

JavaScript Introduction +

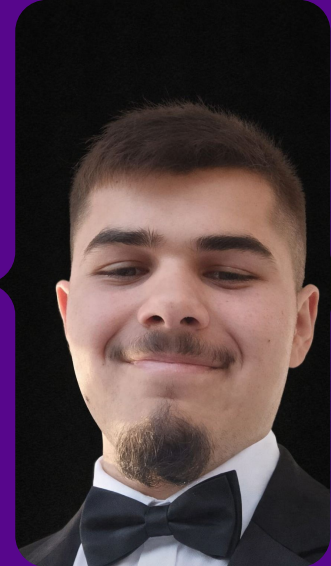
Coding Style + JSDoc + ESLint

First, manners, this is our team



Daniel Martínez
Sopo
[alu0101675893](#)

Marco Aguiar
Álvarez
[alu0101620961](#)



Diego García
Hernández
[alu0101633732](#)

What we will **MISS**

- Basic coding structures (while, if, switch...)
- Details about software installation
- OOP
- Most content already available in the virtual classroom presentations.



JavaScript Introduction

Operators + Behaviours + Functions +
Jutge tips

JS - Stack functionality

- Should be trivial...



JS - Stack functionality

- Should be trivial...

```
function greet(who) {  
    console.log('Hi ' + who);  
}  
greet('Mawi');  
console.log('Bye');
```



JS - Stack functionality

- Should be trivial...

```
function greet(who) {  
    console.log('Hi ' + who);  
}  
greet('Mawi');  
console.log('Bye');
```

[Stack.js](#)



JS - Paradox...?

- What came first?



JS - Paradox...?

- What came first?
- The chicken or the egg?



JS - Paradox...?

- What came first?
- The chicken or the egg?

Let's check



JS - Declarative Functions

- Doesn't matter where its defined.



JS - Declarative Functions

- Doesn't matter where its defined.

```
console.log('Future says:', future());  
  
function future() {  
    return 'You will pass PAI';  
}
```



JS - Declarative Functions

- Doesn't matter where its defined.

```
console.log('Future says:', future());  
  
function future() {  
    return 'You will pass PAI';  
}
```

Go ahead and execute it



JS - No return?

- What does a “returnless” function return?



JS - No return?

- What does a “returnless” function return?

```
function noReturn () {  
  let elcheTeam = 'I am not doing anything';  
}
```



JS - No return?

- And an empty return?



JS - No return?

- And an empty return?

```
function emptyReturn () {  
  let oviedoTeam = 'Me neither';  
  return  
}
```



JS - No return?

- And an empty return?

```
function emptyReturn () {  
  let oviedoTeam = 'Me neither';  
  return  
}
```

Did you hit or miss?



JS - Arrow Functions

- An elegant alternative



JS - Arrow Functions

- An elegant alternative
- Flexible syntax



JS - Arrow Functions

- An elegant alternative
- Flexible syntax

```
const isEven = n => {  
  if (n % 2 === 0) return true;  
  return false;  
};
```



JS - Arrow Functions

- An elegant alternative
- Flexible syntax

```
const isEven = n => {  
  if (n % 2 === 0) return true;  
  return false;  
};
```

[More syntax examples](#)



JS - Arrow Functions

- Looks familiar...?



JS - Arrow Functions

- Looks familiar...?

```
let doubled = input.map(x => x * 2);
```



JS - Arrow Functions

- Looks familiar...?

```
let doubled = input.map(x => x * 2);
```

.map() & .forEach()



JS - Closure

- Function that remembers the scope



JS - Closure

- Function that remembers the scope

```
function multiplier(factor) {  
  return number => number * factor;  
}
```



JS - Closure

- Function that remembers the scope

```
function multiplier(factor) {  
  return number => number * factor;  
}
```

Let's check how it works



JS - VAR

- **ONLY DIDACTIC PURPOSES**



JS - VAR

- **ONLY DIDACTIC PURPOSES**
- Special scope behaviour



JS - VAR

- **ONLY DIDACTIC PURPOSES**
- Special scope behaviour
- Global or function scope



JS - VAR

- **ONLY DIDACTIC PURPOSES**
- Special scope behaviour
- Global or function scope

```
const GLOBALNUM = 10;    // global

if (true) {
    let localNum = 20;    // local
    var globalVar = 30;   // global
}
```



