

# Coding conventions

✂ Javascript & AngularJS ✂



# Agenda

- ◆ General Coding conventions.
- ◆ Javascript Coding conventions and a little about AngularJS.
- ◆ Introduce LINT tools.
- ◆ Tiny Demo.



# Coding for 🍆, or 👤?

- ◆ Programming is the art of telling another human what one wants the computer to do
  - Donald Knuth, the author of The art of Computer Programming Series.
- ◆ Any fool can write code that a computer can understand. Good programmers write code that humans can understand
  - Martin Fowler, author of Refactoring and Improving the Design of Existing Code.

Don't be a fool 🤡



# WHAT's the💩convention?

- ◆ Naming and declaration rules for variables and functions.
- ◆ Rules for the use of white space, indentation, and comments.
- ◆ Programming practices and principles






# WHY? 🤔

- ◆ Improve code readability.
- ◆ Make code maintenance easier.
- ◆ Avoid unknown behavior of Javascript.
- ◆ Help you transfer knowledge across projects.
- ◆ Learn code more quickly on a new project.





# WHO?

- ◆ Sonner is better for  (junior)
- ◆ Learned by . (experienced programmers)
- ◆ Careful consideration and apply for all .



HOW? 😊



# General naming conventions

- ◆ Names should be as specific as possible
- ◆ Speaks to the problem rather than the solution
- ◆ Express the what more than the how

Purpose of Variable	Good Names, Good Descriptors	Bad Names, Poor Descriptors
Running total of checks written to date	<i>runningTotal, checkTotal</i>	<i>written, ct, checks, CHKTTL, x, x1, x2</i>
Velocity of a bullet train	<i>velocity, trainVelocity, velocityInMph</i>	<i>vel, v, tv, x, x1, x2, train</i>
Current date	<i>currentDate, todaysDate</i>	<i>cd, current, c, x, x1, x2, date</i>
Lines per page	<i>linesPerPage</i>	<i>lpp, lines, l, x, x1, x2</i>

inputRec -> computer term 👎

employeeData -> domain term 👍

printerReady is better bitFlag

sum is better calcValue



# Var Name length

numberOfSeatsInTheStadium 😅

maximumNumberOfPointsInModernOlympics 😱

- ◆ Names should have average 9 to 15 characters.
- ◆ Shorter is better, but it should be clear as they need to be.

Too long: *numberOfPeopleOnTheUsOlympicTeam*  
*numberOfSeatsInTheStadium*  
*maximumNumberOfPointsInModernOlympics*

Too short: *n, np, ntm*  
*n, ns, nsisd*  
*m, mp, max, points*

Just right: *numTeamMembers, teamMemberCount*  
*numSeatsInStadium, seatCount*  
*teamPointsMax, pointsRecord*

Too short don't describe enough meaning  
Too long hard to type and look awful



# Naming status

## Better version

```
if ( flag ) ...  
if ( statusFlag & 0x0F ) ...  
if ( printFlag == 16 ) ...  
if ( computeFlag == 0 ) ...
```

```
flag = 0x1;  
statusFlag = 0x80;  
printFlag = 16;  
computeFlag = 0;
```

WTF? 🤔

```
if ( dataReady ) ...  
if ( characterType & PRINTABLE_CHAR ) ...  
if ( reportType == ReportType_Annual ) ...  
if ( recalNeeded = false ) ...
```

```
dataReady = true;  
characterType = CONTROL_CHARACTER;  
reportType = ReportType_Annual;  
recalNeeded = false;
```





# Naming boolean

- ◆ Give boolean variables names that imply TRUE or FALSE  
(Do not ~~status~~)
- ◆ Good examples 💪
  - ◆ Done / isDone
  - ◆ Error / isError
  - ◆ Found / isFound
  - ◆ Success / ok



# JS Naming Conventions 😊💧

- ◆ Use camelCase for variables and functions.
- ◆ Use UPPERCASE for global variables and constants.
- ◆ Use PascalCase when naming constructors or classes.
- ◆ Always use lower case file names  
(vietnam.jpg # Vietnam.jpg).



# Spaces & Indentation

- ◆ Always put spaces around operators ( $=$   $+$   $-$ ), and after commas.
- ◆ Always use 4 spaces for indentation of code blocks.
- ◆ Tips: Do not use tabs for indentations.  
(Different editors treat tabs differently)



- ◆ Always end a simple statement with a semicolon.
- ◆ General rules for complex statements.



