct.js

```
class Request {
  // ...
}

class ImmutableView {
  // ...
}
```

# **Method** Names

Format: **lowerCamelCase**.

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

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

incorrect.js

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

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

correct.js

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

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

# **Enum** Names

Format: **UpperCamelCase** (singular).

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

incorrect.js

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

correct.js

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

# **Parameters and** Local Variables

Format: **lowerCamelCase**.

incorrect.js

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

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

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.

incorrect.js

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

correct.js

```
/**
 * Adds two numbers together.
 * @param {number} firstAddend - The first number to add.
 * @param {number} secondAddend - The second number to add.
 * @return {number} The sum of the two numbers.
 */
function add(firstAddend, secondAddend) {
  return firstAddend + secondAddend;
}
```

# ESLint

## JavaScript linter

- ESLint.
- JSHint.
- StandardJS.
- XO.

# What is a Linter?

**Get better habits**

**Detects possible problems and errors**

**Correct, coherent and consistent code**

```
$ npx eslint --init
```

01

1. **Initialize** ESLint.
2. **Install** Dependencies.
3. Edit **eslint.config.mjs**.
4. Install the **extension in VS Code.**
5. (Optional) Enable **auto-fix on save**.

02

```
$ npm install eslint eslint-config-google @eslint/js globals --save-dev
```

03

eslint.config.mjs

```
import globals from 'globals';
import pluginJs from '@eslint/js';
import googleConfig from 'eslint-config-google';

/** @type {import('eslint').Linter.Config[]} */
export default [
  {
    files: ['**/*.js'],
    languageOptions: {
      sourceType: 'module',
      globals: {
        ...globals.node,
      },
    },
    rules: {
      // Filtra las reglas de Google para excluir "valid-jsdoc" y "require-jsdoc"
      ...Object.fromEntries(
        Object.entries(googleConfig.rules).filter(
          ([key]) => key !== 'valid-jsdoc' && key !== 'require-jsdoc'
        )
      ),
      'quotes': ['error', 'single'], // ✅ Fuerza comillas simples
      'semi': ['error', 'always'],   // ✅ Punto y coma obligatorio
      'max-len': ['off'],            // ✅ Desactiva límite de línea
      'indent': ['error', 2, { 'SwitchCase': 1 }] // ✅ Indentación de 2 espacios
    },
  },
  pluginJs.configs.recommended, // Configuración base de ESLint
];
```

05

settings.json

```
"editor.codeActionsOnSave": {
  "source.fixAll.eslint": "always"
}
```

# 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
```

**Reference for installation** here

# 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
```

**Reference for installation** here


# References


- package.json
- JavaScript Tutorial
- Eloquent JavaScript
- Gemini (dudas puntuales)
- Coding Style
- Google Style Guide
- ESLint
- ChatGPT
- Clean Code

# 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