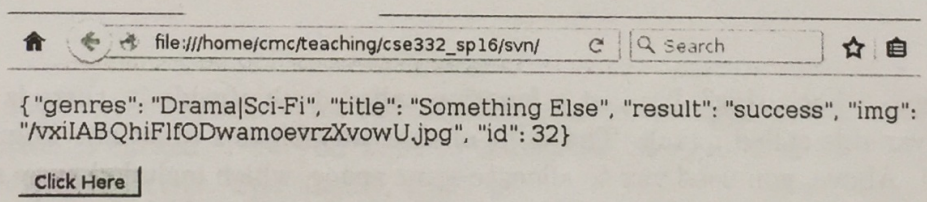
Daily 14 - JavaScript 3

Due March 30th, 2016
CSE332 Programming Paradigms, Spring 2016

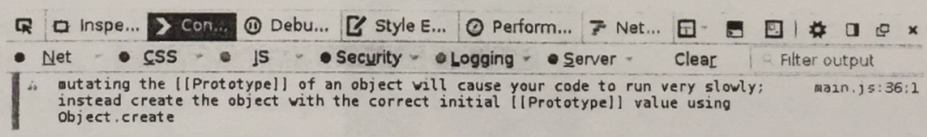
Overview Update your code from daily 13 to use the "approved" object creation process, the **new** operator. Last time you used the `__proto__` property to create inheritance on the fly. Also, put your object definitions into your own JavaScript library called "paradigms". Finally, add an asynchonous call to the movie web service. You will need:

1. An empty HTML template page as with D13.
2. Two JS scripts. One, called `paradigms.js` that contains:
   (a) **Function definitions** (NOT vars like you did before) for two objects: `Label` and `Button`. Leave `Item` the same with the var definition as before.
3. Another, called `main.js` in which you:
   (a) Use the **new** operator to create objects from the `Label` and `Button` functions to create a label and button that look the same as the previous assignment.
   (b) Modify `changeText(args)` so that it no longer gets the text as a parameter input, and instead gets the text from an asynchronous `GET` to `student02.cse.nd.edu:40001/movies/32`. The list `args` should be expected to be `args[0] = label`. The `args[0]` label should be the `Label` object you created.

Here is an example **after** clicking the button:



Discussion Last time you used `__proto__` to set the prototype of your `Label` and `Button` objects on the fly. That is a great demonstration of "prototypical inheritance." But you may have noticed a message like this on your developer console:



It is slow to use `__proto__` for two reasons. First, the interpreter must copy the code from the inheritee object to the inheritor. But second and more importantly, the interpreter only

does property access optimizations when the object is created – if you change the properties through on-the-fly inheritance, the optimizations are wiped out.

So the warning message is gently encouraging you to, whenever possible, create the object with the proper inheritance to begin with. The warning message here suggests using `Object.create`. That's a good option, but so is the `new` operator. In fact, `Object.create` is just a wrapper for `new` and `prototype`. Let's take a look at our "duck" code from before:

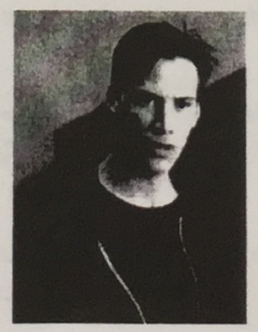
```
var bird = {
    fly: function() {
       alert("away we go!");
    }
}
var duck = {
    quack: function() {
       alert("quacking");
    }
}
duck.__proto__ = bird
duck.quack()
duck.fly()
```

In the above code, `duck` is declared as an object using `var`. Now take a look at this:

```
function duck() {
    this.quack = function() {
       alert("quacking");
    };
}
```

Well now... what's this? It's just a function called `duck`. Inside it, there is a statement that sets a variable called `quack`. The value of that variable is a function. Think about this for a second. Above, you used `var` to allocate some space, which included some space named "quack" that referenced a function. Below, you have a function which does almost the exact same thing. What's the difference? Basically nothing! Functions are just data that has been marked executable!

Here's a picture of Keanu Reeves. Feel free to write "whoa" next to it:



Now you could run the function `duck` above, but nothing would appear to happen. Or you can use the `new` operator to make an object out of the function. Read this for a second:

```
donald = new duck();
donald.quack()
```

What's happening here is that `new` allocates space for an object, and runs `duck` "inside" of that object. In effect, the `duck` function acts like a constructor. Now `donald` is a duck and you can run the function `quack` inside of `donald`. But what if you want `duck` to inherit from `animal`? That's where `prototype` comes in. `prototype` is a property of the *function* `duck`. So when you use `new` on it, `new` "knows" to make the new object inherit from the prototype. Check it out:

```
duck.prototype = animal
donald = new duck();
donald.quack()
```

Now make all that work for `Item`, `Button`, and `Label` in your code. Remember to move your function declaration code (these three definitions) to a new file called `paradigms.js` and import that JS file in your `index.html`. Then take a break and imagine yourself at the end of a peaceful summer afternoon, watching the wind silently make shapes in the clouds. What do you see? Here's some gentle whitespace:

Ah, well now for the asynchronous call. In D13, you set the label text to equal your name when the button is clicked. Now you want to change the label text to be a string response from the movie web service. Specifically, a `GET` to `http://student02.cse.nd.edu:40001/movies/32`. To do this, use the `XMLHttpRequest` JavaScript object.

There are three things you need to handle right now. Actually, there are many more, but we are ignoring error checking for now. The three things are: 1) setting up the asyncronous call, 2) creating a handler for when the response comes back from the server, and 3) issuing the call.

So here are some clues. Use the function `open` in `XMLHttpRequest` to set up the asynchronous call. Set the `onload` property to a handler function, and remember closures: do all your work for this inside `changeText()`. The handler should set the label text equal to the response text from the server. Finally, use `send` in `XMLHttpRequest` to issue the call.

Turn-in Remember that for full credit you need to structure your code properly. You need 1) an HTML file with just the HTML elements and import code, and 2) a Javascript file with the interaction code. The Javascript should go in `scripts`. Upload your code to your student web space in a different obscured folder in the same way that you did for daily 13.

Package your source code into a `zip` or `tar.gz` archive and submit it to `prog.paradigms.secN.sp16@gmail.com` along with the URL to your page.