Computer Science 50
Fall 2011
Scribe Notes

Week 10 Monday: November 7, 2011
Andrew Sellergren

Contents

Computer Science 50
Fall 2011
Scribe Notes

Week 10 Monday: November 7, 2011
Andrew Sellergren

1 Announcements and Demos (0:00–4:00)

- Sign up now for CS50 Seminars! The "Develop for the Blackbery... Like A Boss," hosted by Jason Hirschorn (pumpkin) and Marta Bralic (cloud).
- If you're interested in helping out with some applications to improve campus life, consider signing up for Hack Harvard.
- This week is all about CS50 Shuttle. This game uses the Google Earth API to allow you to drive the Harvard shuttle around campus, picking up teaching fellows and dropping them off at their destinations. Try typing in the Konami Code into the staff solution.
- This week we will introduce you to JavaScript, a client-side, interpreted programming language. JavaScript is the language that powers the Google Earth API and thus CS50 Shuttle.

2 JavaScript (4:00–54:00)

2.1 HTTP Response Codes

- Recall that along with the response, web servers return an HTTP status code such as one of the following:
  - 401 - unauthorized
  - 403 - forbidden
  - 404 - not found
  - 500 - internal server error

  If we create a new file named `foo.html` in our `public_html` directory, but forget to `chmod` it so that it is world-readable, then navigating to it in our browser will result in a HTTP 403 error. With Firebug's Net tab enabled (similar to Live HTTP Headers), we see that there's a Status column that shows "403 Forbidden" in red.

2.2 Comparison to Other Programming Languages

- No longer do we need a `main` function, as any function can be called within a `script` tag in our HTML.
- Conditions, Boolean expressions, switches, loops are the same in JavaScript as they are in PHP.
- Variables in JavaScript look slightly different than in PHP. As in PHP, variables don't need an explicit type. Instead of the `$`, though, we use the keyword `var`:

```
var s = "hello, world";
```

Implicity, `s` will be a string type.

- Conveniently, arrays can be instantiated using square bracket notation like so:

```
var numbers = [4, 8, 15, 16, 23, 42];
```

- The `foreach` construct from PHP is equivalent to the following in JavaScript:

```
for (element in array)
{
    // do this with element
}
```

- One of the biggest selling points of JavaScript is a technology called Ajax which will delve into later today. Ajax makes use of JavaScdript Object Notation (JSON) to communicate with a web server. Objects in JavaScript are implemented as follows:

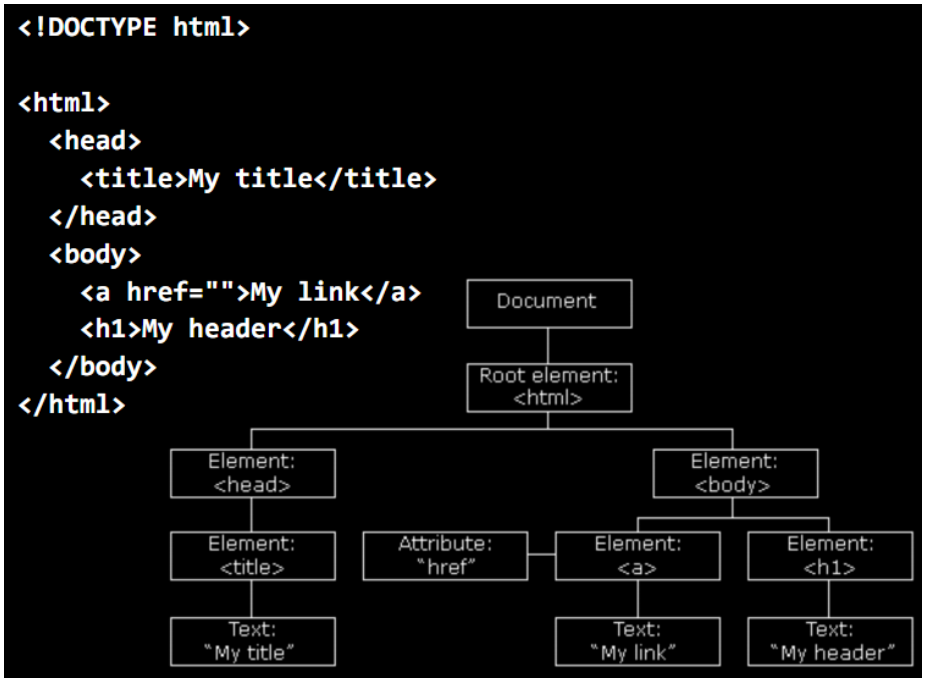
```
var student = {
  id: 123,
  house: "Mather House",
  name: "David Malan"
};
```