# Object rules

- ◆ Place the opening bracket on the same line as the object name.
- ◆ Use colon plus one space between each property.
- ◆ Use quotes around string values (not around numeric values).
- ◆ Do not add a comma after the last property-value pair.
- ◆ Always end an object definition with a semicolon.

```
var programmer = {  
  firstName: "Jon",  
  lastName: "Snow",  
  age: 32,  
  eyeColor: "black",  
  skill: "🔪🏹👑"  
};
```



# JS Best Practices

- ◆ Avoid Global Variables.
- ◆ Local variables must be declared with var keyword, otherwise they will become global variables. (Tips: Strict mode does not allow undeclared variables.)
- ◆ Put all declarations at the top of each script or function, and initialize them.



# Declaration on top

- ◆ Give cleaner code
- ◆ Provide a single place to look for local variables
- ◆ Avoid unwanted global variables
- ◆ Reduce the possibility of unwanted re-declarations

```
// bad
var firstName = "",
    lastName = "",
    price = 0,
    discount = 0,
    fullPrice = 0,
    myArray = [],
    myObject = {};
```

```
// good
var firstName = "";
var lastName = "";
var price = 0;
var discount = 0;
var fullPrice = 0;
var myArray = [];
var myObject;
```



- ◆ Always treat numbers, strings, or booleans as primitive values. Not as objects (because they slows down execution speed, and produce nasty side effects). Don't use `new Object()`.
- ◆ Use `===` comparison (because `==` operator always converts before comparison).
- ◆ Always end your switch statements with a default. Even if you think there is no need for it.



# more IDEAS

- ◆ Don't live with break windows (from Pragmatic).
- ◆ KISS/YAGNI/DRY principles.
- ◆ References list slide



# Code comment

- ◆ Use `//` for single line comments.
- ◆ Use `/** ... */` for multi-line comments. Include a description
- ◆ Prefixing your comments with `FIXME` or `TODO` helps other developers quickly understand or if you're suggesting a solution to the problem that needs to be implemented

```
/**
 * make() returns a new element
 * based on the passed in tag name
 */
@param {Number} num
@return {Object} this
*/
function Calculator(num) {
  // FIXME: shouldn't use a global here
  total = 0;

  // TODO: total should be configurable by an options param
  this.total = 0;

  return this;
}
```





# Style guide

<https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md>



# Intro Lint tool & Demo

- ◆ Why lint?
- ◆ Linting tools
- ◆ Configure ESLint
- ◆ Set up ESLint

```
4   <label>id: </label>{{hero.id}}</div>
5   <div>
6     <label>name: </label>
7     <input [(ngModel)]="hero.name" placeholder="name" />
8   </div>
9   <button (click)="goBack()">Back</button>
10 </div>
```

20 WARNINGS Filter by type or text

- ▲ Special characters must be escaped : [ < ]. [1, 1]
- ▲ Special characters must be escaped : [ > ]. [1, 18]
- ▲ Special characters must be escaped : [ < ]. [7, 5]
- ▲ Special characters must be escaped : [ > ]. [7, 56]
- ▲ Special characters must be escaped : [ < ]. [9, 3]
- ▲ Special characters must be escaped : [ > ]. [9, 29]
- ▲ Tag must be paired, no start tag: [ </button> ] (9, 34)
- ▲ Tag must be paired, no start tag: [ </div> ] (10, 1)
- ✖ heroes.component.html app 9
  - ▲ Special characters must be escaped : [ < ]. [3, 3]
  - ▲ Special characters must be escaped : [ > ]. [5, 29]
  - ▲ Tag must be paired, no start tag: [ </li> ] (7, 3)



# Why Lint?

ENFORCE CONSISTENCY	AVOID MISTAKES
Curly brace position { }	Extra paranthesis ()
confirm / alert	Overwriting function
Globals	Assignment in conditional
eval()	debugger / console.log



# Choose a Linter



Use this 💪





# Configuring ESLint



# Configuring ESLint

1. Config format?
2. Which built-in rules?
3. Warning or errors?
4. Which plugins?
5. Use preset instead?





# #1 - Config format

ESLint supports configuration files in several formats:

- JavaScript - use `.eslintrc.js` and export an object containing your configuration.
- YAML - use `.eslintrc.yaml` or `.eslintrc.yml` to define the configuration structure.
- JSON - use `.eslintrc.json` to define the configuration structure. ESLint's JSON files also allow JavaScript-style comments.
- **Deprecated** - use `.eslintrc`, which can be either JSON or YAML.
- `package.json` - create an `eslintConfig` property in your `package.json` file and define your configuration there.

If there are multiple configuration files in the same directory, ESLint will only use one. The priority order is:

1. `.eslintrc.js`
2. `.eslintrc.yaml`
3. `.eslintrc.yml`
4. `.eslintrc.json`
5. `.eslintrc`
6. `package.json`



# #2 - Which rules"

## Rules

Rules in ESLint are grouped by category to help you understand their purpose.

No rules are enabled by default. The `"extends": "eslint:recommended"` property in a configuration file enables rules that report common problems, which have a check mark ✓ below.

