Universidad de La Laguna

Good programming practices in JavaScript

Adrián Emilio Padilla Rojas
alu0101138558@ull.edu.es

Antonio Raúl Guijarro Contreras
alu0101100494@ull.edu.es

Topics to talk about

- Google and JavaScript Style Guide
  - Source file basics
  - Formatting
  - Language features
  - Naming
- JSDOC
  - Basic concepts
  - Installation process
  - In the practice
- Linters
  - ESlint
  - Setup ESlint & how to use it

Universidad
de La Laguna

Google and JavaScript Style Guide

Source file basics

File name

- All in lowercase.
- Posible use: _ or -
- File names extension must be .js
- All files are encoded in UTF-8.

File name example

Good:

- hello_world.js
- hello-world.js
- helloworld.js

Bad:

- hello_3.js
- hello,world.js
- HeLlO_WoRlD.js

Special characters

Whitespace characters

- All other whitespace characters in string literals are escaped.
- **Not use** tab for indentation.

Non-ASCII characters

- Code easier to read and understand.
- An explanatory comment can be very helpful.

Non-ASCII characters example

```
/* Best: perfectly clear even without a comment. */
const units = 'μs';

/* Allowed: but unnecessary as μ is a printable
character. */
const units = '\u03bcs'; // 'μs'

/* Good: use escapes for non-printable characters with a
comment for clarity. */
return '\ufeff' + content;  // Prepend a byte order mark.
```

Formatting

Braces are used for all control structures (1)

- Bad use :

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

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

Formatting

Braces are used for all control structures (2)

- But exist a exception.

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

Formatting

Nonempty blocks: K&R style

```
class InnerClass {
  constructor() {}

  /** @param {number} foo */
  method(foo) {
    if (condition(foo)) {
      try {
        // Note: this might fail.
        something();
      } catch (err) {
        recover();
      }
    }
  }
}
```

Formatting

Empty blocks: may be concise

- Good vs bad use:

```
function doNothing() {} // Okay

if (condition) { // Bad use
  // ...
} else if (otherCondition) {} else {
  // ...
}

try {
  // ...
} catch (e) {}
```

Formatting

Array and objects: optionally block-like (1)

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

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

```
const a = {
  a: 0,
  b: 1,
};

const b =
    {a: 0, b: 1};
```

Formatting

Function expressions

```
some.reallyLongFunctionCall(arg1, arg2, arg3)
    .thatsWrapped()
    .then((result) => {
      // Indent the function body +2 relative to the indentation depth
      // of the '.then()' call.
      if (result) {
        result.use();
      }
    });
```

Formatting

Switch statements.

```
switch (animal) {
  case Animal.BANDERSNATCH:
    handleBandersnatch();
    break;

  case Animal.JABBERWOCK:
    handleJabberwock();
    break;

  default:
    throw new Error('Unknown animal');
}
```

Statements

Examples:

```
currentEstimate =
    calc(currentEstimate + x * currentEstimate) /
        2.0;
```

```
currentEstimate = calc(currentEstimate + x *
    currentEstimate) / 2.0;
```

Whitespaces

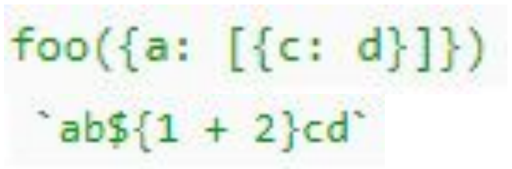
Vertical Whitespaces

- Between consecutive methods in a class or object.
- At the start or end of a function body are not allowed.
- Before the first or after the last method in a class or object (optional).

Whitespaces

Horizontal Whitespaces (1)

- Separating any reserved word (except for function and super), from an open parenthesis.
- Separating any reserved word, from a brace.
- Before any open curly brace, exceptly:
  - Before an object literal.
  - In a template expansion.



```
foo({a: [{c: d}]})
`ab${1 + 2}cd`
```

Whitespaces

Horizontal Whitespaces (2)

- On both sides of any binary or ternary operator.
- After a comma, semicolon or colon.
- On both sides of the double slash.
- After an open-block comment character, example:

```
this.foo = /** @type {number} */ (bar) ; or
```

```
function(/** string */ foo) { ; or baz(/** buzz= */ true)}
```

Whitespaces

Horizontal Whitespaces: discouraged

```
{
  tiny: 42, // this is great
  longer: 435, // this too
};

{
  tiny:   42,  // permitted, but future edits
  longer: 435, // may leave it unaligned
};
```

