EXERCISE: Introduction to client–side JavaScript

Barend Köbben

Version 1.3
March 23, 2015

Contents

Key points

In this exercise you will discover the basics of using the JavaScript programming language to build dynamic web pages. You will learn:

1. What Dynamic HTML (DHTML) is and what you can do with it;
2. How you can use the JavaScript language to create dynamic HTML;
3. How you can include JavaScript in your webpage;
4. Some basic uses of JavaScript and dynamic HTML.

Note that this will only be a first and very limited introduction to the many possibilities of client–side Javascript...

There are several software tools that can help you create JavaScript in HTML web pages: You can use a simple text–editor, but you are advised to use one that is more intelligent, and gives you line numbers, and automatic highlighting of recognised HTML and JavaScript keywords. For Linux/MacOSX this could be BBedit or TextWrangler, on Windows you can use NotePad++. Or you can use specialised webpage editing software, such as BlueGriffon or Dreamweaver, or an IDE (integrated develompent environment) such as WebStorm. Furthermore, modern web–browsers can provide you with useful error messages, code views of HTML, CSS and JavaScript, network traffic monitoring, etc:

- FireFox: `Tools > Web Developer > Toggle Tools`
- Chrome: `View > Developer > Developer Tools`
- Internet Explorer 8+: [F12] or `Developer Tools`

A useful reference for the JavaScript Language can be found at http://www.w3schools.com/jsref/default.asp.

! → In many cases during exercises, you will have to type code (HTML and JavaScript). It's very easy to make mistakes in such code. Some code (e.g., HTML) is not case–sensitive, but other (e.g. JavaScript) is: the variable `mysomething` is different from the variable `MySomeThing`! Also take care of the *special character* (→) in the code examples we provide:

```
here is some code that should all be typed on 1 line in your→
 file but is divided over 2 lines in our example...
```

→ this character means you **should not** type a `return` or `enter` in this place. The line should be typed **without interruption**, the move to the next line in our example is only because

it would not fit otherwise.

1 Dynamic HTML and scripting

Dynamic HTML (DHTML) is built on top of HTML. That is, DHTML includes all the elements that make up a traditional web page. However, with DHTML all of those elements are now programmable objects. You can assign each element an ID and then use scripting to alter them after the page has been downloaded.

On the web page http://kartoweb.itc.nl/courses/JavaScriptExamples/ you will find an example of how this works and links to examples of what you can achieve with scripting DHTML.

2 The scripting language JavaScript

From the name alone, you might think JavaScript is like the programming language Java. However, the JavaScript language resembles Java in some ways, but does not have its static typing and strong type checking. JavaScript does support most Java expression syntax and basic control-flow constructs.

In contrast to Java's compile–time system of classes built by declarations, JavaScript supports a *runtime* system based on a small number of data types representing numeric, Boolean, and string values. This results in so–called *loose typing*, which means variable data types do not have to be declared specifically: if you assign a `string` value to a variable, it automatically will become a string type variable! JavaScript is interpreted (not compiled) by the client software (usually the browser). The code is integrated with, and embedded in, HTML.

3 Using Javascript in a web page

You can use JavaScript in an HTML document in two ways:

1. Embedded in the HTML file using the `<SCRIPT>` tag;
2. Using an external JavaScript code file.

3.1 Using the `SCRIPT` Tag

The `<SCRIPT>` tag is an extension to HTML that can enclose any number of (Java)Script statements. A document can have multiple script tags, and each can enclose any number of JavaScript statements.

Task 1 : In listing 1 you find a simple script. Create a new, empty, WWW-page in coding mode (e.g., use NotePad++ or use

Dreamweaver and choose the HTML-code editor). Type the JavaScript and HTML code as shown in listing 1 into the HTML-code editor window. Save the page and open it in a web browser: It should display the following in the browser:

This is scripted content!
This is plain HTML...