JS - VAR

```
function dummy () {  
  var invisible = 40; // only in function  
  console.log(invisible);  
}  
  
console.log(invisible); // not visible
```



JS - VAR

```
function dummy () {  
    var invisible = 40; // only in function  
    console.log(invisible);  
}  
  
console.log(invisible); // not visible
```

Var file



JS - File System

- Standard module of [Node.js](#)



JS - File System

- Standard module of [Node.js](#)
- Focus on Jutge



JS - File System

- Standard module of [Node.js](#)
- Focus on Jutge
- For more information check:



JS - File System

- Standard module of [Node.js](#)
- Focus on Jutge
- For more information check:

[File System](#)



JS - File System

- Useful functions of the File System:



JS - File System

- Useful functions of the File System:
- `.trim()`



JS - File System

- Useful functions of the File System:
- `.trim()`
- `.split()`



JS - File System

- Useful functions of the File System:
- `.trim()`
- `.split()`
- `.map()`



JS - File System

- Useful functions of the File System:
- `.trim()`
- `.split()`
- `.map()`

Check the examples: [The value of .trim\(\)](#)



JS - Spread Operator “...”

- Sometimes useful for Arrays



JS - Spread Operator “...”

- Sometimes useful for Arrays
- Creates modified copy, doesn't alter original



JS - Spread Operator “...”

- Sometimes useful for Arrays
- Creates modified copy, doesn't alter original

```
const a = [1, 2, 3];  
const b = [...a, 4, 5];  
  
console.log(b);
```



JS - Spread Operator “...”

- Sometimes useful for Arrays
- Creates modified copy, doesn't alter original

```
const a = [1, 2, 3];  
const b = [...a, 4, 5];  
  
console.log(b); // [1, 2, 3, 4, 5]
```



JS - Spread Operator “...”

- Sometimes useful for Arrays
- Creates modified copy, doesn't alter original

```
const a = [1, 2, 3];  
const b = [...a, 4, 5];  
  
console.log(b); // [1, 2, 3, 4, 5]
```

[Check an interesting use](#)



JS - Arguments

- Very friendly syntax



JS - Arguments

- Very friendly syntax

- Just: `process.argv`



JS - Arguments

- Very friendly syntax

- Just: `process.argv`

[Experiment with this example](#)



JS - Distinct “for” usage

- Which gets the value?



JS - Distinct “for” usage

- Which gets the value?

```
for (let member in list) {  
    console.log(member);  
}
```

```
for (let member of list) {  
    console.log(member);  
}
```



JS - Distinct “for” usage

- Which gets the value?

```
for (let member in list) {  
  console.log(member);  
}
```

```
for (let member of list) {  
  console.log(member);  
}
```

[Here is the answer](#)



JS - operator ??

```
console.log( 0 || 100 );  
// → 100
```



JS - operator ??

```
console.log( 0 || 100 );
```

```
// → 100
```

```
console.log( 0 ?? 100 );
```

```
// → 0
```



JS - operator ??

```
console.log( 0 || 100 );
```

```
// → 100
```

```
console.log( 0 ?? 100 );
```

```
// → 0
```

```
console.log( null ?? 100 );
```

```
// → 100
```



JS - operator ??

```
console.log( 0 || 100 );
```

```
// → 100
```

```
console.log( 0 ?? 100 );
```

```
// → 0
```

```
console.log( null ?? 100 );
```

```
// → 100
```

```
console.log( undefined ?? 100 );
```

```
// → 100
```



JS - operator ??

```
console.log( 0 || 100 );  
// → 100  
console.log( 0 ?? 100 );  
// → 0  
console.log( null ?? 100 );  
// → 100  
console.log( undefined ?? 100 );  
// → 100
```

[operator_interrogation.js](#)



Google Style Guide

Best Practices

Style guide: index

- Statements
- Braces
- Indentation
- White spaces



Statements

- Only one statement per line.



Statements

- Only **one statement** per line.

```
let numero = '5';  
let otroNumero = 11;
```



Statements

- Only **one statement** per line.

```
let numero = '5';  
let otroNumero = 11;
```



```
let numero = '5', let otroNumero = 11;
```



Statements: semicolons ;

- They are **MANDATORY**



Statements: semicolons ;

- They are **MANDATORY**
- Relying on *Automatic Semicolon Insertion* is **FORBIDDEN**



Statements: semicolons ;

- They are **MANDATORY**
- Relying on *Automatic Semicolon Insertion* is **FORBIDDEN**

[Semicolons-example.js](#)



Statements - Column limit

- JavaScript has a column limit of 80 characters



Statements - Column limit

- JavaScript has a column limit of 80 characters
- If this is exceeded, **line wrapping** must be performed



Statements - Column limit

- No single formula or rule exists.



