Pulilab Coding Guidelines Documentation

Release 0.3

Pulilab LLC

November 07, 2012

CONTENTS

Contents:

CHAPTER

ONE

STYLE GUIDE

Note: Our style guide is based on Mozilla's, Bracket's and Pocoo's guides.

1.1 HTML

Classes and id's in HTML use all lower-case with dashes (-), not camelCase or under_scores:

Do this:

```
<div id="search-results">
<span class="title-wrapper">
```

Not this:

```
<div id="searchResults">  // Don't use camel-case for ids
<span class="title_wrapper"> // Don't use underscore
```

Always use double quotes (") to border attributes.

Do this:

```
<div id="searchResults">
```

Not this:

```
<div id='searchResults'>
```

1.2 Javascript

- Use 4 space indents (spaces, no tabs)
- Must pass JSLint. Meaningful defaults for JSLint is

  ```
  /*jslint vars: true, plusplus: true, devel: true, nomen: true, indent: 4, maxerr: 50 */
  /*global $ */
  ```

  Note: The above recommendation has one caveat.

  JSLint warns about lines consisting entirely of whitespace, but we ignore those warnings. The JSLint feature built into Brackets filters out these warnings automatically.

Note: JSHint instead? we might configure it with a single .jshintrc file

---

- **Line length 79** characters with a soft limit of 84 if absolutely necessary. Try to avoid too deeply nested code by cleverly placing break, continue and return statements.
- General Naming and Syntax

  Variable and function names use camelCase (not under_scores):

  Do this:

  ```
  var editorHolder;
  function getText();
  ```

  Not this:

  ```
  var editor_holder; // Don't use underscore!
  function get_text(); // Don't use underscore!
  ```

  Never assign multiple variables on the same line.

  Don't do this:

  ```
  var a = 1, b = 'foo', c = 'wtf';
  ```

- Private variables

  Use _ prefixes on private variables/methods: Do this:

  ```
  var _privateVar = 42;
  function _privateFunction()
  ```

  Not this:

  ```
  var privateVar = 42; // Private vars should start with _
  function privateFunction() // Private functions should start with _
  ```

- Arrays and Objects

  Use *[]* to assign a new array, not *new Array()*.

  Use *{}* for new objects, as well.

  Two scenarios for *[]* (one can be on the same line, with discretion and the other not so much):

  ```
  // Okay on a single line
  var stuff = [1, 2, 3];

  // Never on a single line, multiple only
  var longerStuff = [
      'some longer stuff',
      'other longer stuff'
  ];
  ```

- Working with jQuery

  Use *$* prefixes on variables referring to jQuery objects:

  Do this:

```
var $sidebar = $("#sidebar");
```

Not this:

```
var sidebar = $("#sidebar"); // Use '$' to prefix variables referring to jQuery objects
```

- **Use semicolons:** Do this:

```
var someVar;
someVar = 2 + 2;
```

Not this:

```
var someVar   // Missing semicolon!
someVar = 2 + 2   // Missing semicolon!
```

- **Operators** Always use === for comparison. The only exceptions are when testing for *null* and *undefined*

```
if (value !== 0) {
    console.log('value can not be undefined');
}
```

Try to avoid ternary, especially if it would use multiple lines:

This is OK:

```
return user.isLoggedIn ? 'yay' : 'boo';
```

Not this:

```
var foo = (user.lastLogin > new Date().getTime() - 16000) ? user.lastLogin - 24000 : 'wut';
```

- **Quoting** Use double quotes in JavaScript. If a JavaScript string literal contains code within it, use single quotes within the string to avoid escaping.

Do this:

```
var aString = "Hello";
someFunction("This is awesome!");

var htmlCode = "<div id='some-id' class='some-class'></div>";
```

Not this:

```
var aString = 'Hello'; // Use double quotes!
someFunction('This is awesome!'); // Use double quotes!

var htmlCode = '<div id="some-id" class="some-class"></div>'; // Use double quctes!
var htmlCode = "<div id=\"some-id\" class=\"some-class\"></div>"; // Within string, use sing
```

- **Commenting** All comments should be C++ single line style

```
//comment.
```

Even multiline comments should use // at the start of each line

Use C style /* *comments* */ for notices at the top and bottom of the file

Annotations should use the /** *annotation* */ style

```
/** This is my function

@param arg1 string The first argument
```

```
    @return boolean
    */
    var myFunc = function (arg1) {
        return true;
    };
```

Annotate all functions

1.3 CSS

Use Less

1.4 Python

Use the Pocoo style guide

In addition:

- Lint/PEP-8 compliance (Use Pylint)

CHAPTER

TWO

DJANGO GUIDELINES

2.1 Project skeletons

The recommended project skeleton to be used for django projects can be found in our django-skel2 repo.

Note: Requires django 1.4 for project creation.

The following commands will start a new project with some feature-rich settings in *YOUR_PROJECT_NAME* directory.

2.1.1 For normal django projects

To use it simply run the following command when starting a new project:

```
django-admin.py startproject --template https://github.com/pulilab/django-skel2/zipball/pulilab --ext
```

2.1.2 For appengine projects

To be developed.

2.2 Structure

Warning: Likely, as time goes by, some other programmers will have to read and understand your code. As a result, try to follow these guidelines as well as you can!

2.3 Testing

2.4 Deployment

Use fabric. There are pre-written fabric script in our project templates.

2.5 Blogs to follow

- The django community aggregator
- Our collection of django related links
- Djangopackages' RSS feed

CHAPTER

THREE

WEB FRONTEND GUIDELINES

3.1 Frontend Tooling

We recommend using lineman

It offers several handy features for front-end development:

- Browser auto-reloading on file changes
- Immediately compile CoffeeScript, Less, and client-side templates as you edit source files
- Provide a development server for fast feedback
- Concatenate & minify all your CSS & JavaScript for production
- Run specs on demand with *lineman spec* using Testem
- Run specs with output suitable for your CI server using *lineman spec-ci*

3.1.1 Installation

```
npm install -g https://github.com/pulilab/lineman/zipball/master
```

This installs the *lineman* command.

For browser auto-reloading, you should install a livereload extension

3.1.2 Usage

To start a new project

```
lineman new <project_name>
```

This generates our preferred directory structure in the *app* directory.

To serve it for your browser run

```
lineman run
```

To run your tests run

```
lineman spec
```

Handlebar templates

Lineman supports underscore and handlebars templates. Handlebars templates should have one of the following file extensions:

- .hb
- .handlebar
- .handlebars

3.1.3 Troubleshooting

The generated and served files are in the *generated* directory. If you have some mysterious problem, you should check out the generated files first.

3.2 Testing

Use lineman's builtin features. See the Tooling section above for details.

3.3 Javascript libraries

We recommend using one of the following JS frameworks:

- Backbone with Marionette, Relational etc
- Ember

Moreover, we have a continuously growing collection of articles to read.

CHAPTER

FOUR

EDITORS

4.1 Sublime Text 2

4.2 Eclipse + PyDev

CHAPTER

FIVE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*