cis3.5 fall2009 advanced material: javascript

topics:

- whirlwind tour of javascript

resources:

- tutorial:
  `http://www.w3schools.com/JS/default.asp`
- reference:
  `http://www.w3schools.com/jsref/default.asp`

introduction to javascript

- javascript is a type of *client-side* programming
  - *client-side* vs *server-side* web programming:
    - client-side runs on the client: web pages are stored on the server, get downloaded to the client and run on the client
    - server-side runs on the server: code is stored on the server and run there
- javascript code is listed in the **head** portion of your HTML file:
  `<head>...</head>`
  after the `<title>...</title>`
- javascript code goes between **script** tags:
  `<script>...</script>`
- javascript code can also go in between the `<body>...</body>` tags on a web page
- javascript can also be invoked from inside some HTML form options, like *onclick*

first example

- The body of an HTML page is referred to as the *document object* in Javascript. To write something on the page, you invoke the *write* function of the document object:
  `document.write()`
- I put `()` at the end of a function name to indicate that it is a function (instead of some other type of property, like a variable). When you invoke the function (i.e., make it run), sometimes you put things inside the `()`'s, as in the example below.

```
<html>
<head>
 <title>first javascript program</title>
 <script language="javascript">
  document.write( "<h2>hello from javascript!</h2>" )
 </script>
</head>
<body>
hello from the body of your web page!
</body>
</html>
```


variables

- *Variables* are like temporary places to store data. Variables have a *name* and a *type* (which can be a number or a string, i.e., alphanumberic). Variable names have to follow rules: letters and numbers and underscore (_) are allowed, but name cannot start with a number. Variables have to be "declared" before they can be used.
- The browser window is referred to as the *window object* in javascript. To pop up a new window with a "prompt" inside it (something that asks for user input), use the `window.prompt()` function, as shown below.

```
<html>
<head>
<script language="javascript">
 var num;
 num = window.prompt( "enter a number", "0" )
 document.write( "<h2>hello from javascript!</h2>" )
 document.write( "num=" + num )
</script>
</head>
<body>
hello from the body of your web page!
</body>
</html>
```

operators

- *Operators* are like mathematical operators and provide ways of combining data that is stored in variables or *constants*.
- A "constant" is a way of specifying data without storing it in a variable.
  A *numeric* constant is like: 2 or `34.789`
  A *string* constant is written in double quotes, like: `"hello"`
- The operators that manipulate numbers are: $+$ for addition, $-$ for subtraction, $*$ for multiplication, $/$ for division, and $\%$ for modulo (integer division).
- Note that the $+$ operator is also can also be used with strings, to concatenate two strings together.
- The operators that compare two numbers are: $==$ for testing equality, $!=$ for testing inequality, $>$ for testing greater than, $>=$ for testing greater than or equal to, $<$ for testing less than, and $<=$ for testing less than or equal to.
- `parseInt()` is a built-in function (i.e., it is part of javascript) that converts a string value to a numeric value. This is handy in case you want to do math on something the user enters. Typically, the values entered by users are stored in string variables and have to be converted to numeric variables before you can perform mathematical operations on them.

- The example below shows how to prompt the user for input, convert the user's input from string format to numeric format and then perform math on that number (multiply it by 2) and report the result.

```
<html>
<head>
<script language="javascript">
 var snum;
 var num;
 var num2
 snum = window.prompt( "enter a number", "0" )
 num = parseInt( snum )
 num2 = num * 2
 document.write( "<h2>" )
 document.write( "hello from javascript!" )
 document.write( "<p>" )
 document.write( "your number =" + num )
 document.write( "<p>" )
 document.write( "2 times your number =" + num2 )
</script>
</head>
<body>
hello from the body of your web page!
</body>
</html>
```


HTML forms

- *Forms* provide a means in HTML for users to enter input in different ways using elements right inside the browser window (instead of having to pop up a javascript prompt window, as we did above).
- Forms are included in the body of an HTML file and are enclosed in the following tags: `<form> ... </form>`
- With forms, the user enters data and then presses *enter* or clicks a *submit* button that sends the form data to be processed (depending on how the form is designed).
- This form data might be processed on the server, where the HTML file is stored, using a language like PHP.
- The form data could also be processed on the client, using javascript—which is what we'll do here.

- The example below shows how the user can enter information in a "text field" on a form and, after pressing the `enter` key, the data is processed by javascript and the user's input is displayed inside the form.

```
<html>
<body>
hello from the body of your web page!
<form name="form1">
<input type="text" name="formname">
</form>
<script language="javascript">
 var myname;
 myname = prompt( "enter your name:", "" )
 document.form1.formname.value = "hi, " + myname
</script>
</body>
</html>
```

comments

- *Comments* are text that is ignored by browser or javascript interpreter.
- in HTML, comments are enlosed in tags like this:
  `<!-- put comment here -->`
- In javascript, comments begin with two slashes and contine to the end of a line, like this:
  `// put comment here`
- It is good practice to put javascript code inside HTML comments, just in case a browser doesn't support javascript (in which case, the browser wouldn't know what to do with the javascript code and would just display the javascript code like plain text).

```
<html>
<head>
<script language="javascript">
 <!--
 document.write( "<h2>hello from javascript!</h2>" )
 -->
</script>
</head>
<body>
hello from the body of your web page!
</body>
</html>
```


dialog boxes

- A *dialog box* is a window that pops up and asks the user for input. We have already used the `prompt()` dialog box. The dialog boxes are listed below:
  - The `alert()` box displays a message for the user, who only has to click on "ok" to make the box go away:
    `alert( "hello!" )`
  - The `confirm()` box displays a message to which the user can respond with either "ok" or "cancel":
    `okay = confirm( "are you sure?" );`
  - The `prompt()` box asks the user for text input, as you have seen.

