Draft--Comments Welcome and Needed

Writing JavaScript

Views of a Fellow Student

Andy Dean

www.openepi.com

What is it?

JavaScript is a text-based computer language commonly used within HTML pages, where it must lie between <script> and </script> tags. More than one set of script tags can be on a single page. The usual practice is to place functions that will be called in the <header> portion of the page and code to be run when the page is loaded in the <body> of the page. Despite the name, Javascript is only distantly related to Java, a compiled language that was meant to be the universal web language, but whose universality has been seriously compromised by a battle between Sun Microsystems and Microsoft over ownership.

JavaScript is defined by a set of specifications (ECMAScript) that is partially implemented by major browsers. Although there were substantial differences in its implementation by version 4 of both Netscape (NS) and Internet Explorer (IE) browsers, the differences have been largely removed in versions 7 of NS and IE 6, making JavaScript a reasonably universal language (if your universe is IE, NS, Opera, and Mozilla).

For a definition of JavaScript, see, for example:
http://www.marketingterms.com/dictionary/javascript/
In general, for information on anything covered here, a great deal more is available by searching the web.

Pros

JavaScript is an interpreted language, meaning that changes made in the code are executed the next time the code is run and that (barring tricks) the code is generally available to see with the "View:Source" feature of a browser (although it may be copyrighted). It is easy to copy and modify.

What is called "client-side" JavaScript is embedded in HTML pages for use in browsers on individual computers. Since running it requires only a browser (with JavaScript turned on), JavaScript for the browser is essentially operating system independent. The same JavaScript program (with some testing and tweaking) can be written to run in

popular browsers under Microsoft Windows, in Linux, Unix, and the Macintosh. Although JavaScript can also be written for Internet servers, a server is not required to run "client-side" JavaScript—not even a local webserver such as IIS or Personal Web Server. An HTML page stored on a local computer's hard disk can be run locally in a browser, executing the JavaScript that may be contained in the page.

Cons

JavaScript (and Java) are bound hand and foot by security provisions that prevent access to the operating system outside the browser. There is no straightforward and universal way in HTML/JavaScript to do a directory of the client computer, or write a file to the hard disk. The Internet is designed so that storage of data occurs on servers, which would be fine if everyone had constant Internet access, but is a definite drag when doing a survey on a laptop in a village without Internet (or electricity, perhaps). OpenEpi programs (www.openepi.com) are able to use some tricks to write files to Microsoft Windows (Windows) systems, which include the majority of public health computers.

As an interpreted language, JavaScript should be somewhat slower than compiled languages. This will only be apparent in computer-intensive mathematical routines, and the fast processors and abundant Random Access Memory (RAM) in modern computers tend to remove this problem from practical discussion.

Learning all aspects of JavaScript programming is not a trivial task, but learning to write mathematical or statistical programs requires a much more limited investment, particularly when modifying programs written by others. The curly brackets and unique syntax of "if" and "for" statements and some of the operators (+=) turn out after a few days to be assets for efficient programming.

Choosing Tools

Books and Reference Cards

I've tried a number of books, but the ones I use are:

Flanagan, David. JavaScript, The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol, CA, USA. 2002.

Wilton, Paul, Williams, Stephen, and Li, Sing. Practical JavaScript for the Usable Web. Glasshaus Ltd, Birmingham, UK, 2002.

Musciano, Chuck and Kennedy, Bill. HTML, The Definitive Guide. O'Reilly & Associates, Inc., Sebastopol, CA, USA. 1998.

A handy series of 6 reference cards for HTML and JavaScript is available from:

javascript.visibone.com

I keep them in my laptop case.

Google

I could not have used JavaScript without www.google.com. Every time I have a question, I type something like:

Javascript popup window tutorial

into Google, and often find a clearly written tutorial on exactly what I need.

Dreamweaver

You can write HTML and Javascript in Notepad or Wordpad and run it in a browser. Once you have multiple files to manage, it is helpful to have a programming environment that displays several files, highlights JavaScript syntax, constructs HTML pages, and runs IE and NS on demand for testing. Microsoft FrontPage is a good HTML editor, but it does not cater to JavaScript. Dreamweaver MX is the program I use. It works well for making HTML pages and writing JavaScript files. There are other HTML and Javascript editors, both free and commercial, available via web search.

