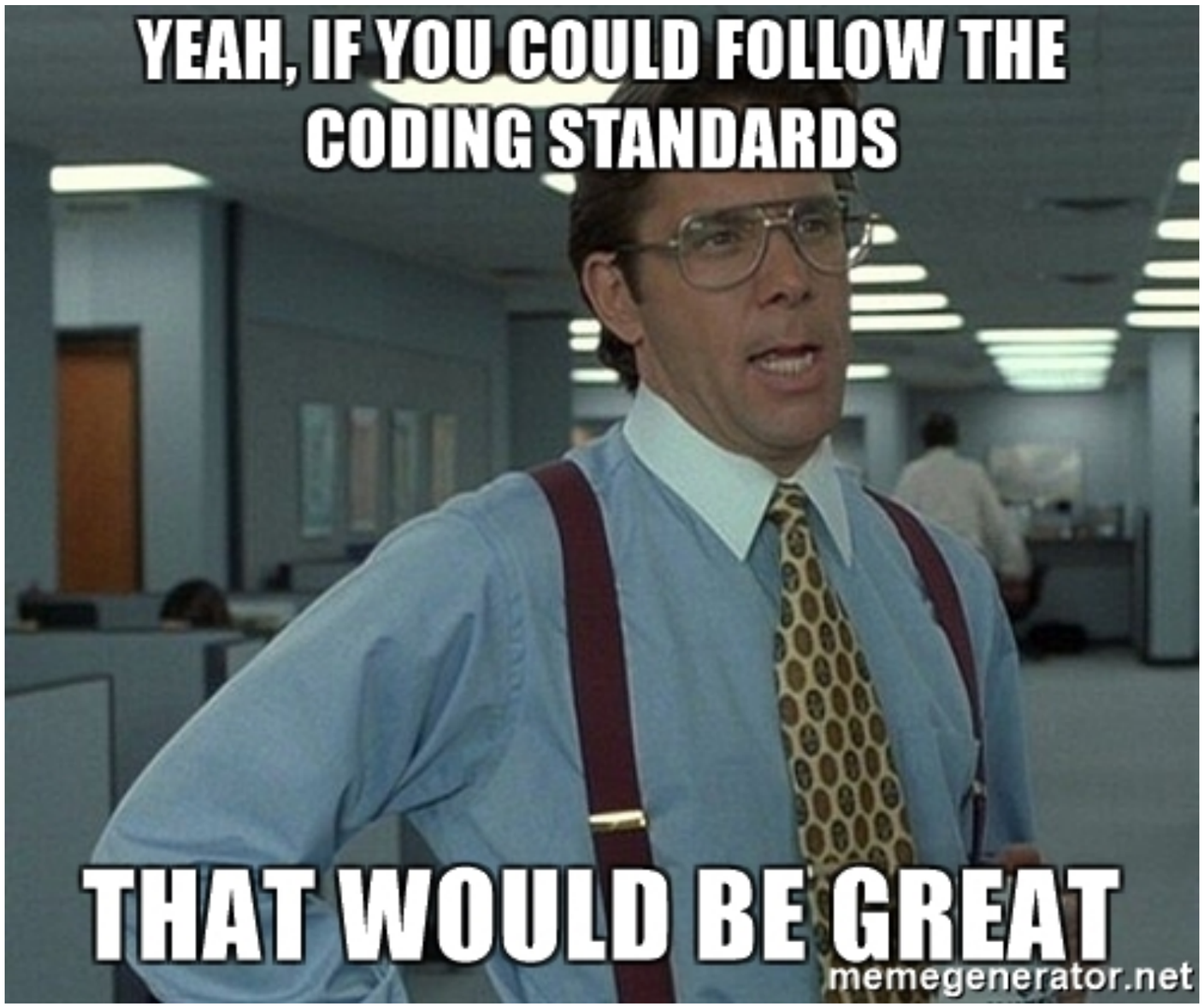


Coding to Standards

A guide to incorporating standards into your projects

What are coding standards?

- A set of guidelines that developers agree on and adhere to when writing code.

- They are generally language specific and can cover all aspects of programming in that language like indentation, comments, white space, naming conventions, programming principles, etc.