The `--fix` option on the command line automatically fixes problems (currently mostly whitespace) reported by rules which have a wrench ⚙ below.



## Possible Errors

These rules relate to possible syntax or logic errors in JavaScript code:

	<code>for-direction</code>	enforce "for" loop update clause moving the counter in the right direction.
	<code>getter-return</code>	enforce <code>return</code> statements in getters
	<code>no-await-in-loop</code>	disallow <code>await</code> inside of loops
✓	<code>no-compare-neg-zero</code>	disallow comparing against <code>-0</code>
✓	<code>no-cond-assign</code>	disallow assignment operators in conditional expressions
✓	<code>no-console</code>	disallow the use of <code>console</code>
✓	<code>no-constant-condition</code>	disallow constant expressions in conditions
✓	<code>no-control-regex</code>	disallow control characters in regular expressions
✓ ⚙	<code>no-debugger</code>	disallow the use of <code>debugger</code>
✓	<code>no-dupe-args</code>	disallow duplicate arguments in <code>function</code> definitions
✓	<code>no-dupe-keys</code>	disallow duplicate keys in object literals
✓	<code>no-duplicate-case</code>	disallow duplicate case labels
✓	<code>no-empty</code>	disallow empty block statements



# #3 - Warning or Errors?

WARNING 	ERROR 
Can continue development	Breaks the build
Can be ignored	Cannot be ignored
Team must agree: Fix warnings	Team is forced to comply



# #4 - Which plugins?

- ◆ eslint-plugin-react
- ◆ eslint-plugin-angular
- ◆ eslint-plugin-node
- ◆ awesome ESLint





# #5 - Use a Preset?



From scratch



Recommended



Presets





# DEMO



- ESLint Recommended
- Run automatically with eslint-watch
- Run linting as part of npm start





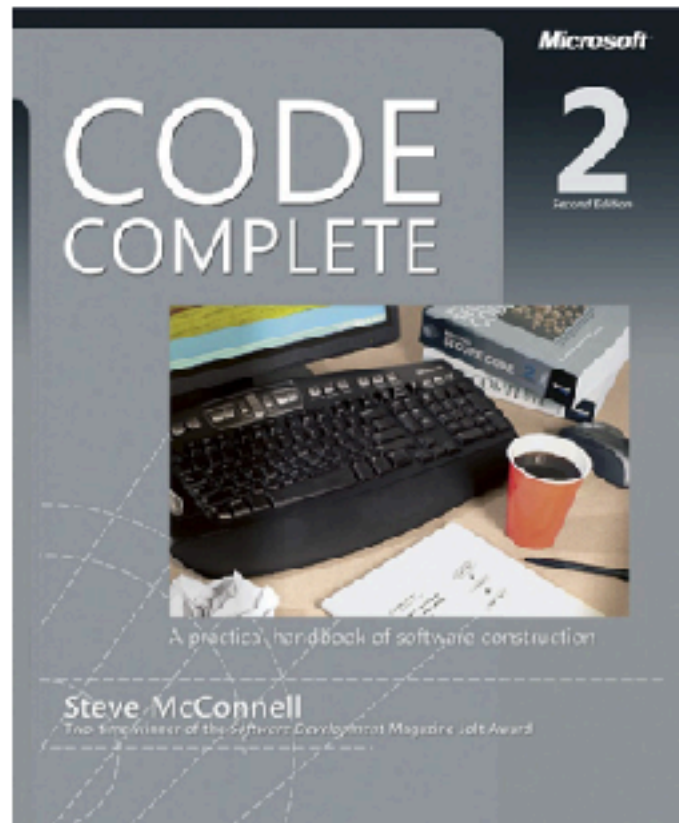
# References

- ◆ <https://eslint.org/docs/rules/#best-practices>
- ◆ [https://www.w3schools.com/js/js\\_best\\_practices.asp](https://www.w3schools.com/js/js_best_practices.asp)
- ◆ <https://github.com/airbnb/javascript/tree/es5-deprecated/es5>
- ◆ <https://github.com/johnpapa/angular-styleguide/blob/master/a1/README.md>

