Hands on with JavaScript

CS106E, Young

In this handout, we describe JavaScript and walk you through how to run a JavaScript program. There is also a JavaScript Language handout that provides an overview of the JavaScript language, teaching you the data types, control structures, and showing how to write functions in JavaScript.

In addition to the handouts, class notes will discuss the various capabilities of JavaScript. For short answer questions, you’ll be responsible for both the class notes and the handouts. As far as our expectations on your ability to write JavaScript goes, you’ll only need this handout and some familiarity with the language as described in the JavaScript Language handout.

Running JavaScript

JavaScript is primarily a client-side language. As we previously discussed programs used in client-side languages are sent to the client’s web browser from the web server, and ultimately they run on the client’s computer. Because of this, we can just load a JavaScript program into our web browser and execute it, exactly as we’ve done with our HTML and JavaScript webpage. We don’t need to interact with the Stanford servers to test our JavaScript.

Debugging JavaScript

As with PHP, if your webpage has a program with JavaScript errors, the program will fail silently, with no indication of what has gone wrong or even that a problem exists. You’ll need to turn on the developer console to get error messages.

The method for displaying the developer console varies from one web browser to another. On Chrome it’s available by going to the menu (accessed from the vertical ellipsis on the far-right, just beyond the bookmark star) and selecting “More Tools” followed by “Developer Tools”.

On Firefox, I can get to it by going to the menu button on the far-right indicated by three horizontal bars (this type of icon is referred to as a hamburger icon in user-interface design) and clicking on the developer wrench icon and then choosing “Developer Toolbar”.

---

1 As we’ll discuss briefly in lecture, JavaScript can be run on a webserver and in other contexts as well. However, the vast majority of its uses is as a client-side language running within a web browser, and that’s what we’ll focus on in this handout.
On both Firefox and Chrome, after you bring up the Developer Tools, you’ll see several tabs running across the top; make sure to choose the one labelled “Console” or “Web Console”.

Keep the Console up at all times when working with JavaScript, otherwise you won’t know if there’s a problem with your JavaScript program.

### Tax Calculator with JavaScript

In the Introduction to PHP / Hands On handout, we built a simple tax calculator. That webpage actually can be done without a web server using just JavaScript running on the client. Here’s what our webpage looks like before and after I click on the “Calculate Tax” button:

Here’s our HTML file. Pay particular attention to the `<input>` elements and the `<script>` tag, both of which we’ll go into in more detail. Also, notice the `<p>` below the input elements.

```html
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8" />
<title>Sales Tax Calculation</title>
</head>
<body>
<h1>Sales Tax Calculation</h1>
<p>Please enter amount of sale:</p>
 Amount: <input type="text" id="amount" />
<input type="button" id="button" value="Calculate Tax" />
<p id="result"></p>

<script>
function calculateTax() {
  var amount = parseFloat(document.getElementById("amount").value);
  var tax = amount * 0.1;
  var total = tax + amount;
  document.getElementById("result").innerHTML = "tax = " + tax + " and total is " + total;
}

document.getElementById("button").addEventListener("click",calculateTax);
</script>
```
Input Elements
As with our PHP, we’ll need a text field and a button to allow the user to initiate the calculation.

Amount: <input type="text" id="amount" />
<input type="button" id="button" value="Calculate Tax" />

You may notice several differences between this and the similar code I had for my PHP tax calculator:

- The <form> tag is gone. A <form> is needed to actually submit data to a web server – as you may recall, the form contained both the location of the PHP program on the server and the method="POST" specifying how the information was to be transmitted to the server. For our client-side code, the program is right in our HTML file and we aren’t sending anything to the server.

- We’ve replaced our type="submit" input element with a type="button". Submit specifically refers to sending a request to a web server. We aren’t doing that here, so instead we’ll have a generic button.

- We’ve given both of our input elements an id. This will allow us to access them from JavaScript.

The <Script> Tag
All our JavaScript is contained between a <script> start tag and a </script> end tag. It’s also possible to load script from a separate external file (similar to how we’ve often loaded our CSS from a separate file). However, for this class, we’ll keep things simple by just sticking our JavaScript directly within the HTML file.

We can also include more than one <script> tag and can include a mix of internal JavaScript and external JavaScript files. This will be useful, for example, if we have some JavaScript that is appropriate for many of our webpages and then some JavaScript that is specific to one page.

Function Definition
We write a function that will be executed when the user clicks on the button.

```javascript
function calculateTax() {
    var amount = parseFloat(document.getElementById("amount").value);
    var tax = amount * 0.1;
    var total = tax + amount;
    document.getElementById("result").innerHTML = "tax = " + tax + " and total is " + total;
}
```

2 You may find <form> tags in client-side code as well. At one point, all form elements (e.g., text fields, push buttons, check boxes) had to be contained within a <form>, but that is no longer the case. Forms can still be used to keep form elements organized on client-side programs.

