COMP519
Web Programming

Autumn 2015

JavaScript

Client-Side Programming

HTML is good for developing static pages.

- Can specify text/image layout, presentation, links, . . .
- Webpage looks the same each time it is accessed.

In order to develop interactive/reactive pages, we must integrate programming in some form or another.

Client-Side Programming

HTML is good for developing static pages.

- Can specify text/image layout, presentation, links, . . .
- Webpage looks the same each time it is accessed.

In order to develop interactive/reactive pages, we must integrate programming in some form or another.

Client-side programming

- Programs are written in a separate programming (or scripting) language, e.g. JavaScript, JScript, VBScript.
- Programs are embedded in the HTML of a Web page, with (HTML) tags to identify that it is a program component.

Client-Side Programming

HTML is good for developing static pages.

- Can specify text/image layout, presentation, links, . . .
- Webpage looks the same each time it is accessed.

In order to develop interactive/reactive pages, we must integrate programming in some form or another.

Client-side programming

- Programs are written in a separate programming (or scripting) language, e.g. JavaScript, JScript, VBScript.
- Programs are embedded in the HTML of a Web page, with (HTML) tags to identify that it is a program component.

Use a <script> . . . </script> pair to identify a block of JavaScript code.

In old style scripts you might see <script type="text/javascript"> . . . <script>, but the default script type is now JavaScript.

Client-Side Programming (cont.)

The browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML.

It is also possible to also allow the user (client) to input information and process it, might be used to validate input before it's submitted to a remote server.

Scripting Languages

A scripting language is a simple, interpreted programming language. Scripts are embedded as plain text and interpreted by application (in our case, a web browser).

Scripting Languages

A scripting language is a simple, interpreted programming language. Scripts are embedded as plain text and interpreted by application (in our case, a web browser).

Simpler execution model: Don’t need a compiler or development environment.

Scripting Languages

A scripting language is a simple, interpreted programming language. Scripts are embedded as plain text and interpreted by application (in our case, a web browser).

Simpler execution model: Don't need a compiler or development environment.

Saves bandwidth: Source code is downloaded, not compiled executable.

Scripting Languages

A scripting language is a simple, interpreted programming language. Scripts are embedded as plain text and interpreted by application (in our case, a web browser).

Simpler execution model: Don't need a compiler or development environment.

Saves bandwidth: Source code is downloaded, not compiled executable.

Platform-independence: Code interpreted by any script-enabled browser, slower than compiled code, and not as powerful/full-featured.

Scripting Languages (cont.)

JavaScript: the first Web scripting language, developed by Netscape in 1995.

- Syntactic similarities to Java/C++, but simpler, more flexible in some respects, but limited in others (e.g. loose typing, dynamic variables, simple objects).

Scripting Languages (cont.)

JavaScript: the first Web scripting language, developed by Netscape in 1995.

- Syntactic similarities to Java/C++, but simpler, more flexible in some respects, but limited in others (e.g. loose typing, dynamic variables, simple objects).

JScript: Microsoft version of JavaScript, introduced in 1996, same core language, but some browser-specific differences.

Fortunately, IE, Netscape, Firefox, etc. can (mostly) handle both JavaScript & JScript.

JavaScript 1.5 & JScript 5.0 cores both conform to ECMAScript standard.

VBScript: Client-side scripting version of Microsoft Visual Basic, which could also be used for web programming.

Common Scripting Tasks

- Adding dynamic features to Web pages.
- Validation of form data (probably still one of the most commonly used application).
- Image rollovers.
- Time-sensitive or random page elements.
- Handling cookies.
- Ajax adds ability to update page elements without (re)loading a webpage.
- Defining programs with Web interfaces.
- Utilize buttons, text boxes, clickable images, prompts, etc.

Script Limitations

Limitations of client-side scripting:

- Since script code is embedded in the page, it is viewable to the world.
- For security reasons, scripts are limited in what they can do e.g. can't access the client's hard drive.
- Since they are designed to run on any machine platform, scripts can not contain platform specific commands.
- Script languages are generally not as full-featured as other programming languages, e.g. JavaScript objects are generally very crude, not ideal for large project development.

