Lab 3

In this lab you will learn how to set up some coding standards and use linting to enforce them, and practice more of our class-construction knowledge to start building the pieces of our TODO application.

Resources

- Maintainable Javascript[1] very good, unfortunately not free, book. Optional.
- ESLint rules[2]
- Code conventions[3]

Linting Basics

Code debugging is hard enough as it is, having ambiguous code just makes it harder. Coding standards enforce some practices. Some are pure formatting issues, others affect how you use the code.

If you are looking for a good book on the subject, Maintainable Javascript[4] is a very good book (alas, not free, and completely optional).

Here are some examples of "coding style":

- Always use 3 spaces for each level of indentation.
- Do not use tabs for indentation. (tab settings might be different on someone else's computer, so your code might not look like you intend it to)
- Always use braces for if/else/for etc, even if they contain only one statement.
- Put open braces on the same line as the construct that precedes them.
- Always put a space before the open brace in function definitions.
- Never put a non-boolean result as the test in an "if".
- Do not do assignments in the test part of an "if".
- Use semicolons at the end of all statements except for function declarations and conditionals.
- Always declare variables before using them.
- Declare all variables at the top of the function that marks their scope.
- Combine `var` statements.
- Use appropriate variable names.
- Camel-case variable names. Do not use underscores.
- Use appropriate verbs in the name to descrite what methods do (has/is for booleans, get/set for setters, new for constructors etc)

[1] http://shop.oreilly.com/product/0636920025245.do
[2] http://eslint.org/docs/rules/
[3] http://eslint.org/docs/developer-guide/code-conventions.html
[4] http://shop.oreilly.com/product/0636920025245.do

- Always capitalize constructors and class objects.
- Never use == (except when comparing to null, and that's only if you want to capture the case of undefined). Use === instead.
- Use blank lines to separate code in contextual blocks
  - inbetween methods
  - before each control statement within a method
  - between variable declarations at top of function
- Keep the complexity of your methods below a limit (complexity is how many different branches your code can take within the method, based on conditionals)
- Use double-quotes for strings (or single quotes, but fix one convention)
- Code should not exceed "this many" characters per line.
- Use "proto" as the name for the prototype object when using the Object.create form.
- Use "double-slash" comments for single line inline comments.
- Use "slash star" comments for multi-line inline comments.
- Indent inline comments the same way as surrounding code.
- Add inline comments only where they clarify the code.
- Use "slash star star" comments for documentation comments before each function.
- Never use ++ or --. Use += 1 instead.
- Avoid using continue, using a condition instead.
- Avoid for–in loop in favor of Object.keys.
- If you use for–in loop, always include a hasOwnProperty check.
- Do not put space between function name and the open parenthesis in a call.
- Always put spaces around the parentheses in control-flow blocks (if, while etc).
- Always put spaces around operators.
- Always wrap immediate function invocations in parentheses.
- Avoid eval and Function.

We will be using ESLint[5], which can automatically check your code for many of the above types of things, and it is fairly customizable. You can see a list of all the rules[6] it will look for, and how you can set them up. Clicking through each rule will tell you more about it.

You can even add your own rules, but we will not do so here.

When you check out Lab3, you will find in the root directory a file .eslintrc. This contains directives for ESLint, take a moment to look through it and look the corresponding rules up[7].

The provided files in Lab3 already contain some "errors" in there. You should be able to see a report of those errors by running "eslint fileName". Make sure to fix those errors before continuing with the rest of the assignment.

[5] http://eslint.org/
[6] http://eslint.org/docs/rules/
[7] http://eslint.org/docs/rules/

Basic steps

1. You should have already decided which of the two partner's GitHub account to use.
2. If you have not switched roles with your partner for a while, this is a good time to do so.
3. You need to first bring your version of the lab up-to-date by fetching from the remote that is linked to MY project. This is a bit tedious.
    - Use "git remote –v" to see the remote branches. There should be one named mainSource and pointing to skiadas/WebAppsLabs.
    - We will start by fetching the newest version of that, with: "git fetch mainSource". It should tell you about a new branch, Lab3.
    - Create a new branch with "git checkout –b Lab3 ––track mainSource/Lab3".
    - Push this repository to your fork, by "git push origin Lab3".
    - Set your local branch to in the future update your repository by: "git branch ––set–upstream Lab3 origin/Lab3".
    - To make sure this is set up properly, do "git branch –vv". You should see [origin/Lab3] next to the Lab3 branch line.
4. Check out the correct branch, via git checkout Lab3.
5. Open the README.md file there, it will contain the instructions on what you need to do.
6. When you are ready to submit simply email me a link to your project and the SHA of the commit that contains your final submission.
7. You should use the issues page to track your progress. Create a new label called "Lab3" and use it to tag all issues related to this lab. I will review those issues to look at your work.
8. Needless to say it, but you are NOT allowed to look at other people's forks of the project and their issues/solutions.
9. I expect you to do the coding using pair programming.