Notice that JavaScript objects are functionally similar to C structs, but are syntactically simpler. By itself, a single object like this might not be compelling, but consider an array of objects:

```
var students = [
  {
    id: 123,
    house: "Mather House",
    name: "David Malan"
  },
  {
    id: 456,
    house: "Mather House",
    name: "Tommy MacWilliam"
  }
];
```

Using this syntax, we don't need a new type or even specify a name for objects that we create on the fly.

- Recall from a few weeks ago the concept of the Document Object Model (DOM), a tree structure used to represent the HTML of webpage:



  When a browser downloads an HTML document, it parses it hierarchically as this diagram would suggest. If an open tag is encountered within another open tag, the second tag is considered to be a child of the first tag. This tree structure can be very easily traversed and manipulated using JavaScript.

- To begin to comprehend the power of JavaScript, consider the Facebook homepage. When you first login and view your News Feed, the Facebook web server is only able to send to your browser the updates that have occurred prior to your login. And yet, if you hang around long enough,[1] you'll see status updates pop up periodically. If you haven't refreshed your page, then how has Facebook sent you these updates? Using JavaScript.

2.3 Events

- Think back to Week 0 when we saw events in the context of Scratch. We implemented a simple game of Marco Polo by having one sprite broadcast an event—saying "Marco"—which another sprite listened for and responded to—saying "Polo." So too in JavaScript do we have the notion

[1] Very, very long in David's case.

of events. In fact, each of the HTML elements we've examined so far has some number of events associated with it. If we want to respond to these events, we can implement *event handlers*, some of which are named below:

- `onblur`
- `onchange`
- `onclick`
- `onfocus`
- `onkeydown`
- `onkeyup`
- `onload`
- `onmousedown`
- `onmouseup`
- `onmouseout`
- `onmouseover`
- `onmouseup`
- `onresize`
- `onselect`
- `onsubmit`

If you want to implement a popup that displays when a user hovers over an image, you might use the `onmouseover` handler, for example. You can also detect keystrokes, page loads, form submissions, and more. With JavaScript, you can do almost anything![2]

2.4 Form Validation

2.4.1 `form1.html`

- `form1.html` demonstrates a standard HTML login form that has no client-side validation:

```
<!--

form1.html

A form without client-side validation.

Computer Science 50
David J. Malan
```

[2] Anything at all.

```
-->

<!DOCTYPE html>

<html>
  <head>
    <title>form1</title>
  </head>
  <body>
    <form action="dump.php" method="get">
      Email: <input name="email" type="text">
      <br>
      Password: <input name="password1" type="password">
      <br>
      Password (again): <input name="password2" type="password">
      <br>
      I agree to the terms and conditions: <input name="agreement" type="checkbox">
      <br><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

That is, a user can input almost anything as his e-mail and password and the form will submit to the back end. Of course, the back end might have some validation built in so that if a malformed e-mail address is passed, the form submission will fail and the user will be bounced back to the input page. It would be nice, however, if instead of going through the trouble of clicking Submit and waiting for the HTTP response indicating a submission error, the user could be warned that his inputs are wrong. This is where JavaScript comes in.

2.4.2 `form2.html`

- `form2.html` introduces the client-side validation that we just discussed:

```
<!--

form2.html

A form with client-side validation.