JavaScript

```
<html>
<!-- COMP519  js01.html  2015.10.03 -->

<head>
  <title>JavaScript Page</title>
</head>

<body>
  <script>
    // silly code to demonstrate output

    document.write("<p>Hello world!</p>");
    document.write(" <p>How are <br/> " +
                   " <i>you</i>?</p> ");
  </script>

  <p>Here is some static text as well.</p>

</body>
</html>
```

view page

JavaScript code can be embedded in a Web page using <script> tags.

The output of JavaScript code is displayed as if it was directly entered in HTML.

Beginning JavaScript

The document.write command is used to display text in the page. This text can include HTML tags, and the tags are interpreted by the browser as any other regular HTML tags.

As in C/C++/Java, statements end with a semi-colon.

JavaScript comments are also similar to C/C++/Java.

```
// This is a single-line comment.

/* A multiple-line comment has similar syntax
to other programming languages. */
```

Data Types and Variables

JavaScript has only three primitive data types.

String: "foo" 'how do you do?' "I said 'hi'." ""

Number: 12 3.14159 1.53e6

Boolean: true false

Null and Undefined have special meanings in JavaScipt.

```
<html>
<!-- COMP519  js02.html  2015.10.03 -->
<head>
  <title>Data Types and Variables</title>
</head>
<body>
  <script>
    var x, y;
    x= 1024;
    y=x;  x = "foobar";
    document.write("<p>y = " + y + "</p>");
    document.write("<p>x = " + x + "</p>");
  </script>
</body>
</html>
```

view page

Data Types and Variables (cont.)

Variable names are sequences of letters, digits, and underscores that start with a letter or underscore.

Variables names are case-sensitive.

You don't have to declare variables, they will be created the first time used, but it's better if you use var declarations (because of scoping issues).

```
var message, pi=3.14159;
```

Variables are loosely typed, can be assigned different types of values (Danger!).

JavaScript Operators and Control Statements

All of the usual C++/Java type of operators and control statements are provided in JavaScript.

- $+$, $-$, $*$, $/$, %, $++$, $--$, ... (mathematical operators)
- $==$, $!=$, $<$, $>$, $<=$, $>=$ (comparison operators)
- &&, $||$, !, $===$, $!==$ (logical operators)
- if , if ... else, switch (branching statements)
- while, for, do {...} while, ... (looping constructs)

A JavaScript example

```
<html>
<!-- COMP519 js03.html  2015.10.03 -->

<head>
  <title>Folding Puzzle</title>
</head>

<body>
 <script>
    //  distances expressed in inches
    var distanceToSun = 93.3e6*5280*12;
    var thickness = .002;

    var foldCount = 0;
    while (thickness < distanceToSun) {
        thickness *= 2;
        foldCount++;
    }
    document.write("<p>Number of folds = " +
                   foldCount+"</p>");
  </script>
</body>
</html>
```

PUZZLE: Suppose you took a piece of paper and folded it in half, then in half again, and so on.

How many folds before the thickness of the paper reaches from the earth to the sun?

view page

JavaScript Math Routines

The built-in Math object in JavaScript contains many functions and constants.

Math.sqrt()
Math.pow()
Math.abs()
Math.max()
Math.min()
Math.floor()
Math.ceil()
Math.round()
Math.log()
Math.log10()

Math.PI
Math.E

Math.random function returns a real number in [0..1)

Generating random numbers (in a slightly fancy way)

```
<html>
<!-- COMP519  js04.html  2015.10.03 -->
<head>
  <title>Random Dice Rolls</title>
</head>

<body>
  <div style="text-align:center">
    <script>
      var roll1 = Math.floor(Math.random()*6) + 1;
      var roll2 = Math.floor(Math.random()*6) + 1;

      document.write("<img src='http://www.csc.liv.ac.uk/"+
       "~martin/teaching/comp519/Images/die" +
       roll1 + ".gif' alt='die showing " + roll1 + "'/>");
      document.write("  ");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
       "~martin/teaching/comp519/Images/die" +
       roll2 + ".gif' alt='die showing " + roll2 + "'/>");
    </script>
  </div>
</body>
</html>
```

view page

Interactive Pages Using prompt

Somewhat crude user interaction can take place using the JavaScript prompt function.

