

# Style Guides



**Universidad**  
de La Laguna

# Introduction

- Code Styles are used to unify the code and make it more readable for everyone
- Good coding style is important, so that something like this does not happen:

```
const badFunc = (a,b) => {var r=1,i=0; for(;i<b;i++)r*=a;return r};
```

# What are we going to see?

- Coding Style
- Google Style Guide
- ESLint
- JSDoc

# Coding Style (I) - Functions

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

function takeElement(elem) {
  ...
}

function dropElement(elem) {
  ...
}

// the code which uses them
let elem = createElement();
takeElement(elem);
dropElement(elem);
```



```
// the code which uses them
let elem = createElement();
takeElement(elem);
dropElement(elem);

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

function takeElement(elem) {
  ...
}

function dropElement(elem) {
  ...
}
```



# Coding Style (I) - Functions

```
function main() {  
  const myArgs = processArgs();  
  
  isLowercase(myArgs) ? console.log('La letra es minúscula')  
    : console.log('La letra es mayúscula');  
  
  isVowel(myArgs) ? console.log('La letra es una vocal')  
    : console.log('La letra es una consonante');  
}  
  
if (require.main === module) {  
  main();  
}
```

```
const isVowel = function(letter) {  
  switch (letter) {  
    case 'a':  
    case 'e':  
    case 'i':  
    case 'o':  
    case 'u':  
    case 'A':  
    case 'E':  
    case 'I':  
    case 'O':  
    case 'U':  
      return true;  
    default:  
      return false;  
  }  
};
```



# Coding Style (II) – White spacing

```
const wrongFunction=function(){  
  let total=0;  
  for(let i=1;i<=10;i++){  
    total+=i;  
  }  
  return total;  
};  
console.log(wrongFunction());
```



```
const correctFunction = function() {  
  let total = 0;  
  for (let i = 1; i <= 10; i++) {  
    total += i;  
  }  
  return total;  
};  
console.log( correctFunction() );
```



# Coding Style (III) – Indentation



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



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



# Coding Style (IV) – If else

```
if (n < 0) {console.log(`This is not well done`);}
```

```
if (n < 0)  
  console.log(`Easy to have errors`);
```



```
if (n < 0) console.log(`This is acceptable, if it is short`);
```

```
if (n < 0) {  
  console.log(`This is a work of art`);  
}
```





# Coding Style (V) – Naming

```
var a = "";  
const auxFunction = function(param1, param2) {  
  var b = param1 * param2;  
  a = `The product between ${param1} and ${param2} is ${b}`;  
}
```



```
const productString = function(number1, number2) {  
  let result = number1 * number2;  
  return `The product between ${number1} and ${number2} is ${result}`;  
}
```



# Coding Style (V) – Naming



Exercise1/ex1.js  
finalExercise/defEx.js

practices/pascal-triangle.js  
src/inverse\_matrix.js



# Google Style Guide

- Created by Google to reinforce their coding standards
- Covers aesthetics as well as conventions and code standard
- The style guide is very extensive, so it is not possible to cover everything
- For other styles there are alternatives. E.g., Airbnb or `eslint:recommended`

# Google Style Guide – Basic Styling

- Non-ASCII characters are permitted

```
const units = '\u03bcs'
```



```
const units = 'μs'
```



- Files should always be encoded in UTF-8
- Filenames are allowed to include `-` and `\_` as special character.

# Google Style Guide – Naming

- Variables and Functions
  - Written in lowerCamelCase e.g.: getAllFishes(), totalCostCombined
- Constants
  - Written in CONSTANT\_CASE e.g.: ADMIN\_TOKEN, TAX\_RATE
- Classes and Enums
  - Written in UpperCamelCase e.g.: CarParts, TaxCalculator
- Always use concise wording e.g.: errorCount instead of cErr

# Google Style Guide – Spacing

```
const countTo = (number) => {  
  for (let i = 0; i < number; i++) {  
    console.log(i);  
  }  
};
```

- 2 more spaces per block
- No line break before opening braces
- Line break after opening and closing brace
- Space after reserved words (if, catch, for), except `function` and `super`
- Space after `;` | `,` | `:` and `//`

# Google Style Guide - Formatting

- Horizontal Alignment is discouraged, but permitted
- General line limit: 80

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

Ln 14, Col 13   Spaces: 2   UTF-8

- In VSCode the Indent can be set at the bottom right corner

# Google Style Guide – General Styling

- Arrow Functions are preferred