Computer Science 50
David J. Malan

-->
```

```
<!DOCTYPE html>

<html>
  <head>
    <script>

        function validate()
        {
            if (document.forms.registration.email.value == "")
            {
                alert("You must provide an email adddress.");
                return false;
            }
            else if (document.forms.registration.password1.value == "")
            {
                alert("You must provide a password.");
                return false;
            }
            else if (document.forms.registration.password1.value !=
                     document.forms.registration.password2.value)
            {
                alert("You must provide the same password twice.");
                return false;
            }
            else if (!document.forms.registration.agreement.checked)
            {
                alert("You must agree to our terms and conditions.");
                return false;
            }
            return true;
        }

    </script>
    <title>form2</title>
  </head>
  <body>
    <form action="dump.php" method="get" name="registration"
      onsubmit="return validate();">
      Email: <input name="email" type="text">
      <br>
      Password: <input name="password1" type="password">
      <br>
      Password (again): <input name="password2" type="password">
      <br>
      I agree to the terms and conditions: <input name="agreement" type="checkbox">
```

```
      <br><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Here we see that a new attribute of the `form` tag is defined: `onsubmit`. As you might guess, it controls what happens when the form is actually submitted. In this case, we're calling a JavaScript function named `validate`. The `onsubmit` attribute, according to the W3C's specification of HTML, can take actual JavaScript code as its value. If this code evaluates to true, then the form will be submitted. Otherwise, the form will not submit. We're relying on the `validate` function to evaluate to true if the form inputs are valid and to evaluate to false otherwise.

- At the top, in the `head` element, we have our `script` tag.[3]
- Within `validate`, the first if condition checks for a blank e-mail field. We do this by accessing a global object named `document` that is provided to us natively in JavaScript. `document` is actually a kind of object or struct which contains the entire hierarchy of the page's content. You can infer from these if conditions that the `document` object has within it an object named `forms` which contains all of the page's `form` elements. Within `document`, we access `forms` followed by the `registration` form in particular. Notice that `registration` corresponds to the value we gave to the `name` attribute of our form. Finally we access the `value` attribute of the `email` field. If this value is the empty string, then we call a built-in function called `alert` which pops up a window. After this window pops up, we return false, which is important to ensure that the form doesn't actually submit.
- In the next conditions, we check for a blank password field and for non-matching password fields. Then we access the `checked` property of the checkbox field to make sure that it has been clicked. Note that we can use the `==` operator to compare strings, unlike in C.
- Question: do you need to specify the return type of a JavaScript function? No. It is loosely typed, just like PHP. In fact, you can return multiple different types within the same function, if you want.
- Question: could `else if` be replaced by `if` in the logic above? Yes, because each if condition has a return statement within it, so multiple conditions will never be executed.

[3]Incidentally, the `script` tag can be placed elsewhere on the page other than in the `head` element. There are good reasons one might do so and we'll see examples of it later in the course.

- Question: when `validate` returns false, is the form cleared? No, it is left as is, another advantage of doing client-side validation.
- So why bother with server-side validation if client-side validation is so simple and elegant? As it turns out, users can disable JavaScript in almost every major browser with a few clicks of the mouse. If a malicious user were to do this, he could get past all your client-side validation. Thus if you have no server-side validation, he could have a field day with your form. Just as importantly, not every browser fully supports JavaScript—the BlackBerry browser being a good example. When you're developing a website, then, you need to consider what users you might be alienating if you choose to implement functionality which absolutely requires JavaScript.
- Still, client-side validation is very compelling for its speed. These days, many users are interacting with websites on mobile devices which don't always have super fast internet connections. Providing users with quick feedback is all the easier if you don't have to make unnecessary calls to the server.
- Question: is there a way to detect if the user has disabled JavaScript? Yes. You might, for example, attempt to assign a cookie to a user and later check if the assignment succeeded. If it didn't, it might be because the user has JavaScript disabled.

2.4.3 `form3.html`

- `form3.html` improves upon `form2.html` by simplifying the validation code:

```
<!--

form3.html

A form with client-side validation demonstrating "this" keyword.

Computer Science 50
David J. Malan

-->

<!DOCTYPE html>

<html>
  <head>
    <script>

        function validate(f)
```

```
            {
                if (f.email.value == "")
                {
                    alert("You must provide an email adddress.");
                    return false;
                }
                else if (f.password1.value == "")
                {
                    alert("You must provide a password.");
                    return false;
                }
                else if (f.password1.value != f.password2.value)
                {
                    alert("You must provide the same password twice.");
                    return false;
                }
                else if (!f.agreement.checked)
                {
                    alert("You must agree to our terms and conditions.");
                    return false;
                }
                return true;
            }

        </script>
        <title>form3</title>
    </head>
    <body>
        <form action="dump.php" method="get" onsubmit="return validate(this);">
            Email: <input name="email" type="text">
            <br>
            Password: <input name="password1" type="password">
            <br>
            Password (again): <input name="password2" type="password">
            <br>
            I agree to the terms and conditions: <input name="agreement" type="checkbox">
            <br><br>
            <input type="submit" value="Submit">
        </form>
    </body>
</html>
```

This incarnation of `validate` takes a single argument instead of none. The argument that is passed to `validate` is the actual form object itself. We achieve this by writing `validate(this)` in the `onsubmit` attribute. The value of `this` will change from context to context. Here, it stands for

the form object.