1st argument: the prompt message that appears in the dialog box.

2nd argument: a default value that will appear in the box (in case the user enters nothing).

Interactive Pages Using prompt

Somewhat crude user interaction can take place using the JavaScript prompt function.

1st argument: the prompt message that appears in the dialog box.

2nd argument: a default value that will appear in the box (in case the user enters nothing).

The prompt function returns the value (as a string) that the user entered in the dialog box.

Interactive Pages Using prompt

Somewhat crude user interaction can take place using the JavaScript prompt function.

1st argument: the prompt message that appears in the dialog box.

2nd argument: a default value that will appear in the box (in case the user enters nothing).

The prompt function returns the value (as a string) that the user entered in the dialog box.

If the value is a number, you (as the web programmer) must use parseFloat or parseInt to convert it to a number.

The example on the next page illustrates the use of the prompt function.

We will see other ways to interact with a user later.

A prompt Example

```
<html>
<!-- COMP519  js05.html  2015.10.03  -->
<head>
  <title>Interactive page</title>
</head>
<body>  <p>
<script>
 var userName = prompt("What is your name?", "");
 var userAge = prompt("Your age?", "");
 var userAge = parseFloat(userAge);

    document.write("Hello " + userName + ".")
    if (userAge < 18) {
      document.write("  Do your parents know " +
                     "you are online?");
    }
    else {
      document.write("  Welcome friend!");
    }
</script>  </p>
  <p>Here is the rest of the webpage...</p>
</body>
</html>
```

view page

Function Definitions

Functions can defined and the syntax is similar to C++/Java, except:

- No return type is specified for the function (since variables are loosely typed).
- No variable typing for parameters (since variables are loosely typed).
- By-value parameter passing only (parameter gets copy of argument).

You can limit variable scope to the function. If the first use of a variable is preceded with var, then that variable is local to the function (without the var keyword, the function will treat the variable as a global variable (Danger!!)).

For proper modularity, you should make all variables in a function local to that function.

Function Definitions (cont.)

Function definitions (usually) go in the <head> section.

The <head> section is loaded first, so then the function is defined before code in the <body> is executed (and, therefore, the function can easily be used later in the body of the HTML document).

```
function isPrime(n)
// Assumes: n > 0  and is an integer
// Returns: true if n is prime, else false
{
  if (n < 2) {   return false;    }
  else if (n == 2) {
    return true;
  }
  else {
      for (var i = 2; i <= Math.sqrt(n); i++) {
        if (n % i == 0) {
          return false;
        }
      }
      return true;
  }
}
```

An Example Using a Function

```
<html>
<!-- COMP519  js06.html  2015.10.03 -->
<head>
  <title>Prime Tester</title>
  <script>
    function isPrime(n)
    // Assumes: n > 0
    // Returns: true if n is prime
    {      // CODE AS SHOWN ON PREVIOUS SLIDE    }
  </script>
</head>
<body>  <p>
 <script type="text/javascript">
    var testNum = parseInt(prompt("Enter a positive integer", "7"));
    if (isPrime(testNum)) {
      document.write(testNum + " <b>is</b> a prime number.");
    }
    else {
      document.write(testNum + " <b>is not</b> a prime number.");
    }
  </script>  </p>
</body>
</html>
```

view page

Random number generation

Recall the “dynamic dice” example from before.

We can define a function for generating an integer in a range, and then use it whenever we need it later.