Notice that there is no difference in appearance between the first line, generated with JavaScript, and the second line, generated with plain HTML. •

Listing 1: A simple scripted page

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
   document.write("This is scripted content!");
</SCRIPT>
</HEAD>
<BODY>
<P>This is plain HTML...
</BODY>
</HTML>
```

Generally, you should define the JavaScript for a page in the `HEAD` portion of a document. That way, all functions are defined before any content is displayed. Otherwise, the user might perform an action while the page is still loading that triggers an event handler and calls an undefined function, leading to an error.

3.2 Using a JavaScript code file

Rather than embedding the JavaScript in the HTML, the `SRC` attribute of the `<SCRIPT>` tag lets you specify a file as the JavaScript source. For example:
`<SCRIPT LANGUAGE="JavaScript" SRC="common.js"></SCRIPT>`
This is especially useful for sharing the same JavaScript code among many different pages. Note the ending `</SCRIPT>` tag is necessary to create a valid HTML script object!

The SRC attribute can specify any URL, relative or absolute. For example: `SRC="http://www.mysite.com/functions/jsfuncs.js"`

External JavaScript files cannot contain any HTML tags: they must contain only JavaScript statements and function definitions. External JavaScript files must have the file name suffix `.js`.

Task 2 : Open a new empty file. Put the JavaScript (only the script!) you made in Task 1 into this file (you can use copy and paste) and save this file. Make sure it's saved using a `.js` extension! Change the Web Page you made in Task 1, so that it calls the JavaScript from a code file, instead of having the code embedded. •

4 JavaScript Objects and the Document Object Model

JavaScript is an object–oriented language, like Python, C++, Java and many others. There are several types of objects you can use:

JavaScript core language objects: these are objects that are part of the JavaScript language. Examples are the `Date` object that can be used to manipulate dates and times, the `String` object for manipulating text strings and the `Math` object for manipulating numbers.

Browser or Document Object Model (DOM) objects: the parts of the browser and web page that are made accessible to the scripting language are structured in the so–called Document Object Model (DOM); this is a structured hierarchical object tree that provides a way to control elements on the page using scripting, provides multimedia controls for animations and other effects, and provides a way to bind data to an HTML page. Unfortunately, the browser manufacturers have not yet agreed on one common DOM, therefore there are small differences in the DOM for different browsers. The scheme in figure 1 shows you the structure of the lowest common–denominator DOM. The DOM objects can be used to manipulate the HTML web page, and turn it into DHTML, as shown in the examples in section 1.

4.1 The `document.write` method

As you saw in the previous example, the `write()` method of the DOM object `document` displays output in the browser, and its syntax follows JavaScripts standard object notation:
`objectName.methodName(arguments)`, where `objectName` is the name of the object, `methodName` is the name of the method, and `arguments` is a list of arguments to the method, separated by commas.

"Big deal," you say, "HTML already does allow me to write text in a web page." But in a script you can do all kinds of things you can't do with ordinary HTML. For example, you can display text

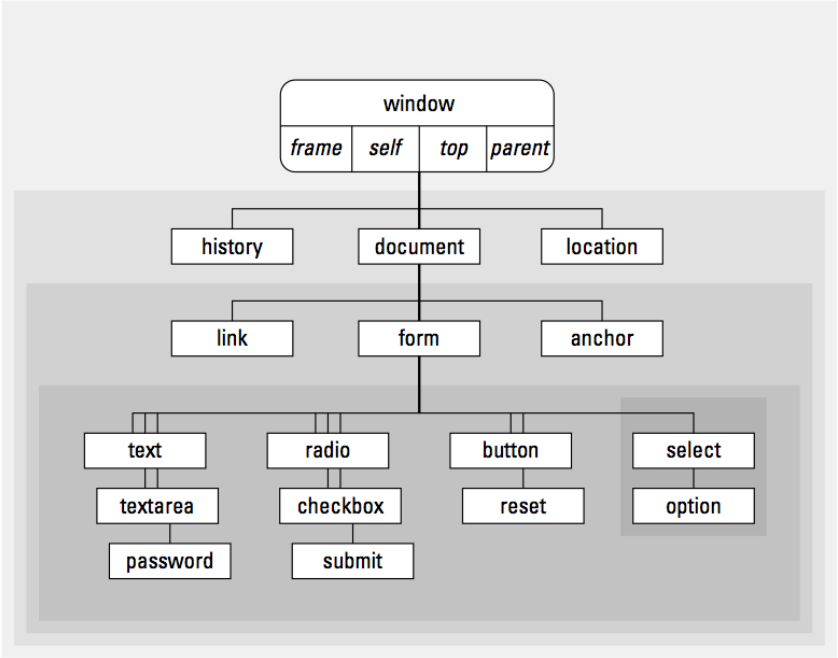

Figure 1: The lowest common–denominator DOM.

conditionally or based on variable arguments. For these reasons, `write` is one of the most often–used methods. The `write` method takes any number of arguments, that can be string *literals* (a text between quotes that will be show literally), or *variables*, that is any variable that evaluates to a string value. You can also use the string *concatenation* operator (`+`) to create one long string from several separate ones, thus after running the script fragment:

```
myNameString = "Barend Kobben";
myStringVar = "My name is " + myNameString;
```

`myStringVar` would have the value `My name is Barend Kobben`.

Task 3 : Write a script that shows the line "Hello, it is now: . . . " , where the . . . part is showing the current date and time. •

To achieve this, you should use:

- the method `write()` of the DOM object `document`;
- the JavaScript built–in method `Date()` that returns the current date and time.

4.2 The `document.getElementById` method

Another much–used DOM method was already mentiond in section 1: The `getElementById` method of the `document` object. This finds an HTML object by its unique `id`, thus creating a programmable object. You can assign an id to virtually all HTML objects, by simply adding an attribute `id="someID"` to it. You have to take care yourself the id is unique!

5 Using functions

Functions are one of the fundamental building blocks in JavaScript. A function is a JavaScript procedure — a set of statements that performs a specific task. A function definition has these basic parts:

- The function keyword.
- A function name.
- A comma-separated list of arguments to the function in parentheses.
- The statements in the function in curly braces {}.

It's important to understand the difference between *defining* and *calling* a function. Defining the function simply names the function and specifies what to do when the function is called. Calling the function actually performs the specified actions with the indicated parameters.

Task 4 : Open a new empty WWW-page. Type the code shown in listing 2 into the HTML-editor window. Save the page and open it in a web browser. •

Listing 2: A simple function

```
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
  function cubed(number) {
    return number * number * number;
  }
</SCRIPT>
</HEAD>
<BODY>
<SCRIPT>
   document.write("The function returns: ", cubed(5) );
</SCRIPT>
<P>All done.</P>
</BODY>
```

This script defines a simple function in the `HEAD` of a document and then calls it in the `BODY` of the document: The function `cubed` takes one `argument`, called `number`. The function consists of one statement: `return number * number * number;`. The `return` statement specifies which value will be returned by the function, so this means the function will return as result the argument cubed. In the `BODY` of the document, the statement `cubed(5)` calls the function with an argument of `5`. The function executes its statements and returns the value `125`.

6 CHALLENGE: Using user inputs

It is of course rather futile to have a function like the one in listing 2, that takes its arguments from a static code in the page. We'd rather have the user input the argument, so that the page can be used as an universal cube calculator (or a square calculator, or any other type of calculator for that matter). We provide you with the following **challenge**:

Task 5 : Change the page you made in Task 4 into a page that lets the user input a number in a Web FORM and returns the number **squared**. •

You have been presented during this exercise with all the elements you need to achieve this:

- listing 2;
- the description of DOM methods `write` and `getElementById`;
- the examples (especially the 3rd one) in the web page you looked at in section 1. You can use the Developer Tools in your browser to study the code;
- the JavaScript reference mentioned on page 1.