Grouping parentheses: recommended

```
if (2 * 3 > 2  * 3 + 1 && !istrue || 9 * 3 === 3 * 9)
```

```
if ((2 * 3 > (2 * 3) + 1) && !(istrue) || (9 * 3 === 3 * 9))
```

Comments

```
/**
 * This is
 * okay.
 */

// And so
// is this.

/* This is fine, too. */
```

```
someFunction(obvious Param, /* shouldRender= */ true, /* name= */ 'hello');
```

Language features

Local variable declarations

- Declare all local variables with either `const` or `let`.
- Every local variable declaration declares only one variable, example bad use: `let a = 1, b = 2;`
- Local variables are declared close to the point they are first used.

Language features

Array literals

```
const a1 = [x1, x2, x3];
const a2 = [x1, x2];
const a3 = [x1];
const a4 = [];
```

Language features

Objects literals

- Include a trailing comma whenever set a new property.
- Use an object literal ({} or {a: 0, b: 1, c: 2})

```
method() { return this.foo + this.bar; }
```

Language features

Classes

- Constructors are optional.
- The class keyword allows clearer and more readable class definitions than defining prototype properties.
- Do not use JavaScript getter and setter properties.

Language features

String literals

```
function arithmetic(a, b) {
  return `Here is a table of arithmetic operations:
${a} + ${b} = ${a + b}
${a} - ${b} = ${a - b}
${a} * ${b} = ${a * b}
${a} / ${b} = ${a / b}`;
}
```

```
const longString = 'This is a very long string that far exceeds the 80 ' +
    'column limit. It does not contain long stretches of spaces since ' +
    'the concatenated strings are cleaner.';
```

Language features

Number literals

- Numbers may be specified in decimal, hex, octal, or binary.
- Error:

```
let number = 0123;
```

Language features

For loops

- Three different kinds of for loops.
- All may be used, though for-of loops should be preferred when possible.
- Use exceptions.

Language features

Other important annotations

- Use identity operators ===/!==.
- Do not use the with keyword.
- Always terminate statements with semicolons.

```
Object example;      Object example();
```

Naming

Rules common to all identifiers

- Identifiers use only ASCII letters and digits.
- Give as descriptive a name as possible.
- Do not use abbreviations that are ambiguous.

```
let val;
let value;
let gravityValue;
```

Naming

Method names

- Method names are written in lowerCamelCase.
- Method names are typically verbs or verb phrases.

```
test<MethodUnderTest>_<state>_<expectedOutcome>
```

Naming

Other names

- Constant: in uppercase letters.
- Local variable and Parameter: in lowerCamelCase.
- Template parameter: Single-word or single-letter identifiers, and must be all-caps.

https://javascript.info/ninja-code

Universidad
de La Laguna

JSDOC

JSDOC

The basic

- Documentation generator
- Use comments to work like Doxygen
- The output is a web page
- It is very easy to use

Commenting vs Documenting Code

```
                              [illegible] size_t length, [illegible]
size_t src_idx = 0;
size_t dst_idx = 0;
for (; (src_idx + 2) < length; src_idx += 3, dst_idx += 4)
{
    uint8_t s0 = data[src_idx];
    uint8_t s1 = data[src_idx + 1];
    uint8_t s2 = data[src_idx + 2];

    dst[dst_idx + 0] = charset[(s0 & 0xfc) >> 2];
    dst[dst_idx + 1] = charset[((s0 & 0x03) << 4) | [illegible]
    dst[dst_idx + 2] = charset[((s1 & 0x0f) << 2) | [illegible]
    dst[dst_idx + 3] = charset[(s2 & 0x3f)];
}
if (src_idx < length)
{
    uint8_t s0 = data[src_idx];
    uint8_t s1 = (src_idx [illegible]
```

```
someFunction(obvious Param, /* shouldRender= */ true, /* name= */ 'hello');
```

Ok but, how do you install it?