- Important to note that they are *guidelines*, there will always be a need for exceptions and thats ok.

Why code with standards?

Guidelines help

- Avoid common coding errors
- Enforce secure coding practices
- Improve the readability of your code
- Make maintenance easier
- Ensure that code written within a single project is consistent

Without Standards

- No agreed upon practices
    - Code Reviews can get bogged down with formatting issues, distracting from actual code review
        - Pull requests with 100 changes that are just spacing
- You choose not to follow them
    - Build processes may run as code gets pushed to each environment, so a failed linter can block a release
    - Some open-source projects may run linters on the pull request, if your code does not pass the linter you can't submit

But I work for myself, by myself!
I can read my own code!

- Have you ever returned to a project years later?
- Can you guarantee that you will always only work by yourself? What if you need to hand off your work to someone else?
- Ever want to contribute to an open-source project?
- Following coding guidelines can make you a more confident coder

Types of Coding Standards

- There are coding standards for each language you use
- In a typical WP project you will have css, html, javascript and php.
- There are a few standards that can be set environment wide like spaces vs tabs and trailing spaces. But most are set and managed on a per language basis.

WordPress Coding Standards

- The WordPress community has developed a set of standards for CSS, HTML, JavaScript and PHP.
- If you want to contribute to WordPress core, you must follow these standards.
- https://make.wordpress.org/core/handbook/best-practices/coding-standards

Tools can help!

- **Linters** - analyze your code and find errors
- **Formatters** - format your code to a standard

- Build Tools (gulp, webpack, etc)
- Editor Extensions
- CLI (Command Line Interface)

Recommended Tools

- StyleLint (CSS)
- ESLint (Javascript)
- PHP Code Sniffer (PHPCS)
- Prettier (CSS & JS)
- PHP Code Beautifier (PHPCBF)

Why these?

- Each have modules for common build tools (gulp, webpack, grunt)
- Each have Extensions for common Editors (VSCode, Atom, Sublime)
- The Formatters have existing relationships with the Linters
- Each have an associated WordPress Configuration

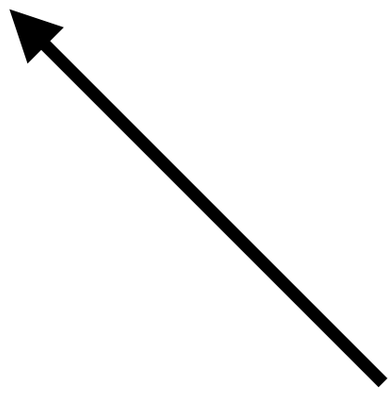
Three Components for Each Language

Linters (and Formatters) in your **Build Processes**

- Enforce consistency across contributors
- Live with the project code-base
- Errors are reported the time of build

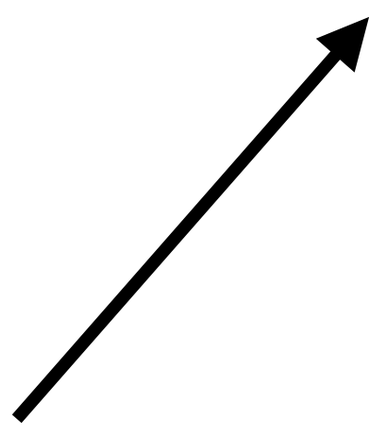
Linters and Formatters in your **Editor**

- Help you write code that meets the guidelines
- In your local environment
- Errors are reported/fixed as you work



Configuration Files keep the two in sync

Contains the ruleset



Stylelint & Prettier

1. Add NPM Modules to the root of your project

```
$ npm init
$ npm install stylelint stylelint-config-wordpress --save-dev
```

2. Create a config file in the root of your project

```
$ touch .stylelintrc.json
```

Edit File to tell it about the config file

```
{
    "extends": "stylelint-config-wordpress"
}
```

Can also configure for SCSS

```
{   "extends": [
      "stylelint-config-wordpress/scss"
      ]
  }
```

Stylelint & Prettier (cont'd)

3. Add StyleLint to your build process

4. Add StyleLint Extensions to your editor
   Sublime
   Atom
   VSCode

5. Add Prettier Extension to your editor
   (note: in VSCode if you have Beautifier installed, disable it, there is a known bug where Beautifier takes precedence over Prettier)

