**Dynamic Web Pages (JavaScript and Switching Styles)**

Static web pages are pages that remain the same when you view their URL (unless they were edited by the page creator). Everything we created so far via HTML and XHTML were static pages. Dynamic web pages are pages where some or all of the content is dependent on some conditions or user interaction. Google’s search results are dynamic, as an example, since their content depends on the user search query and on the current state of the Google database(s). Dynamic web pages are created using:

* Client-side scripts embedded in an HTML page for processing on the client (your computer)
* Server-side programs that are processed on a server computer
* A mix of client-side scripts and server-side processing

**Server-Side Programming**

Server-side dynamic pages are generated at request-time by a program running on a server. It could be a response to a user inputting data on a form, or it could be a web page generated on the fly from data in a database. Many large web sites or content management systems, for example, are generating HTML pages on the fly. A reporter, for example, files her story into the company’s database, complete with title, tags, and content. That story could be used for the printed version of the paper or be used to generate an HTML response on the fly when a user requests “today’s news stories” via a web URL.

Server-side programs can be written in many languages. Popular are JSP (Java Server Pages), ASP (Active Server Pages), or scripting languages such as PHP or Perl, or C, C++, or pretty much anything else that adheres to a protocol called Common Gateway Interface (CGI) to regulate information flowing between HTML and a program. However, server-side programming requires special access to the web servers and is therefore not suitable for “simple” users.

**Client-Side Programming**

Client-side programs are embedded inside a web pages and execute on the client. They rely on the assumption that the client (any web browser, e.g. Firefox, IE, Safari, Opera) can understand and process the instructions properly. The language most commonly used is JavaScript, which is relatively simple. Examples include drop-down menus on web pages, form data validation (e.g. “is the date entered a valid date”), some simple simulations/games, or other behavior of a web pages based on “mouse events”, i.e. where you click, double-click, or hover.

**Mixed Client/Server Side Programming**

This is most common: part of the (pre-)processing happens on the client to avoid the delay in passing data back and forth to the server, but the server processes more complex requests, usually utilizing other resources such as a database. For example, Google Maps is such a complex, dual purpose “application”. The hot topic these days is combined client/server processing nicknamed AJAX: shorthand for [asynchronous](http://en.wikipedia.org/wiki/Asynchronous_I/O) [JavaScript](http://en.wikipedia.org/wiki/JavaScript) and [XML](http://en.wikipedia.org/wiki/XML) (see <http://en.wikipedia.org/wiki/Ajax_(programming))>. We will get into that later. Since JavaScript is the language to know for dynamic web pages- albeit not required - we’ll start with that:

**Introduction to JavaScript**

From Wikipedia: *“JavaScript is a scripting language used to enable programmatic access to objects within other applications. It is primarily used in the form of client-side JavaScript for the development of dynamic websites. JavaScript is […] characterized as a dynamic,* ***weakly typed****, prototype-based language with first-class functions. JavaScript was influenced by many languages and was designed to look like Java, but to be easier for non-programmers to work with.*

*JavaScript, despite the name, is essentially unrelated to the Java programming language even though the two do have superficial similarities. Both languages use syntaxes influenced by that of C syntax, and JavaScript copies many Java names and naming conventions. The language's name is the result of a co-marketing deal between Netscape and Sun, in exchange for Netscape bundling Sun's Java runtime with their then-dominant browser. […]*

*"JavaScript" is a trademark of Sun Microsystems. It was used under license for technology invented and implemented by Netscape Communications and current entities such as the Mozilla Foundation.”*

Generally you define JavaScript functions in the header part of a document and call them when an event such as a mouse click occurs, but JavaScript programs could also be located in external script files for easy reusability.

**Example:**

Our first JavaScript program just adds some text to a web page

<html>

<head>

<title>Simple Inline JavaScript Example</title>

</head>

<body>

<h1>Spock says</h1>

**<script language="javascript" type="text/javascript">**

**document.writeln("Live long and prosper");**

**</script>**

</body>

</html>

This example uses the “document” object and calls on its “writeln” function to add text to an HTML document (I could have much easier created a static page for this).

**Example:**

A more interesting example is to add the current date to a document.

<html>

<head>

<title>Simple Inline JavaScript Example 2</title>

</head>

<body>

<p>Captain Picard says:

<script language="javascript" type="text/javascript">

var today = new Date();

document.writeln("Today is " + today + ". ");

</script>

Engage!

</p>

</body>

</html>

This is more interesting. When you run this example it will display the current time of your computer, which will be different from this screenshot – the essence of a dynamic page: it can be different every time. **Question**: is the above proper XHTML?

**Data Types and basic Operations**

Note that in the above code we declare a variable named ‘today’ whose type is determined by what I store in it (in this case an object of type Date). That’s a weakly-type language. The type of a variable can even be different depending what you store in it, and the operation between two variables can also have different meaning depending on their current value. Basic operations are of course +, -, \*, / but also ++ (increment by 1) and -- (decrement by 1). The + operator in particular can have multiple meanings:

var x = 3;

var y = ‘Bert’

document.writeln(x + x) // will produce ‘6’

document.writeln(y + y) // will produce “BertBert”

**Events**

JavaScript is event-driven, i.e. it can react to events such as clicks on buttons, checking checkboxes, loading pages, hovering with the mouse, and more. Let’s use some events to introduce the three simple dialog boxes that are available (which incidentally ‘block’ execution until dismissed).