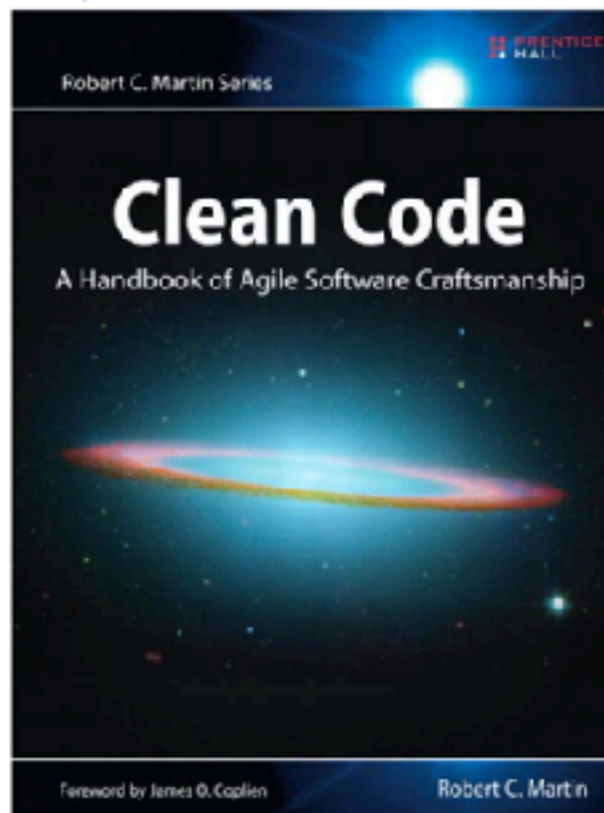


👉 One more thing...

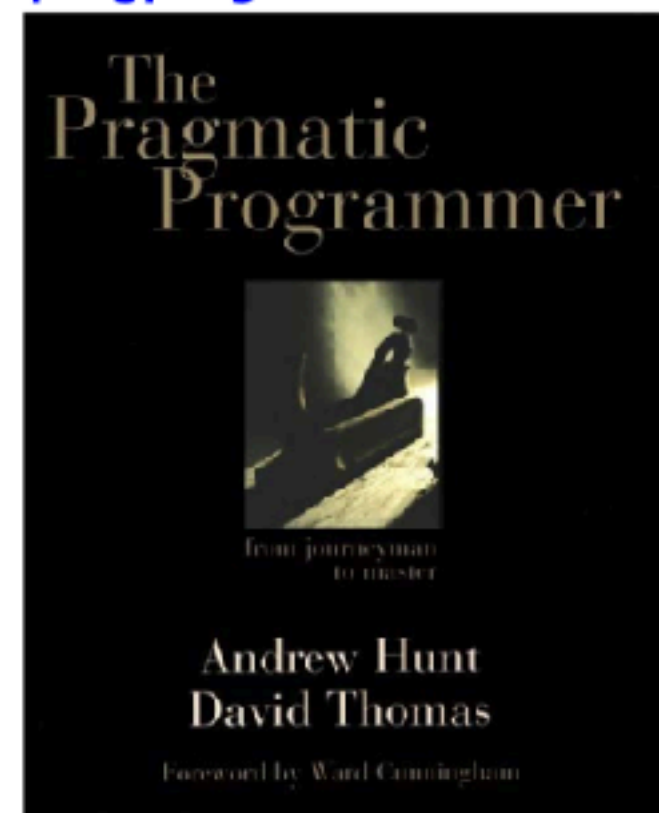
Steve McConnell  
[stevemcconnell.com](http://stevemcconnell.com)



Robert C. Martin  
[objectmentor.com](http://objectmentor.com)



Andrew Hunt, David Thomas  
[pragprog.com](http://pragprog.com)



Best books



🔥 To become a better programmer 🔥



# Don't Live with Broken Windows

*In inner cities, some buildings are beautiful and clean, while others are rotting hulks. Why? Researchers in the field of crime and urban decay discovered a fascinating trigger mechanism, one that very quickly turns a clean, intact, inhabited building into a smashed and abandoned derelict*

*A broken window.*

*One broken window, left unrepaired for any substantial length of time, instills in the inhabitants of the building a sense of abandonment—a sense that the powers that be don't care about the building. So another window gets broken. People start littering. Graffiti appears. Serious structural*

*damage begins. In a relatively short space of time, the building becomes damaged beyond the owner's desire to fix it, and the sense of abandonment becomes reality.*

*The "Broken Window Theory" has inspired police departments in New York and other major cities to crack down on the small stuff in order to keep out the big stuff. It works: keeping on top of broken windows, graffiti, and other small infractions has reduced the serious crime level.*

***Don't leave "broken windows"** (bad designs, wrong decisions, or poor code) unrepaired. Fix each one as soon as it is discovered. If there is insufficient time to fix it properly, then board it up. Perhaps you can comment out the offending code, or display a "Not Implemented" message, or substitute dummy data instead. Take some action to prevent further damage and to show that you're on top of the situation.*

*We've seen clean, functional systems deteriorate pretty quickly once windows start breaking. There are other factors that can contribute to software rot, and we'll touch on some of them elsewhere, but neglect accelerates the rot faster than any other factor.*

*You may be thinking that no one has the time to go around cleaning up all the broken glass of a project. If you continue to think like that, then you'd better plan on getting a dumpster, or moving to another neighborhood. Don't let entropy win.*