Writing

Sources of code

The Internet is filled with useful JavaScript code, much of which is available for use and modification in your application. Searching in Google will usually turn up what you want, including the fine collection of hundreds of javascript statistical calculators assembled by John C. Pezzullo (choose "Statistical Portal" in www.openepi.com, or search Google for JavaScript statistics). Searching for
Javascript matrix math
For example, turns up more than 10,000 references in Google. Luckily, not all are relevant, but there are some real pearls among the flotsam and jetsam.

Licenses

JavaScript code may be copyrighted and commercial (for sale, usually with distribution and modification rights), copyrighted with rights reserved, not copyrighted, or copyrighted with an open source license such as GNU, BSD, or MIT. Open source programming and licensing is a relatively new phenomenon best explained at:

www.opensource.org/licenses/index.php

Many of the open source licenses retain ownership of the original code but permit others to modify and distribute the code as long as a copy of the license is included and credit is given. They usually include a disclaimer to ward off lawsuits if the code injures a favorite pet or launches Saturn 5 prematurely.

Giving credit

In the huge, fluctuating mass that comprises the Web, it is probably best to develop habits that satisfy both legal and scientific conventions for use of intellectual property of others. In other words, check the license and copyright, if any, and, where doubt exists, contact the author for permission (sometimes refused). As in scientific research papers, it is courteous and ethical to cite sources of code being used, even though modified.

Style

There are many styles of programming, even in JavaScript, and many articles on how to document and comment code. Obviously statistical code is more useful if others can tell what is going on through meaningful variable names and embedded comments. In the 1960's, it was important to make code run faster by using single-letter variable names, but this is rarely helpful now except in an inner loop of a highly iterative routine (and even there, subject to confirmation by experiment). There are many styles for variable names, including:

MHChiSquare

mhchisquare

mhChiSquare

mh_chi_square

JavaScript variable names cannot contain spaces and they are absolutely case-sensitive, so these are all different variables.

It is important to adopt a convention so that when you remember the name of a variable next week or 30 lines down, you automatically write it with the correct case and word-separation convention.

Curly bracket hygiene is another place for individual styles. Those who are used to begin and end statements in other languages will need a few days to get used to enclosing the bodies of functions, the consequences of if statements, and the contents of for loops in curly brackets. They are so compact that it is tempting to write:

if (a>0){v=a}else{a=0}

When I first saw something like:

```
if (a>0)
  {
   v=a
  }
else
  {
   a=0
  }
```

I thought to myself, "How silly and pedantic!" Since than, I have come to like the second style. First, I realized that JavaScript ignores white space; there is no (or minimal) extra charge for extending down the page. Then I found that, when an if statement is inside a couple of nested loops inside a function and something is wrong, you can line up all the brackets up so that the pairs are in the same column; you don't have to count and recount left brackets and right brackets to find the problem. So I am a fan of indenting and having brackets on lines by themselves. The latter also prevents commenting out a bracket by mistake when you are removing an alert box or making a comment.

Pascal programmers spend their first year learning to end each line with a semicolon. JavaScript accepts, but does not require, a semicolon at the end of a line, leading to the conclusion by one web wag that "Inserting semicolons does nothing, but it builds character." The only time they are needed is when putting more than one statement on a line—not a great idea in any case.

Debugging

Other languages, such as Visual Basic, Pascal, and C++, have useful debuggers, programs that display the code and allow you to run it one line at a time, while detecting errors and seeing what is happening. Unfortunately, I have not found such a program for JavaScript, although I have tried to use the debugger that comes with Dreamweaver and the massive debuggers that are supplied with Microsoft Visual Studio .NET. I have given up debating whether this is my fault or the vendors' and have settled on the following tools:

1) Microsoft Internet Explorer (IE 6.xx) with debugging messages turned on:

Go to **Tools|Internet Options|Advanced|** and place check marks in both of the following options:

[ ] Disable script debugging
[ ] Display a notification about every script error

IE seems to detect errors quite well, but may have no clue about the line where the error occurred or even what file it occurred in. It pretends to provide both, but they are often in some unknown ballpark when the error is finally found. Hence, when an error occurs,

particularly a syntax error, I run the HTM file in Netscape (by pressing ctrl-F12 rather than F12 in Dreamweaver).

2. Netscape (NS 7.xx) is good at detecting bugs and their location and gives descriptive error messages, but it runs merrily on, ignoring the problem, unless you know how to find the debugger. When I get an error message in IE, I get none in Netscape, although the application may not do what it is supposed to. When this happens, I choose Tools|Web Development|JavaScript Console from the Netscape 7 menu, and an error window pops up displaying one or more errors with the line number, the name of the file with the error, and a descriptive message. There is probably a more efficient way to get to this facility, but I have not had time to discover it! It is particularly good for syntax errors.