2.4.4 `form4.html`

- `form4.html` demonstrates a clever use of the `disabled` property:

```
<!--

form4.html

A form with client-side validation demonstrating disabled property.

Computer Science 50
David J. Malan

-->

<!DOCTYPE html>

<html>
  <head>
    <script>

        function toggle()
        {
            if (document.forms.registration.button.disabled)
                document.forms.registration.button.disabled = false;
            else
                document.forms.registration.button.disabled = true;
        }

        function validate()
        {
            if (document.forms.registration.email.value == "")
            {
                alert("You must provide an email adddress.");
                return false;
            }
            else if (document.forms.registration.password1.value == "")
            {
                alert("You must provide a password.");
                return false;
            }
            else if (document.forms.registration.password1.value !=
                     document.forms.registration.password2.value)
            {
```

```
                alert("You must provide the same password twice.");
                return false;
            }
            else if (!document.forms.registration.agreement.checked)
            {
                alert("You must agree to our terms and conditions.");
                return false;
            }
            return true;
        }

    </script>
    <title>form4</title>
  </head>
  <body>
    <form action="dump.php" method="get" name="registration"
      onsubmit="return validate();">
      Email: <input name="email" type="text">
      <br>
      Password: <input name="password1" type="password">
      <br>
      Password (again): <input name="password2" type="password">
      <br>
      I agree to the terms and conditions: <input name="agreement"
        onclick="toggle();" type="checkbox">
      <br><br>
      <input disabled="disabled" name="button" type="submit" value="Submit">
    </form>
  </body>
</html>
```

In this version of our login form, the Submit button isn't clickable until the checkbox has been checked. We achieve this by initially setting the `disabled` attribute of the checkbox to the value `disabled`.[4] Then, we assign a JavaScript function `toggle` to be the listener for the click event (via the `onclick` attribute) on the checkbox. Whenever the checkbox is checked, the `toggle` function will be called. If the checkbox is checked, the `disabled` attribute of the Submit button will be set to false.

2.4.5 `form5.html`

- `form5.html` makes use of *regular expressions* to check that the user's input matches certain expected patterns:

[4] Yes, it's a stupid convention.

```
<!--

form5.html

A form with client-side validation demonstrating regular expressions.

Computer Science 50
David J. Malan

-->

<!DOCTYPE html>

<html>
  <head>
    <script>

        function validate()
        {
            if (!document.forms.registration.email.value.match(/.+@.+\.edu$/))
            {
                alert("You must provide a .edu email adddress.");
                return false;
            }
            else if (document.forms.registration.password1.value == "")
            {
                alert("You must provide a password.");
                return false;
            }
            else if (document.forms.registration.password1.value !=
                     document.forms.registration.password2.value)
            {
                alert("You must provide the same password twice.");
                return false;
            }
            else if (!document.forms.registration.agreement.checked)
            {
                alert("You must agree to our terms and conditions.");
                return false;
            }
            return true;
        }

    </script>
    <title>form5</title>
  </head>
```

```
  <body>
    <form action="dump.php" method="get" name="registration"
      onsubmit="return validate();">
      Email: <input name="email" type="text">
      <br>
      Password: <input name="password1" type="password">
      <br>
      Password (again): <input name="password2" type="password">
      <br>
      I agree to the terms and conditions: <input name="agreement" type="checkbox">
      <br><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

In JavaScript, many variable types have built-in methods. Strings, for example, have a built-in `match` method that returns true if the string matches the particular regular expression, or pattern, provided as input. Here we're checking if the user-inputted e-mail address contains an `@` symbol and ends in `.edu`. The `.` character is a wildcard and `+` denotes "one or more," so `.+` translates to "one or more of any character." The `\.` means a literal dot character: the backslash escapes it so as to distinguish it from the wildcard character. The `$` signifies "ends with," i.e. that the `.edu` comes at the absolute end of the string. You can use `^` to signify "starts with." Of course, this regular expression isn't all that rigorous since many invalid e-mail addresses will match it, but at least it's a start. There are libraries available that do this kind of e-mail address validation right out of the box. Be careful, though: you don't want to use a regular expression that is so strict as to exclude e-mail addresses that have subdomains, as some forms do. David, for one, has experienced this when trying to register with his `post.harvard.edu` e-mail address.

2.4.6 `form6.html`

- `form6.html` introduces the jQuery library:

```
<!--

form6.html

A form with client-side validation demonstrating jQuery.