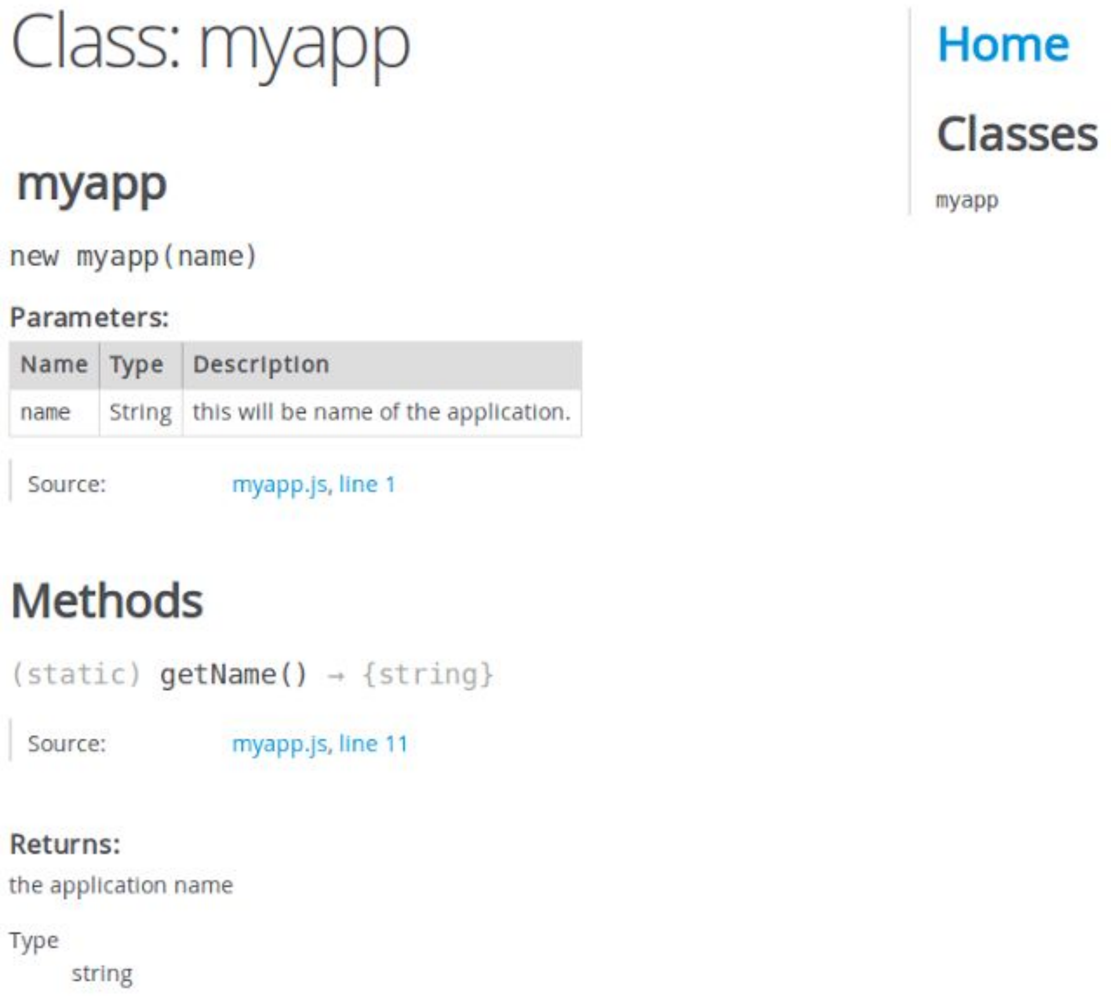
Let npm do it for you!

```
$ npm install -g jsdoc
```

or...

```
$ npm install jsdoc
```

A web page generated with it



Class: myapp

myapp

```
new myapp(name)
```

Parameters:

Name	Type	Description
name	String	this will be name of the application.

Source: myapp.js, line 1

Methods

```
(static) getName() → {string}
```

Source: myapp.js, line 11

Returns:

the application name

Type
string

Home

Classes

myapp

Now you can use it!

1° Need a JS code with JSdoc comments

```
/**
 * Represents a book.
 * @constructor
 * @param {string} title - The title of the book.
 * @param {string} author - The author of the book.
 */
function Book(title, author) {
}
```

Tags are the key

- @param | @argument
- @return | @returns
- @exemple
- @module
- @todo

Other jsdoc cheatsheet & original

Bake it!