**Example:**

Bringing up blocking dialog boxes with text when a button is clicked:

<html>

<head>

<title>Simple Inline JavaScript Example 3</title>

</head>

<body>

<p>Words of wisdom by:</p>

<form>

<input type="button" value="Kirk" onclick="alert('Go boldly')">

<input type="button" value="Spock" onclick="confirm('Live long')">

<input type="button" value="Picard" onclick="var x=prompt('Engage at Warp ')">

</form>

</body>

</html>

**Functions and basic logic**

JavaScript code can, and usually is, combined into functions to be called at the appropriate time. Functions are typically defined in the header section of a document so that they load first before they are called later in the document. Functions can specifically return a value via a return statement or they can cause something to happen without returning a specific value.

<html>

<head>

<title>Simple Inline JavaScript Example 4</title>

<script language="javascript" type="text/javascript">

**function greetVisitor()**

**{**

**alert("live long and prosper");**

**}**

</script>

</head>

<body>

<p>Click to see the Vulcan greeting:</p>

<form>

**<input type="button" value="Greetings" onclick="greetVisitor()">**

</form>

</body>

</html>

Functions can also have (optional) input arguments and they can return values via the return statement. Functions can even be stored in external files:

<head>

<title>Simple Inline JavaScript Example 3</title>

**<script language="javascript" type="text/javascript" src="myscript.js">**

**</script>**

</head>

Here is a web page that might give you some chills; try to figure out what will happen if you load that page.

<html>

<head>

<title>Simple Inline JavaScript Example 7</title>

<script language="javascript" type="text/javascript">

function typeManuscript()

{

for (var count = 1; count <= 1000; count = count + 1)

{

document.writeln("All work and no play makes Jack a dull boy<br />");

}

}

</script>

</head>

<body>

<h1>Name this Movie</h1>

<script language="javascript" type="text/javascript">

typeManuscript();

</script>

</body>

</html>

**Switching Style Sheets**

We learned how to attach multiple style sheets to a document, which all ‘cascade’ into one style. You can also define alternate style sheets so that the user can switch between them to give a page the look they prefer. You can, for example:

* use a default style sheet for your average user
* provide an alternate style sheet with large print for users with poor eye sight
* provide another style sheet in gray-scale to use for printing
* use yet another style sheet to optimize your content to viewers with small devices (smart phones)

Of course we need (at least) two style sheets. For simplicity, let’s create two very simple style sheets right now (alternatively, copy an existing style sheet and make some modifications):

**File style1.css**

body {

margin: 10px;

width: 100%;

}

h1 {

background-color: blue;

color: yellow;

}

.justified

{

text-align: left;

}

**File style2.css**

body {

margin: 10px auto;

width: 480px;

}

h1 {

background-color: yellow;

color: blue;

}

.justified

{

text-align: right;

}

Now let’s create a simple HTML file with the first style sheet attached:

**File style-test.html**

<html>

<head>

<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

<title>Style Switcher</title>

<link href="style1.css" rel="stylesheet" type="text/css" />

</head>

<body>

<h1>Style Switcher</h1>

<p class="justified">This is some text.</p>

</body>

</html>

As usual, the style determines the look of the header and paragraph, as we learned some time ago. To add the second style sheet as an alternate style, we give our first style sheet a title and then add the alternate sheet as follows (bold lines have changed):

<html>

<head>

<meta content="text/html; charset=utf-8" http-equiv="Content-Type" />

<title>Style Switcher</title>

**<link href="style1.css" rel="stylesheet" type="text/css"**

**title="Default Style" />**

**<link href="style2.css" rel="alternate stylesheet" type="text/css"**

**title="Mobile Style" />**

</head>

<body>

<h1>Style Switcher</h1>

<p class="justified">This is some text.</p>

</body>

</html>

Save the modified HTML document and view your main page in Firefox (not IE, and not in design) – nothing has changed! But now select from the Firefox menu “View | Page Style” and pick you alternate style! Pretty cool (I hope you think) but there are two disadvantages:

* the new style does not ‘stick’ and has to be re-selected every time
* this is *not* supported by Internet Explorer!!

We need to develop a method to switch styles that remedies both problems. That’s where JavaScript comes in! First we need to give the two style tags an ID so that we can reference them in the DOM. Add the following to the style tags above:

**<link href="style1.css" rel="stylesheet" type="text/css"**

**title="Default Style" id="default" />**

**<link href="style2.css" rel="alternate stylesheet" type="text/css"**

**title="Mobile Style" id="mobile" />**

Now we can add the script to get the style elements and swap them. We’ll create a new function for that and add it to the header:

<script type="text/javascript">

function switch\_style()

{

style\_default = document.getElementById("default");

style\_mobile = document.getElementById("mobile");

if (style\_default.disabled)

{

style\_default.disabled = false;

style\_mobile.disabled = true;

}

else

{

style\_default.disabled = true;

style\_mobile.disabled = false;

}

}

</script>

Now all we need is a trigger to call the function. We could use, for example, a button:

<form>

<input type="button" value="Switch style" onclick="switch\_style()" />

</form>

Now you should be able to put everything together for a style-switching page that will work in every modern web browser.

BUT, the style switch does not stick, i.e. does not persist between pages or visits. In other words, if we visit the page again at a later time, the default style will be valid again, and if we switch to another pages, our choice of styles will not be saved. The solution here is cookies.