This promotes easy re-use of functions.

```
<script>
  function randomInt(low, high)
  // Assumes: low <= high
  // Returns: random integer in range [low..high]
  {
    return Math.floor(Math.random()*(high-low+1)) + low;
  }
</script>
```

Typically, we would insert this JavaScript code into the <head> section of a webpage. Alternatively, we can create an external file that contains the code (without the <script> tags), and refer to it in several webpages.

Random dice revisited

```
<html>
<!-- COMP519  js07.html  2015.10.04 -->
<head>
  <title> Random Dice Rolls Revisited</title>
  <script>
    function randomInt(low, high)
    // Assumes: low <= high
    // Returns: random integer in range [low..high]
    {    return Math.floor(Math.random()*(high-low+1)) + low;    }
  </script>
</head>
<body>    <div style="text-align: center">
    <script>
        var j = randomInt(1,6);
        for (var i = 1; i <= j; i++)
          {   roll = randomInt(1, 6);
              document.write("<img src='http://www.csc.liv.ac.uk/"+
               "~martin/teaching/comp519/Images/die" +
               roll + ".gif' alt='die showing " + roll + "'/>");
               document.write("  ");
          }
   </script>
  </div>   </body>   </html>
```

view page

JavaScript Libraries

If you have functions that you use often, you can store them in an external file, and then load the library when you need to do so by referring to the external file.

The file
http://cgi.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js
contains definitions of the following functions:

`randomNum(low, high)`	Random real number in range [low..high).
`randomInt(low, high)`	Random integer in range [low..high).
`randomChar(string)`	Returns random character from the string.
`randomOneOf([item1,...,itemN])`	Returns random item from list/array.

Note: as with external style sheets, do not put <script> tags in the external JavaScript library file.

JavaScript Library Example

```
<html>
<!-- COMP519  js08.html  2015.10.04 -->
<head>
  <title> Random Dice Rolls Revisited</title>
  <script
src="http://cgi.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.html
  </script>
</head>
<body>    <div style="text-align: center">
   <script>
       var j = randomInt(1,8);
       for (var i = 1; i <= j; i++)
         {   roll = randomInt(1, 6);
             document.write("<img src='http://www.csc.liv.ac.uk/"+
              "~martin/teaching/comp519/Images/die" +
              roll + ".gif' alt='die showing " + roll + "'/>");
              document.write("  ");
         }
   </script>
  </div>   </body>   </html>
```

view page

JavaScript Objects

JavaScript has several built-in objects. The Math "library" is actually an object with many functions defined as object methods.

Another built-in object in JavaScript is the String object.

Properties of a String include:
`length` : the number of characters in the string.

Methods include:
`charAt(index)` : returns the character stored at the given index (as in C++/Java, indices start at 0).

`substring(start, end)` : returns the part of the string between the start (inclusive) and end (exclusive) indices.

`toUpperCase()` : returns copy of string with letters uppercase.

`toLowerCase()` : returns copy of string with letters lowercase.

JavaScript Objects (cont.)

To create a String in JavaScript, you can assign a variable using the new keyword, or (in the case of a String) just make a direct assignment (new is implicit for a String assignment).

word = new String("foo"); word = "foo";

For strings, it is better to *not* use the new keyword (for memory and speed issues).

Properties/methods are called exactly as in C++/Java:

word.length word.charAt(0)

JavaScript also has the capability of defining your own objects. These objects are not as rich as objects in other programming languages, e.g. there is limited capability for creating “private” data or functions.

String Example: Palindromes

Suppose we want to test whether a word or phrase is a palindrome (i.e. a phrase that reads the same backwards or forwards, disregarding punctuation, whitespace, and case of letters).

String Example: Palindromes

Suppose we want to test whether a word or phrase is a palindrome (i.e. a phrase that reads the same backwards or forwards, disregarding punctuation, whitespace, and case of letters).

To do this, let us define some functions to strip out non-letters in the word or phrase, and then to perform the comparisons necessary to check if a string is a palindrome.