```
const moduleLocalFunc = (numParam, strParam) => numParam + Number(strParam);
```

- Use single quotes, when possible
- Switch-Case statements need a default case
- If possible, use for-of loops

```
switch (input) {  
  case 1:  
  case 2:  
    prepareOneOrTwo();  
  // fall through  
  case 3:  
    handleOneTwoOrThree();  
    break;  
  default:  
    handleLargeNumber(input);  
}
```



# Google Style Guide - Equality

- Use the `===` / `!==` operator, except for one case:  
Catching null and undefined values in one case

```
if (someObjectOrPrimitive == null) {  
    // Checking for null catches both null and undefined for objects and  
    // primitives, but does not catch other falsy values like 0 or the empty  
    // string.  
}
```

# Google Style Guide – Eval()

- Do NOT use eval or the Function(...String) constructor
- Can be used to run code, that is passed in during runtime
- Both are always a security risk

```
const ADMIN_PASSWORD = 'superSecretPassword';

const simpleCalc = (userInput) => {
  console.log(eval(userInput));
};
```

# Google Style Guide - Imports

```
import '../directory/file.js';  
  
import * as bigAnimals from './biganimals.js';
```

- Always add the file extension
- Do NOT import a file multiple times
- When doing named imports use lowerCamelCase

# ESLint



- Helps to follow style guide rules
- Can automatically format code
- NOT a replacement for knowing style guides
- There are alternatives e.g., JSLint TSLint or JSHint

# ESLint: Installation

## Global install

- ``npm install -g eslint``
- Removes need to install for every project
- It is possible to create default ruleset for every project

## Project install

- ``npm install eslint --save-dev``
- Easier to install for new people in the project
- All files in one folder

# ESLint: Initialisation

- ``npm init @eslint/config``

```
? How would you like to use ESLint? ...  
  To check syntax only  
  To check syntax and find problems  
▶ To check syntax, find problems, and enforce code style
```

- For this class: To check syntax, find problems, and enforce code style

# ESLint: Initialisation

```
? What type of modules does your project use? ...  
▶ JavaScript modules (import/export)  
  CommonJS (require/exports)  
  None of these
```

- In this class: JavaScript modules

# ESLint: Initialisation

```
? Which framework does your project use? ...  
  React  
  Vue.js  
▶ None of these
```

- In this class: None of these

```
? Does your project use TypeScript? ▶ No / Yes
```

- In this class: No



# ESLint: Initialisation

? Where does your code run?  
✓ Browser  
✓ Node

- In this class: Node

# ESLint: Initialisation

```
? How would you like to define a style for your project? ...  
▸ Use a popular style guide  
   Answer questions about your style
```

- In this class: Use a popular style guide

```
? Which style guide do you want to follow? ...  
   Airbnb: https://github.com/airbnb/javascript  
   Standard: https://github.com/standard/standard  
▸ Google: https://github.com/google/eslint-config-google  
   XO: https://github.com/xojs/eslint-config-xo
```

- In this class: Google

# ESLint: Initialisation

```
? What format do you want your config file to be in? ...  
▸ JavaScript  
  YAML  
  JSON
```

- In this presentation: JavaScript

```
? Would you like to install them now with npm? ▸ No / Yes
```

- Select yes

# .eslintrc

- This file is used to change eslint settings
- It is possible to add, remove or modify rules, set style guides etc:

```
'rules': {  
  'max-len': ['error', {'code': 81, 'ignoreComments': false}],  
  'linebreak-style': ['error', 'unix'], // try to change to windows  
},
```

- To stop ESLint for searching for other config files:

```
'root': true, // stops eslint from looking further
```

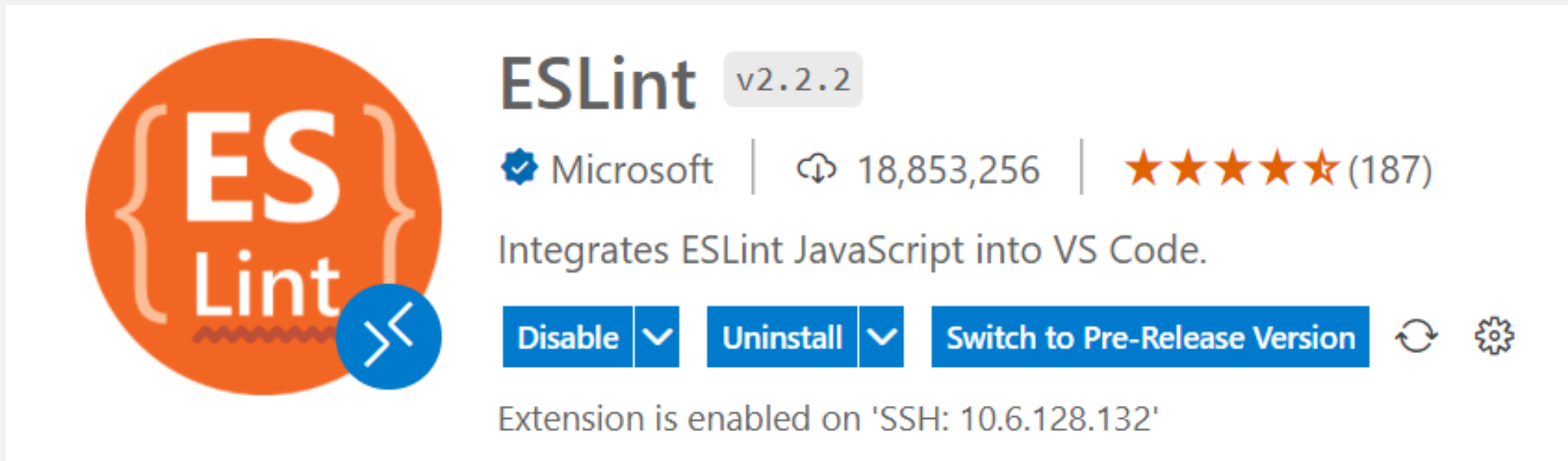
# .eslintignore

- This file is used to ignore files from eslint
- It basically works like a .gitignore file