branching statements

- The main type of branching statement in javascript is `if ... else`.

```
<html>
<body>
<script language="javascript">
<!--
alert( "hello friend!" )
myname = prompt( "what is your name?", "" )
okay = confirm( "so, your name is " + myname + "?" )
if ( okay ) {
  document.write( "hello " + myname + ", and welcome to my chocolate factory!" )
}
else {
  document.write( "sorry, wrong number!" )
}
//-->
</script>
</body>
</html>
```


events

- An *event* is something the user does, like click on a button, move the mouse, or enter text.
- Javascript can be used in conjunction with HTML to "catch" (recognize) user events and respond to them. One commonly caught event is called `onclick`, which refers to situations where the user clicks on a button in an HTML form.
- You can embed javascript within the HTML form to make the button respond in a particular way, such as displaying text in the HTML window or popping up a dialog box.

```
<html>
<body>
<form name="form1">
please enter your name:<br>
<input type="text" name="yourname"><br>
click this button
<input type="button" value="greet" onclick="form1.txtgreet.value='hello, ' + form1.yourname.value">
<br>
<input type="text" name="txtgreet">
</form>
</body>
</html>
```

functions

- In addition to using javascript's *built-in* functions (i.e., those that are part of the language), you can also write your own functions.
- An example is shown below. Things to note:
  - the keyword `function`
  - the function name (the word "greet" after the `function` keyword, in the example) is something that you define yourself; the naming rules are the same as those described above for variables
  - the parentheses ( ) are placed after the function name
  - the *body* (i.e., content) of the function is enclosed in curly brackets { }

- In this example, the response is the same as the previous example without a function. Using the function makes the code neater. It also provides a bit of "reusable" code, since you can call the same function from other `onclick` elements.

```
<html>
<head>
<script language="javascript">
<!--
function greet() {
  document.form1.txtgreet.value = "Hello, " +  document.form1.yourname.value
}
//-->
</script>
</head>
<body>
<form name="form1">
please enter your name:<br>
<input type="text" name="yourname"><br>
click this button
<input type="button" value="greet" onclick="greet()">
<br>
<input type="text" name="txtgreet">
</form>
</body>
</html>
```


- The next example shows how you can read information from a form, process it and display output all in the same form. The function makes that easier.

```
<html>
<head>
<script language="javascript">
<!--
function calcIt() {
  var num1, num2
  num1 = Number( document.form1.mynum1.value )
  num2 = Number( document.form1.mynum2.value )
  document.form1.txtsum.value = num1 + num2
}
//-->
</script>
</head>
<body>
<form name="form1">
enter first number:<br>
<input type="text" name="mynum1"><br>
enter second number:<br>
<input type="text" name="mynum2"><br>
click this button to add them up!
<input type="button" value="add'em" onclick="calcIt()">
<br>
<input type="text" name="txtsum">
</form>
</body>
</html>
```


function arguments

- Sometimes you want a function to do something with data where you don't specify inside the function which part of the form that the data comes from. This can make a function more flexible, so that it could be used to manipulate or respond to data elements from different parts of forms. This flexibility is provided by the use of *function arguments*.
- Function arguments are listed in between the parentheses ( ) just after the function name. Function arguments are variables, and they are used just like variables inside the body of the function.
- In the example below, the name of the function is `calcIt`. It has two arguments, which are named `num1` and `num2`. The function adds its two arguments together and displays the answer in a text field in the form.

```
<html>
<head>
<script language="javascript">
<!--
function calcIt( num1, num2 ) {
  var sum
  sum = num1 + num2
  document.form1.txtsum.value = sum
}
//-->
</script>
</head>
<body>
<form name="form1">
enter first number:<br>
<input type="text" name="mynum1"><br>
enter second number:<br>
<input type="text" name="mynum2"><br>
click this button to add them up!
<input type="button" value="add'em"
onclick="calcIt( Number(form1.mynum1.value), Number(form1.mynum2.value) )">
<br>
<input type="text" name="txtsum">
</form>
</body>
</html>
```


iteration

- Sometimes you want to do things multiple times. Iteration, or *looping* statements, allow you to do this.
- A `for` loop will do something for a specified number of times.
- The example below draws a table, based on the number of rows specified by the user.


```
<html>
<head>
<script language="javascript">
<!--
function draw_table( numrows ) {
  var i;
  document.write( "<table>" );
  for ( i=0; i<numrows; i++ ) {
    document.write( "<tr>" );
    document.write( "<td bgcolor=#999999>row " + i + "</td>" );
    document.write( "<td bgcolor=#ff0000>hello</td>" );
    document.write( "</tr>" );
  }
  document.write( "</table>" );
}
//-->
</script>
</head>
<body>
<form name="form1">
enter number of rows for your table:<br>
<input type="text" name="mynum"><br>
click this button to draw the table!
<input type="button" value="make table"
onclick="draw_table( Number(form1.mynum.value) )">
</form>
</body>
</html>
```