String Example: Palindromes

Suppose we want to test whether a word or phrase is a palindrome (i.e. a phrase that reads the same backwards or forwards, disregarding punctuation, whitespace, and case of letters).

To do this, let us define some functions to strip out non-letters in the word or phrase, and then to perform the comparisons necessary to check if a string is a palindrome.

```
function strip(str)
// Assumes: str is a string
// Returns: str with all but letters removed
{
  var copy = "";
  for (var i = 0; i < str.length; i++) {
    if ((str.charAt(i) >= "A" && str.charAt(i) <= "Z") ||
        (str.charAt(i) >= "a" && str.charAt(i) <= "z")) {
      copy += str.charAt(i);
    }
  }
  return copy;
}
```

Palindromes (cont.)

```
function isPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
  var str = strip(str.toUpperCase());

  for(var i = 0; i < Math.floor(str.length/2); i++) {
    if (str.charAt(i) != str.charAt(str.length-i-1)) {
      return false;
    }
  }
  return true;
}
```

Palindromes (cont.)

```
<html>
<!-- COMP519  js09.html  2015.10.04 -->
<head>
 <title>Palindrome Checker</title>
  <script>
       function strip(str)
          {       // CODE AS SHOWN ON PREVIOUS SLIDE      }
       function isPalindrome(str)
          {      // CODE AS SHOWN ON PREVIOUS SLIDE       }
  </script>
</head>
<body>  <p>
  <script>
    var text = prompt("Enter a word or phrase", "Madam, I'm Adam");
    if (isPalindrome(text)) {
      document.write("'" + text + "' <b>is</b> a palindrome.");
    }
    else {
      document.write("'" + text + "' <b>is not</b> a palindrome.");
    }
  </script>  </p>
</body>
</html>
```

view page

JavaScript Arrays

Arrays store a sequence of items, accessible via an index.

Since JavaScript is loosely typed, array elements do not have to be the same type.

To create an array, you may allocate space using new (or you can assign an array directly using the proper syntax shown below).

```
items = new Array(10); // allocates space for 10 items (but can grow)
items = new Array(); // if no size given, will adjust dynamically
items = [0,0,0,0,0,0,0,0,0,0];   // can assign size & values
```

The final method is preferred if possible, as it is quicker and uses less memory.

JavaScript Arrays (cont.)

To access an array element, use C++/Java-like syntax:

```
for (i = 0; i < 10; i++) {
        items[i] = i;       // initializes array with values 0, . . . , 9
    }
```

The length property stores the number of items in the array.

```
for (i = 0; i < items.length; i++) {
     document.write(items[i] + "<br/>"); // displays elements one
                                         // line at a time
    }
```

Array Example

```
<html>
<!-- COMP519  js10.html  2015.10.04 -->
<head>
 <title>Die Statistics</title>
 <script
src="http://www.csc.liv.ac.uk/~martin/teaching/comp519/JS/random.js">
 </script>
</head>
<body>  <p>
 <script>
    var numRolls = 60000;
    var dieSides = 6;
    var rolls = new Array(dieSides+1);
    for (var i = 1; i < rolls.length; i++) { rolls[i] = 0;  }
    for(i = 1; i <= numRolls; i++) {
        rolls[randomInt(1, dieSides)]++;       }
    for (i = 1; i < rolls.length; i++) {
        document.write("Number of " + i + "s = " +
                       rolls[i] + "<br />");
    }
  </script>  </p>
</body>
</html>
```

view page

Arrays (cont.)

Arrays have predefined methods that allow them to be used as stacks, queues, or other common programming data structures.

```
var stack = new Array();
stack.push("blue");     // append the string "blue" to array
stack.push(12);     // stack is now the array ["blue", 12]
stack.push("green");     // stack = ["blue", 12, "green"]
var item = stack.pop();     // item is now equal to "green"
                            // and stack = ["blue", 12] again
var q = [1,2,3,4,5,6,7,8,9,10];
item = q.shift();     // item is now equal to 1, remaining
                      // elements of q move down one position
                      // in the array, e.g. q[0] equals 2
q.unshift(125);      // q is now the array [125,2,3,4,5,6,7,8,9,10]
q.push(244);     // q = [125,2,3,4,5,6,7,8,9,10,244]
```

Date Object

String and Array are the most commonly used objects in JavaScript. Other, more specialized, objects are also pre-defined.