3 If you’re planning to use the same JavaScript code in several different webpages, you can put it in an external file and add a src property to the <script> tag: <script src="my-javascript.js"></script>.
Notice that as with PHP, JavaScript functions do not specify a return type, so we instead use the keyword “function” where a C or C++ function or a JavaScript method would put the return type.

Let’s break down that first line of the function:

```javascript
var amount =
    parseFloat(document.getElementById("amount").value);
```

As you can probably tell, we’re creating a variable to store the amount the user entered into the webpage.

In order to get that value, we need to use something called the Document Object Model (DOM). Every HTML element on the webpage has a corresponding JavaScript object. Together these JavaScript objects form the Document Object Model. We access the DOM through a special variable called document.

Using the document variable, there are a variety of ways to get to elements on the webpage. For example, we could use CSS style selection rules to access elements. However, one of the simplest ways to get to an element on the webpage is to give an element an id, and then to use getElementById. As you may recall, we created the input text field using:

```
Amount: <input type="text" id="amount" />
```

So we can access it using:

```javascript
document.getElementById("amount")
```

We can retrieve and manipulate different pieces of information about this input element. For example, we could retrieve the length of the text field. However, in this case what we really want to know is what value the user has currently entered into the text field. So we access the value property:

```javascript
document.getElementById("amount").value
```

Finally, text fields on a webpage can contain text or numbers. Unfortunately, we’ll occasionally run into situations where JavaScript isn’t quite sure whether or not to treat something as a number or a string, so we’ll explicitly tell it we want to treat the amount entered as a number, by calling the function parseFloat which converts a string to a floating point number:

```javascript
parseFloat(document.getElementById("amount").value)
```

You can probably figure out the next two lines on your own:

```javascript
var tax = amount * 0.1;
var total = tax + amount;
```

For our last line, we are going to put our results back on the webpage. We’re going to place them within the paragraph, which appears in the HTML file just after the `<input>` elements. Here’s the HTML for this paragraph:

```
<p id="result"></p>
```
Notice the id="result". As I mentioned previously, we’ll use id to retrieve or access HTML elements on our webpage.

Okay, let’s look at the last line of our calculateTax function:

```
document.getElementById("result").innerHTML =
    "tax = " + tax + " and total is " + total;
```

We can see we are accessing that id="result" paragraph using the `document.getElementById`. Previously we were interested in getting the value of our input elements. This time we’re getting to a special property called `innerHTML`. Assigning a value to this property replaces the contents of the element with some new elements. For example:

```
document.getElementById("result").innerHTML =
    "<b>Go <i>Stanford</i></b>";
```

would essentially replace the previously empty paragraph with this one:

```
<p id="result">Go <i>Stanford</i></p>
```

In this case, we’re replacing the contents of the paragraph with a string we form by concatenating various strings with variable values:

```
"tax is " + tax + " and total is " + total
```

So if tax is 10 and total is 110, our paragraph will now look like:

```
<p id="result">tax is 10 and total is 110</p>
```

Assigning our Event Handler

One last step, we need to actually have our calculateTax function execute when the user clicks on the “Calculate Tax” button. You’ll recall we created this button using:

```
<input type="button" id="button" value="Calculate Tax" />
```

We access this button using `getElementById`, and then we call a special method `addEventListener` on the button.

```
document.getElementById("button").addEventListener(
    "click", calculateTax);
```

Add event listener takes two parameters 1) the name of an event that we want to use as a trigger and 2) the function that we want executed when that event occurs.

There’s quite a few events we can write event handlers for, and we’ll take a look at few in lecture.

**Summary of JavaScript Webpage Connection**

Let’s briefly summarize how JavaScript programs connect to webpages:

- We define a JavaScript program for our webpage using the `<script>` tag.
• Within our JavaScript, we can access, manipulate, change, or even replace HTML elements on the webpage through the Document Object Model (DOM). For our class, we’ll do this by giving those elements id attributes in the HTML and then by calling document.getElementById(idString).

• Once we get a hold of an element on the webpage, we’ll need to know what sort of methods or properties are available on that object. In this handout, we’ve used the value property, which gets the contents of <input type="text" /> text fields and the innerHTML property which allows us to completely replace the HTML contained within an element.

• We can assign different functions to run in response to different types of events by calling addEventListener.

Hints for Debugging with JavaScript

As we’ve previously seen, you really need to get the Developer Console displayed. Without it you will be working completely in the dark.

Once you’ve got the Developer Console displayed, you may find the console.log method useful. This is the equivalent of a print statement. It will output whatever you pass in as a parameter out to the Console.

```javascript
var x = 12;
console.log("x is " + x);
```

Limitations of JavaScript

While working with JavaScript can be easier than PHP (because we don’t have to transfer files back and forth for quick testing), do keep in mind that JavaScript does have some important limitations. Because it does not work with the actual web server, applications such as the RSVP website from your homework, or the message board website I showed in class cannot be done with JavaScript. Anything requiring getting information from a user and storing it in a database cannot be done with client-side programming alone. In some cases, client-side programming can enhance these types of websites, but ultimately for many types of websites data must be sent back to a server and processed server-side where it can be stored in a database.