3. Visual inspection. Apart from errors in logic, there are several common causes of bugs that pop up again and again. Some of these are listed below.

The Usual Causes of Bugs

1) Unmatched curly brackets. Too few and the entire program is disabled by the sound of one bracket clapping (or failing to clap). Too many and a function ends prematurely with code hanging out in the breeze after the extra bracket. Lining up the brackets as described above allows a quick visual check to see if they line up in matched pairs.

2) Upper/Lower case problems. JavaScript, unlike Visual Basic, is absolutely fussy about case. Capitalize the "if" or "for" command, and you get an "object expected" error because If and For are not JavaScript commands (and the browser's diagnosis is limited to, "huh?"). Create a function called mhChiSquare and call it as mhchiSquare() and it is never called because of the lower case "c". Leave off the parentheses in the call, and JavaScript thinks you are referring to a variable and not a function.

3) The ^&*^%$# "==" operator. Once in a while, you do an if statement with a test to see if two things are equal. JavaScript takes "=" as an assignment rather than a test. In other words, "t=1" sets t to 1 as in other languages. But "**if (t==1)**" is required to test whether t is indeed equal (or should I say, "equal equal") to 1. About once a week, I find a bug caused by writing "if(t=1)" rather than "if(t==1)", and the next time I use such an expression is just far enough away so that I forget and make the mistake again. The problem is made worse by the fact that "<" and "<=" both work as expected and are more frequently used than the test for equality. Maybe some genius will write a browser that knows you can't use a single "=" inside an **if** condition and shares this information with the programmer at the right time.

Alert Boxes

When I need to trace what is happening line by line or can't figure out whether a particular function is being called, I use the barehands technique of inserting alert boxes in the code. The command, alert(212), inserted on line 212, for example will pop up the message "212" and wait for you to press the OK button. If you run the program and the message box comes up, you know your program executed line 212. By inserting several

such lines in the program, you can localize where an error is occurring because the alert box stops everything until you press OK.

Alert boxes can also show you the value of any variable or expression. For example,

alert("chisq="+chisq)

displays the value of chisq at the point in the program where the alert is placed. Note the initial quoted string, "chisq=", which is simply a message to indicate what is being shown. It is joined to the value of chisq by the + sign, which (unlike the = sign) is versatile enough to do concatenation of strings but switches to addition when dealing with numbers.

When writing statistical code, it may be helpful to confirm values after every calculation by including an alert box in the code. When the program functions correctly, the alerts can be commented out by inserting // in front of the line. With this technique, the alerts can be reactivated merely by removing the //, in case there is a need for more debugging.

Handling Errors

Primitive error handling can be done with if statements, such as:

```
if (b==0 || c==0)
  {
   alert("Neither a nor b can be zero; please try again")
   return
  }
```

This will stop the program with the "modal" alert box. After the user clicks OK, the "return" exits from the current function. You can also use "return false" so that the function returns a value of "false" when the calculation is not performed.

More capable error handling makes use of the **try…catch….finally** statement in JavaScript, for which there are many tutorials on the Internet.

Testing

Testing can take many forms, but should at least include the major browsers and datasets with extreme values, negative values, zeroes, and missings, as well as numbers for which results are available from other sources. The results should be recorded so that others can know what has been tested (and what has not.)

Writing Programs Using the Open Epi Toolkit

Open Epi (www.openepi.com) is an open source website promoting the writing and distribution of epidemiologic software in JavaScript. It contains a toolkit that can be downloaded and adapted to create new programs. The most difficult parts of the programming are already done—providing the input and output tables and allowing the user to enter data without having to take special precautions with the Enter key. Open Epi authors provide instructions to format the input table, write a statistical JavaScript (.js) file, and send the output to an HTML table using the commands provided in the OpenEpi Toolkit. A menu is provided for the application that allows uniform access to documentation, pictures of the input and output screens, and, if desired, demonstration data to give an example of how the application is used. Instructions for use of the Toolkit are provided as a separate document available from the Open Epi website.

Please send comments and suggestions to:

agdean9 at Hotmail.com

(Replace " at " with @. Don't put email addresses on a website with the @ intact, as they will be grabbed by "spiders" for advertising mailing lists. Use a JavaScript program to dynamically construct the address from unrecognizable parts, or otherwise obscure it as above. Thanks to Jens Lauritsen for this tip.)