```
1 // For the VSCode ESLint Plugin this file needs to be in root
2 src/eslint/ignored.js
```

# VSCode and ESLint

- Install the ESLint Extension





The image shows the ESLint extension interface within the Visual Studio Code ecosystem. On the left is the ESLint logo, which consists of an orange circle containing the text 'ESLint' in white, with a blue circle containing a white '<<' symbol at the bottom right. To the right of the logo, the text 'ESLint' is displayed in a large, bold, dark blue font, followed by a version tag 'v2.2.2' in a smaller, grey font. Below this, the publisher 'Microsoft' is listed with a blue checkmark icon, followed by a download icon and the number '18,853,256', and a five-star rating with '(187)' reviews. A description line reads 'Integrates ESLint JavaScript into VS Code.' Below the description are three blue buttons: 'Disable' with a dropdown arrow, 'Uninstall' with a dropdown arrow, and 'Switch to Pre-Release Version'. To the right of these buttons are icons for refreshing and settings. At the bottom, a status line indicates 'Extension is enabled on 'SSH: 10.6.128.132''.

**ESLint** v2.2.2

✓ Microsoft | 18,853,256 | ★★★★★ (187)

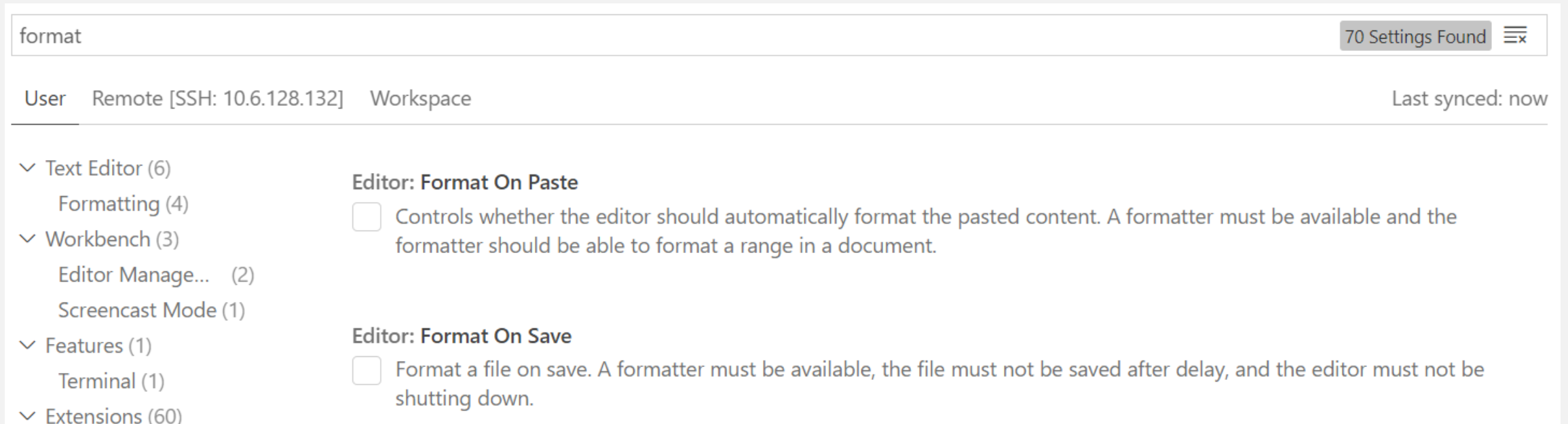
Integrates ESLint JavaScript into VS Code.

[Disable](#) [Uninstall](#) [Switch to Pre-Release Version](#)  

Extension is enabled on 'SSH: 10.6.128.132'

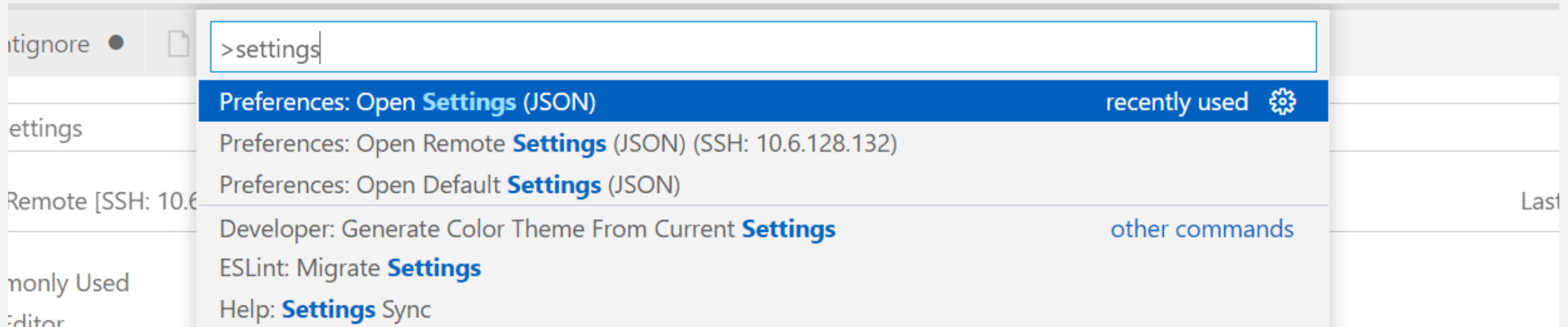
# VSCode and ESLint

- Disable Autoformat
  - File -> Preferences -> Settings -> type 'format' and disable Format on Paste and Format on Save



# VSCode and ESLint

- Press `F1` and enter settings to open the Settings in JSON format



- Important: Open the local file, not the remote one



# VSCode and ESLint

```
"eslint.alwaysShowStatus": true,  
"editor.codeActionsOnSave": {  
  "source.fixAll.eslint": false  
},  
"eslint.run": "onSave",  
"eslint.options": {  
  "resolvePluginsRelativeTo": "/home/usuario/node_modules"  
},
```

- 1: shows indication of the bottom of the IDE
- 2-4: formats automatically on save
- 5: when should eslint run: onSave or onType
- 6-8: set path to style guide, only necessary on global installs

# Bad Code

```
var t = '';  
  
const badFunc = (a,b) => {var r=1,i=0; for(;i<b;i++)r*=a;return r};
```

- Can you tell us the mistakes made in this code sample?

# Bad Code – Formatted by ESLint

```
const t = '';  
  
const badFunc = (a, b) => {  
  let r=1; let i=0; for (;i<b; i++)r*=a; return r;  
};
```

- ESLint can not make bad code magically into good code

# Good Code