Statements - Column limit

- No single formula or rule exists.
- Same code, multiple valid ways to style it.



Statements - Column limit

- No single formula or rule exists.
- Same code, multiple valid ways to style it.

[Check some examples!](#)



Braces

- Braces { } are **MANDATORY**



Braces

- Braces { } are **MANDATORY**
- Applies to if, else, do, while, for, ...



Braces

- Braces { } are **MANDATORY**
- Applies to if, else, do, while, for, ...
- **Sole Exception:** Single-line if (no else) for readability.



Braces - bad examples

```
if (someVeryLongCondition())  
    doSomething();
```



```
for (let i = 0; i < 10; i++) bar(foo[i]);
```

Braces - bad examples

```
if (someVeryLongCondition())  
    doSomething();
```



```
for (let i = 0; i < 10; i++) bar(foo[i]);
```

- **Exception:**

```
if (shortCondition()) foo();
```



Braces - bad examples

```
if (someVeryLongCondition())  
    doSomething();
```



```
for (let i = 0; i < 10; i++) bar(foo[i]);
```

- **Exception:**

```
if (shortCondition()) foo();
```

[The value of braces.js](#)



Braces - K&R style

- **Kernighan and Ritchie style** for nonempty blocks and block-like constructs



Braces - K&R style

- **Kernighan and Ritchie style** for nonempty blocks and block-like constructs

[Check here the rules ;\)](#)



Indentation

- The body of a function is **always** indent +2 spaces inward.



Indentation

- The body of a function is **always** indent +2 spaces inward.
- Indentation can be relative to:



Indentation

- The body of a function is **always** indent +2 spaces inward.
- Indentation can be relative to:
 - The statement prefix (start of line).



Indentation

- The body of a function is **always** indent +2 spaces inward.
- Indentation can be relative to:
 - The statement prefix (start of line).
 - The function call (visual alignment).



Indentation

- The body of a function is **always** indent +2 spaces inward.
- Indentation can be relative to:
 - The statement prefix (start of line).
 - The function call (visual alignment).
 - [function_indentation.js](#)



Indentation - arrays

- Which of these is correct?



Indentation - arrays

- Which of these is correct?

```
const a = [  
  0,  
  1,  
  2,  
];
```



Indentation - arrays

- Which of these is correct?

```
const a = [  
  0,  
  1,  
  2,  
];
```

```
const b =  
  [0, 1, 2];
```



Indentation - arrays

- Which of these is correct?

```
const a = [  
  0,  
  1,  
  2,  
];
```

```
const b =  
  [0, 1, 2];
```

```
const c = [0, 1, 2];
```



White Spaces - vertical

- **MANDATORY:** Between class methods.
- **Optional:** In object literals (to group fields).



White Spaces - vertical

- **MANDATORY:** Between class methods.
- **Optional:** In object literals (to group fields).
- **Inside methods:** Use to separate logical steps.
- **FORBIDDEN:** At start or end of function body.



White Spaces - vertical

- **MANDATORY:** Between class methods.
- **Optional:** In object literals (to group fields).
- **Inside methods:** Use to separate logical steps.
- **FORBIDDEN:** At start or end of function body.

hey you, click me!



White Spaces - horizontal

- **Leading:** Start of line (Indentation).



White Spaces - horizontal

- **Leading:** Start of line (Indentation).
- **Internal:** Inside lines (Formatting).



White Spaces - horizontal

- **Leading:** Start of line (Indentation).
- **Internal:** Inside lines (Formatting).
- **Trailing:** End of line. **STRICTLY FORBIDDEN.**



White Spaces - horizontal MUST do

- **Space** after if, else, catch **before** (.



White Spaces - horizontal MUST do

- **Space** after if, else, catch **before** (.
- **Exception:** No space for function or super.



White Spaces - horizontal MUST do

- **Space** after if, else, catch **before** (.
- **Exception:** No space for function or super.
- **Space before** { (except objects as arguments).



White Spaces - horizontal MUST do

- **Space after commas, semicolons, and colons.**



White Spaces - horizontal MUST do

- Space after commas, semicolons, and colons.
- **NEVER** before punctuation.



