6.170 Recitation 1 - 9/4/2014

Prerequisites

1. Have Git installed and configured. (See: https://help.GitHub.com/articles/set-up-git)
2. Sign-up for a GitHub account. (http://www.github.com)

Goals of this Tutorial

1. Learn about the Chrome debugger and how it can help you debug your application
2. Create a Github repository and learn about how to push/pull from it
3. Begin discussing some of the core language features of Javascript, and how we can use them to our advantage.

Chrome Debugger

Access via CTRL + SHIFT + I (CTRL + SHIFT + J to go straight to the Javascript console). 3 main components that we'll be dealing with:

1. Elements tab: shows the structure of the content of the web page
2. Network tab: shows the communication between the browser and the outside world
    a. We'll talk more about this in later recitations
3. Javascript console: shows the console output of Javascript running on the page (via console.log and console.err, for example). Also provides an REPL with the environment of the loaded page.

The full documentation can be found here: https://developer.chrome.com/devtools. We cover approximately the first half of the document.

Git and Github

Git is a distributed version control system, the basics of which you should be familiar with from 6.005. Many large projects use Git because of its distributed nature, ease of collaboration (especially when facilitated with a service such as Github), fast branching, and other features. Today, we will just cover the basic workflow of Git, enough for completing and submitting the first few assignments.

Ensure that you have Git installed -- you may check this simply by running `git` at the command prompt. If you get a message telling you how to use the Git commands (e.g. "add", "branch", …), then you're good to go! Make sure you also have an account at github.com.

We cover the first 2 chapters of the *Pro Git* book, which is available online. We'll briefly touch on topics from chapters 3-5, but you don't need to worry about the inner workings of the system for now.

http://git-scm.com/book

Javascript

Javascript is the language you will all become extremely familiar with. You will write Javascript code for both the web framework Node.js as well as for the browser. We will have a number of lectures covering the features and best practices of Javascript. The next few recitations will also help you get accustomed to writing Javascript for both the browser and the server.

We start with a simple example found here:
https://github.com/kongming92/cloud-to-butt/blob/master/Source/content_script.js

Note that, at first glance, the syntax of Javascript is very similar to that of Java, which you should have experienced. But there are also some key differences. We'll just observe some interesting points here:

- Variables types are not declared, although Javascript has a number of variable types.
- Note the `var` keyword. We discuss how the *scope* of variables differs from that of Java. Variables preceded with `var` have *function scope*; others have *global scope*.
- Functions can be assigned to variables! This is a very important feature of Javascript known as *first-class functions.* We'll talk about this a bit more, time permitting, and this will form the basis of the things that we talk about in lecture for the next week!

What is this closure magic, and why should I care?

(Only if time permits. Note: we will go into much more detail about these topics in lecture next week. This is just to give a quick taste for what is to come)

Let's write a light switch flipper.

```
var state = "now off";
var flipLightSwitch = function() {
  if (state === "now off") {
    state = "now on";
  } else {
    state = "now off";
  }
  return state;
}
```

Try running this in the Chrome console. See what happens.

Can we do better? What's bad about this example?

- We want to provide an interface in which the *only* thing that can be done is call `flipLightSwitch`. We don't want to give the caller direct access to the `state` variable.

- Notice how currently, you can access the `state` variable from the console directly!

Consider this version now:

```
var flipLightSwitch = (function() {
  var state = "now off";
  return function(){
    if (state === "now off"){
      state = "now on";
    } else {
      state = "now off";
    }
    return state;
  };
})();
```

Run this in the Chrome console as well -- and observe the behavior.

So what's going on here?

Let's first look at the outer function.

- The state variable is defined *inside* the function. *We do not have access to the variable outside the function*. Try this in the console -- and watch it fail.
- The outside function *returns a function*!
- Notice the parentheses around the outside function. This immediately executes the function. So the value of `flipLightSwitch` is the inside function.
- But notice that the inside function makes references to the `state` variable. After the outside function returns, the `state` variable is *kept alive, but can only be accessed through the* `flipLightSwitch` *function.*