```
const goodFunc = (base, exponent) => {  
  let result = 1;  
  
  for (let i = 0; i < exponent; i++) {  
    result *= base;  
  }  
  
  return result;  
};
```

# JSDoc – What is it?



- Documentation tool.
- Free software.
- Produce documentation accessible in formats like HTML.
- Quite similar to Javadoc.
- Same purpose as other used tools in the degree as Rdoc or Doxygen.

# JSDoc – How to use it

- Installation

- \$ sudo npm install -g jsdoc -> Globally
- \$ sudo npm install --save-dev jsdoc -> Locally (package.json)

- Development

- Use JSDoc

# JSDoc – How to use it

- Installation
- Development
  - Document your code
  - Write the code
- Use JSDoc

# JSDoc – Headers

```
/**  
 * Universidad de La Laguna  
 * Escuela Superior de Ingeniería y Tecnología  
 * Grado en Ingeniería Informática  
 * Programación de Aplicaciones Interactivas  
 *  
 * @author Jorge Hdez. Batista  
 * @since Feb 28 2022  
 * @desc Documentation  
 *   This is the way we should comment a header, so all the information is added  
 *   During this file, you'll find an example of how we can use the tool JSDoc.  
 *  
 * @see {@link https://jsdoc.app}  
 */
```



# JSDoc – Variables



```
/** @const
    @type {string}
    @default
 */
const RED = 'FF0000';
```

```
/**
 * A number, or a string containing a number.
 * @typedef {(number|string)} NumberLike
 */

/**
 * Set the magic number.
 * @param {NumberLike} x - The magic number.
 */
function setMagicNumber(x) {
}
```

# JSDoc – Functions



```
/**  
 * @desc Given a number of layers, determines the total preparation time.  
 *  
 * @param {number} numberOfLayers  
 * @returns {number} the total preparation time  
 */  
export function preparationTimeInMinutes(numberOfLayers) {  
  return numberOfLayers * 2;  
}
```

# JSDoc – Classes (I)



```
/**  
 * @classdesc Class representing a Pokémon  
 * @class  
 * @property {string} name  
 * @property {string} [nickname]  
 */  
class Pokemon() {
```

# JSDoc – Classes (II)

```
/**
 * @class Pokémon
 * @classdesc Class representing a Pokémon
 * @property {string} name
 * @property {string} [nickname]
 */
class Pokemon {
  constructor (pokedex, number, nickname = "") {
    /**
     * Name is a property that contains the name of a Pokémon
     * @type {string}
     * @public
     */
    this.name = pokedex.name(number);
    /**
     * Nickname is an optional property that may contains the
     * name the trainer gived to his pokémon
     * @type {string}
     * @public
     */
    this.nickname = nickname;
    // ...
  }
}
```

# JSDoc – Classes (III)



```
class Pokemon {  
  // ...  
  attack(move, opponent) {  
    // ...  
  }  
}
```

```
/**  
 * @class Pikachu  
 * @classdesc Class representing a Pikachu  
 * @extends Pokémon  
 */  
class Pikachu extends Pokemon {  
  // ...  
  /**  
   * Pikachu attacks another pokémon with thunderbolt  
   * @override  
   */  
  attack(opponent) {  
    move = 'Thunderbolt';  
  }  
}
```

# JSDoc – How to use it

- Installation
- Development
- Use JSDoc
  - jsdoc file.js



/out

# Sources

- <https://google.github.io/styleguide/jsguide.html> (02.03.2022)
- <https://github.com/yannickcr/eslint-plugin-react/issues/2339> (02.03.2022)
- <https://eslint.org/docs/user-guide/getting-started> (02.03.2022)
- <https://lenguajejs.com/javascript/caracteristicas/eslint/> (02.03.2022)
- <https://www.freecodecamp.org/news/google-publishes-a-javascript-style-guide-here-are-some-key-lessons-1810b8ad050b/> (02.03.2022)
- [https://www.w3schools.com/js/js\\_best\\_practices.asp](https://www.w3schools.com/js/js_best_practices.asp) (2.03.2022)
- [https://www.w3schools.com/js/js\\_best\\_practices.asp](https://www.w3schools.com/js/js_best_practices.asp) (02.03.2022)
- <https://javascript.info/coding-style> (02.03.2022)
- <https://codezen.rishimohan.me> (02.03.2022)
- <https://jsdoc.app> (02.03.2022)
- <https://deepsources.io/blog/javascript-code-quality-best-practices/> (02.03.2022)
- <https://www.geeksforgeeks.org/documentation-comments-in-jsdoc/> (02.03.2022)
- [https://www.w3.org/wiki/JavaScript\\_best\\_practices](https://www.w3.org/wiki/JavaScript_best_practices) (02.03.2022)