Computer Science 50
David J. Malan

-->

<!DOCTYPE html>

<html>
  <head>
    <script src="http://code.jquery.com/jquery-latest.js"></script>
    <script>

        $(document).ready(function() {
            $("#registration").submit(function() {
                if (!$("#email").val().match(/.+@.+\.edu$/))
                {
                    alert("You must provide a .edu email adddress.");
                    return false;
                }
                else if ($("#password1").val() == "")
                {
                    alert("You must provide a password.");
                    return false;
                }
                else if ($("#password1").val() != $("#password2").val())
                {
                    alert("You must provide the same password twice.");
                    return false;
                }
                else if (!$("#agreement").attr("checked"))
                {
                    alert("You must agree to our terms and conditions.");
                    return false;
                }
                return true;
            });
        });

    </script>
    <title>form6</title>
```

```
  </head>
  <body>
    <form action="dump.php" id="registration" method="get">
      Email: <input id="email" name="email" type="text">
      <br>
      Password: <input id="password1" name="password1" type="password">
      <br>
      Password (again): <input id="password2" name="password2" type="password">
      <br>
      I agree to the terms and conditions:
      <input id="agreement" name="agreement" type="checkbox">
      <br><br>
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

jQuery is a third-party JavaScript library that greatly simplifies the syntax for many common operations in the language. As the example above demonstrates, jQuery makes the accessing of DOM nodes and form fields much less tedious. To begin using jQuery, we link to it by providing its URL as the `src` attribute of a `script` tag. We can actually navigate to this URL in our browser and view the source code that we're importing. Many commonly used JavaScript features come built in to jQuery so that when you want to use something like autocomplete, you don't have to implement it yourself.

- The first thing to notice in `form6.html` is that our JavaScript is no longer intermingled with our HTML. All of the function calls are within the `script` tag. This is good design because it allows us to stay organized and better understand what's going on in our code. Before this `script` tag in which we write our own JavaScript, we must import jQuery in a separate `script` tag so that its functions will be available to us.

- To access the DOM using jQuery, we write `$(document)` instead of just `document`. This allows us to make use of some methods that jQuery adds to the plain old `document` object. Here, we're calling the `ready` method of the jQuery `document` object. This method executes by default when the entire page has loaded and is thus "ready." When we call `ready` we pass it a single argument which takes the form of a *lambda function*. The syntax `function() {...}` defines a function without a name, otherwise known as a lambda function or an anonymous function. Since we're never going to call this function in any other context, we don't need to give it a name that we can reference later. By passing this lambda function to the `ready` method, we tell jQuery to call it when the `ready` method is called, i.e. when the page has fully loaded.

- The HTML in `form6.html` is slightly different from previous incarnations in that we defined the `id` attribute, not the `name` attribute, of the `form` element. Having done so, we can immediately access the `form` element by writing `$("#registration")` using jQuery's syntax. The `form` element that is returned by jQuery has a special `submit` method that is called when this form is submitted. As before with the `ready` method, we pass an anonymous function that we want to be executed when this happens. This anonymous function is what does the actual validation of the user's input. To do this validation, we use the `#` syntax again to grab the form fields and the `val` method to access their values.
- To carry the decoupling of HTML and JavaScript one step further, we can write our JavaScript code in a separate file and import it just as we imported the jQuery source code. In general, this is good design.

2.4.7 `form7.html`

- For our last trick, we'll call one jQuery function to validate our entire form. Play around with it for yourself!

3 Words of Wisdom from Jocelyn Goldfein (54:00–82:00)

- Please welcome Jocelyn Goldfein, Director of Engineering at Facebook! Tune in to the lecture video to hear her words of wisdom.