White Spaces - horizontal MUST do

- Space after commas, semicolons, and colons.
- **NEVER** before punctuation.
- Surround ternary operators.



Final boss guide style

- Let's analyze this code:
 - [incorrect-guide-style.js](#)



Final boss guide style

- Let's analyze this code:
 - [incorrect-guide-style.js](#)
- Here is the correct version:
 - [correct-guide-style.js](#)



JSDoc

comments + tags

JSDoc

- API documentation generator for JavaScript



JSDoc

- API documentation generator for JavaScript
- Similar to Doxygen



JSDoc

- API documentation generator for JavaScript
- Similar to Doxygen
- `//, /** */`



JSDoc

- @desc
- @author
- @param
- @return
- @link



JSDoc

- @desc
- @author
- @param
- @return
- @link

JSDoc



JSDoc

```
/**
 * @desc This function prints
 * 'Hello, World'
 * @return 'Hello, World'
 *
 */
function foo() {
    return 'Hello, World!';
}
```



JSDoc

```
/**
 * @desc This function prints
 * 'Hello, World'
 * @return 'Hello, World'
 *
 */
function foo() {
    return 'Hello, World!';
}
```

[JSDoc-style.js](#)



ESLint

Linters + AutoFix + ESLint for TS

Lint

- Improve source code



Lint

- Improve source code
- Origin in C



Lint

- Improve source code
- Origin in C
- “Remove the lint”



ESLint

- JavaScript Linter



ESLint

- JavaScript Linter
- Display errors



ESLint

- JavaScript Linter
- Display errors
- 3 parts



ESLint

- JavaScript Linter
- Display errors
- 3 parts
 - Praser



ESLint

- JavaScript Linter
- Display errors
- 3 parts
 - Praser
 - Rules



ESLint

- JavaScript Linter
- Display errors
- 3 parts
 - Praser
 - Rules
 - Result



ESLint

- Install
 - \$ npm install -D eslint



ESLint

```
import globals from "globals";  
import pluginJs from "@eslint/js";
```

```
export default [  
  { languageOptions: { globals: globals.node }  
},  
  pluginJs.configs.recommended,  
];
```



ESLint

- Interesting URL:
 - [ESLint_rules](#)



ESLint

- Interesting rules



ESLint

- Interesting rules
 - no-seft-compare



ESLint

- Interesting rules
 - no-seft-compare
 - no-unreachable



ESLint - Google Style Guide

- Dependencies
 - `$ npm install -D eslint eslint-config-google @eslint/eslintrc globals`



ESLint - Google Style Guide

- Dependencies
 - \$ npm install -D eslint eslint-config-google @eslint/eslintrc globals

[Google Style Guide config](#)



ESLint - AutoFix

- AutoFix:
 - `$ npx eslint --fix`



ESLint

- Example:
 - [ESLint_example.js](#)



ESLint for TypeScript

- Install



ESLint for TypeScript

- Install
 - `$ npm install --save-dev eslint @typescript-eslint/parser`



ESLint for TypeScript

- Install
 - \$ npm install --save-dev eslint @typescript-eslint/parser
 - \$ touch .eslintrc



ESLint for TypeScript

```
{  
  "root": true,  
  "parser": "@typescript-eslint/parser",  
  "plugins": [ "@typescript-eslint" ],  
  "extends": [  
    "eslint:recommended",  
    "plugin:@typescript-eslint/eslint-recommended",  
    "plugin:@typescript-eslint/recommended"  
  ]  
}
```



ESLint for TypeScript

- `touch .eslintignore`



ESLint for TypeScript

- touch .eslintignore
 - /build
 - /students



ESLint for TypeScript

- touch .eslintignore
 - /build
 - /students
- npm run lint



ESLint for TypeScript

- Rules for TypeScript:
 - [Overview | typescript-eslint](#)



Thanks for the attention
Doubts, questions?



References

- [Coding Style](#)
- [Google JavaScript Style Guide](#)
- [JSDoc](#)
- [ESLint: Linter Javascript - Javascript en español](#)
- [Pasos para instalar y configurar ESLint en nuestro proyecto con Typescript](#)



References

- [Eloquent JavaScript](#) (chapter 1 - 4)
- [The Modern JavaScript Tutorial](#)
- [Presentations-Tips.ppt](#)
- [Oral-Presentations-Tips](#)