Stylelint & Prettier (cont'd)

6. Configure your workspace

    - Tell Editor to Format on Save, set Default Formatter

    - Enable StyeLint

    - Tell Prettier to use Stylelint

    - Restart VSCode

VSCode: settings.json

```
"[css]": {
    "editor.formatOnSave": true,
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  },
  "stylelint.enable": true,
  "prettier.stylelintIntegration": true,
```

ESLint & Prettier

1. Add NPM Modules to the root of your project

   ```
   $ npm install eslint @wordpress/eslint-plugin --save-dev
   ```

2. Create a config file in the root of your project

   ```
   $ touch .elslintrc.json
   ```

   Edit File to tell it about the config file

   ```
   {
        "extends": [ "plugin:@wordpress/eslint-plugin/recommended"
   ]
   }
   ```

ESLint & Prettier (cont'd)

3. Add ESLint to your build process

4. Add ESLint Extensions to your editor
   Sublime, Atom, VSCode

5. Add Prettier Extension to your editor

ESlint & Prettier (cont'd)

6. Configure your workspace

    - Tell Editor to Format on Save, set Default Formatter
    - Tell Prettier to use ESlint

VSCode: settings.json

```
"[javascript]": {
    "editor.formatOnSave": true,
    "editor.defaultFormatter": "esbenp.prettier-vscode"
  },
"prettier.eslintIntegration": true,
"eslint.autoFixOnSave": false,
```

PHP Standard

- The WP PHP Standard is comprehensive and may feel strict if you are new to it
- There are four options

1. **WordPress** – A complete set with all of the sniffs in the project. Includes Core, Docs, Extra, and VIP.
2. **WordPress-Core** – The main ruleset for WordPress core coding standards.
3. **WordPress-Docs** – Additional ruleset for WordPress inline documentation standards.
4. **WordPress-Extra** – Extended ruleset for recommended best practices; not sufficiently covered in the WordPress core coding standards. Includes WordPress-Core.

PHP Code Sniffer and Beautifier

1. Add Composer Modules to the root of your project

```
$ composer require --dev dealerdirect/
phpcodesniffer-composer-installer
$ composer require --dev wp-coding-standards/
wpcs
```

   - This installs phpcs, phpcbf, the WP standard, and tell WPCS to look for your standard

2. Add to your build process
   Gulp-PHPCS

PHP Code Sniffer and Beautifier (cont'd)

3. Install the Editor Extensions
   PHPCS - PHP Codesniffer
   PHPCSBF - Code Beautifier and Fixer
   - VSCode: PHPCS, PHPCBF
   - Atom
   - Sublime
4. Configure your Workspace

PHP Code Sniffer and Beautifier (cont'd)

3. Configure Your Workspace
   VSCODE:

```
{
  "editor.formatOnSave": false,
  "phpcs.enable": true,
  "phpcbf.onsave": true,
  "phpcs.composerJsonPath": "app/public/wp-content/themes/my-wp-theme/composer.json",
  "phpcs.standard": "WordPress",
  "phpcbf.debug": true,
  "phpcbf.standard": "WordPress",
  "phpcbf.executablePath": "/Users/PattyO/Local_Sites/my-site/app/public/wp-content/t
my-wp-theme/vendor/bin/phpcbf"
}
```

Bonus: Editor Config

- Language agnostic rules like tabs v spaces, trailing whitespace, final new line. Works across editors and IDE's.

- Wordpress Editor Config
  - Add .editorconfig file to the root of your project.
  - May need to install an extension for your editor (more info at EditorConfig.org)

Additional Editor Tools:

- TrailingSpaces (highlight and remove trailing white-space)

Overriding the Standard

- Turn off rules when extending the configuration

```
{
    "extends": "stylelint-config-wordpress",
    "rules" : {
        "at-rule-empty-line-before": null
        }
}
```

- Ignore lines within your code
  - StyleLint
  - ESLint
  - PHPCS

Resources

- WordPress Coding Standards
- StyleLint, ESLint, PHPCodeSniffer
- stylelint-config-wordpress
- NPM WordPress ESLint Plugin
- WordPress PHP Coding Standards for PHP_CodeSniffer