2º Execute JSdoc

```
$ jsdoc <FileNames>
```

or...

```
$ ./node_modules/.bin/jsdoc <FileNames>
```

Configuration

To generate the template file

```
$ jsdoc -c /path/to/conf.json
```

or...

```
$ jsdoc -c /path/to/conf.js
```

Proper ways to document your code

- Header comments
- Enum and typedef comments
- Class comments
- Method and function comments

Header

```
/**
 * @author Antonio Guijarro <alu01012@ull.edu.es>
 * @file This is my cool script.
 * @copyright Antonio Guijarro 2019
 * @since 10.11.2019
 */
```

Enum and typedef

```
/**
 * Types of bandersnatches.
 * @enum {string}
 */
const BandersnatchType = {
  /** This kind is really frumious. */
  FRUMIOUS: 'frumious',
  /** The less-frumious kind. */
  MANXOME: 'manxome',
};
```

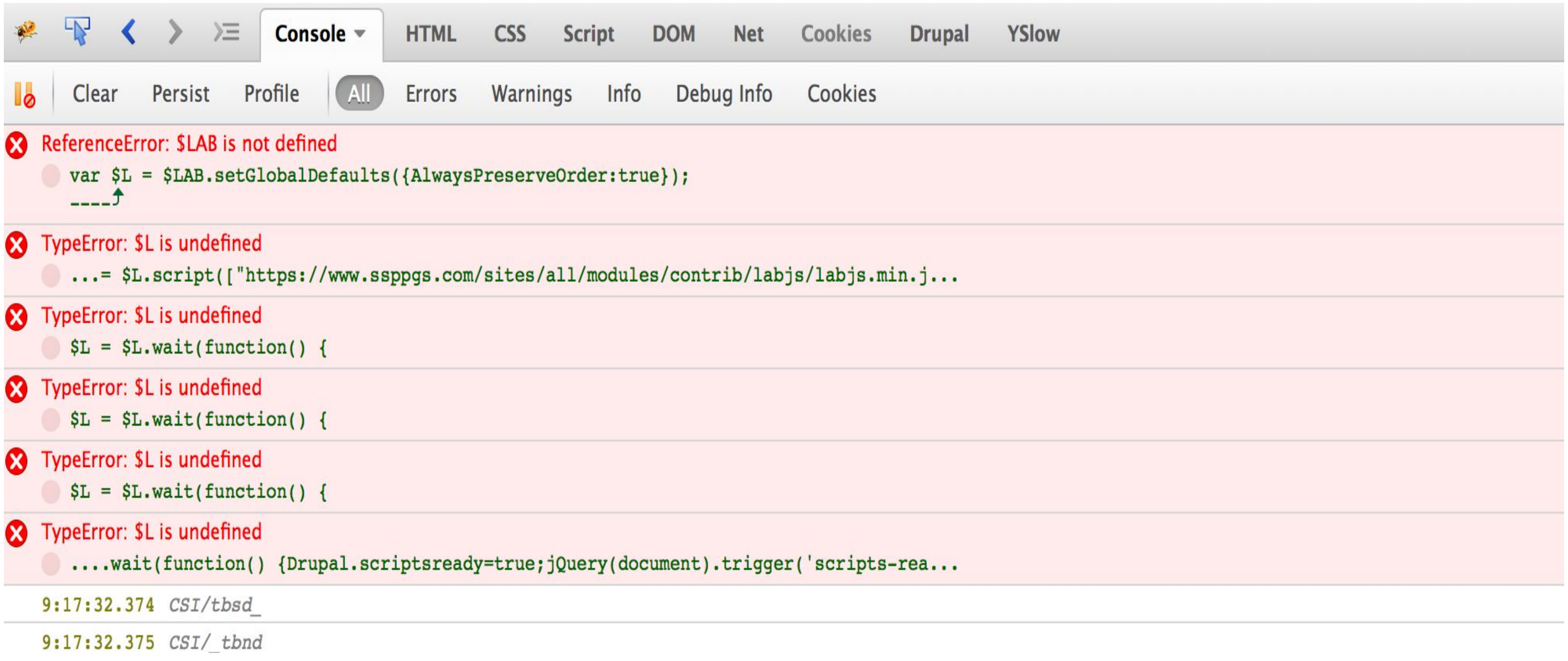
Class, method and function

```
/**
 * A fancier event target that does cool things.
 */
class MyFancyTarget extends EventTarget {
  /**
   * @param {string} arg1 An argument that makes
this more interesting.
   * @param {!Array<number>} arg2 List of numbers
to be processed.
   */
  constructor(arg1, arg2) {
  }
};
```

Universidad
de La Laguna

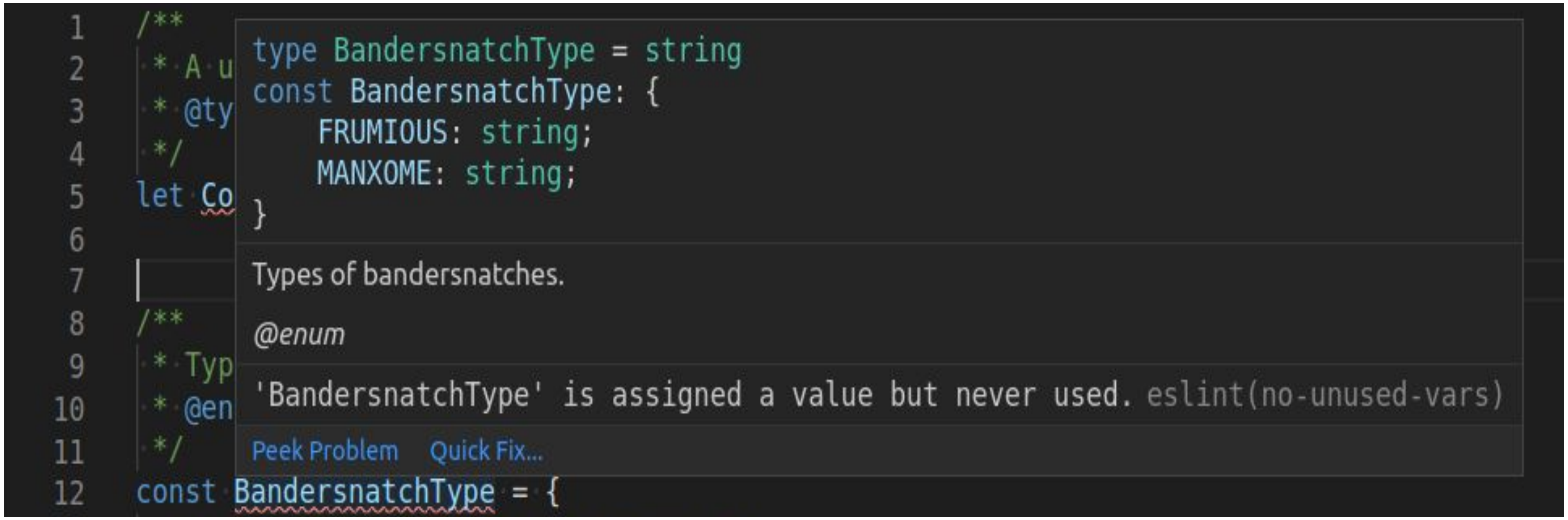
Linters

The times change and we change with them



```
ReferenceError: $LAB is not defined
  var $L = $LAB.setGlobalDefaults({AlwaysPreserveOrder:true});
  ----⮥
TypeError: $L is undefined
  ...= $L.script(["https://www.ssppgs.com/sites/all/modules/contrib/labjs/labjs.min.j...
TypeError: $L is undefined
  $L = $L.wait(function() {
TypeError: $L is undefined
  $L = $L.wait(function() {
TypeError: $L is undefined
  $L = $L.wait(function() {
TypeError: $L is undefined
  ....wait(function() {Drupal.scriptsready=true;jQuery(document).trigger('scripts-rea...
9:17:32.374 CSI/tbsd_
9:17:32.375 CSI/_tbnd
```

No more headaches 😌



```
/**
 * A u
 * @ty
 */
let Co

/**
 * Typ
 * @en
 */
const BandersnatchType = {
```

```
type BandersnatchType = string
const BandersnatchType: {
    FRUMIOUS: string;
    MANXOME: string;
}
```

Types of bandersnatches.

@enum

```
'BandersnatchType' is assigned a value but never used. eslint(no-unused-vars)
```

- Check syntax
- Find problems
- Enforce code style (Google obviously)

Where can I buy it?!

- Do not worry, it is free
- npm will save your life again

```
$ npm install --save-dev eslint
```

or with the Google configuration…

```
$ npm install --save-dev eslint
eslint-config-google
```

First steps and configuration

After the installation process

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

Configuration

- How would you like to use ESLint?
- What type of modules does your project use?
- Which framework does your project use?
- Where does your code run?
- How would you like to define a style for your project?
- What format do you want your config file to be in?
- Would you like to install them now with npm?

Bibliography

- Google and JavaScript Style Guide
- JSDOC
  - Installation
  - Configuration
- Linters
  - ESlint
  - Setup ESlint & how to use it

Universidad
de La Laguna