The Date object can be used to access the date and time, and to perform “date arithmetic”.

To create a Date object, use new and supply year, month, and/or day/. . . as desired.

```
var today = new Date();     // sets to current date and time
var newYear = new Date(2015,0,1); //sets to Jan 1, 2015
                                  // 12:00AM
```

Methods include:

```
newYear.getYear()
newYear.getMonth()          newYear.getDay()
newYear.getHours()          newYear.getMinutes()
newYear.getSeconds()        newYear.getMilliseconds()
```

Date Example

```
<html>
<!-- COMP519  js11.html  2015.10.04 -->
<head>   <title>Time page</title>
  <script>
      function pad(s)
      {  var str = new String(s);
         if (str.length < 2)  {   str = "0" + str;  }
         return str;       }
  </script>
</head>
<body>
  <script>
    var now = new Date();
    document.write("<p>Time when page was loaded:  " + now + "</p>");
    var time = "AM";
    if (now.getHours() == 0)  {  var hours = 12;  }
    else if (now.getHours() >= 12) {
        time = "PM";
        var hours = now.getHours();
        if (hours > 12)  {   hours -= 12;  }         }
    document.write("<p>" + pad(hours) + ":" + pad(now.getMinutes())
                   + ":" + time + "</p>");
  </script>    </body>   </html>
```

view page

Another Date Example

```
<html>
<!-- COMP519  js12.html  2015.10.04 -->
<head>
  <title>Elapsed Year Time</title>
</head>
<body>
  <p>Elapsed time in this year:
  <script type="text/javascript">
    var now = new Date();
    var newYear = new Date(now.getFullYear(),0,1);
    var secs = Math.round((now-newYear)/1000);
    var days = Math.floor(secs / 86400);
    secs -= days*86400;
    var hours = Math.floor(secs / 3600);
    secs -= hours*3600;
    var minutes = Math.floor(secs / 60);
    secs -= minutes*60
    document.write(days + " days, " + hours + " hours, " +
                   minutes + " minutes, and " +
                   secs + " seconds.");
  </script>
  </p>
</body>      </html>
```

view page

The document Object

Web browsers allow you to access information about an HTML document using the document object. Methods also exist to modify the current webpage, either by inserting information as the page loads and/or by using JavaScript methods to modify the Document Object Model.

document.write(...) is the method that displays text (or, indeed, any HTML element) in the webpage.

document.URL is a property that gives the location of the current HTML document.

document.lastModified is a property (a String) that gives the date and time the current HTML document was last changed.

The document Object (cont.)

Here is some JavaScript code that creates a Date object with the current date the file was last modified (allowing you to use Date methods to access parts of this object).

```
<script>
var d = new Date(document.lastModified);
var month;

switch ( d.getMonth() )  {
   case 0:  month = "January";
            break;
   case 1:  month = "February";
            break;
   case 2:  month = "March";
            break;
   ...
   ...
   case 11: month = "December";
            break;
}
document.write("Last modified: " + d.getDate() + " " + month +
                          " " + d.getFullYear() );
</script>
```

The document Object (cont.)

Other properties and methods of the document object include:

document.writeln(...) Same as document.write() but adds a newline character at the end of the written object.

document.title returns or sets the title of the webpage.

document.images returns a collection of all images in the document.

document.getElementById() return the HTML element that has the ID attribute with the specified value. (We will use this later when talking about forms and getting user input...)

document.links returns a collection of all <a> and <area> elements that have a href attribute.

The W3Schools webpage gives a description of the common document properties and methods. We will see some more in the future.

User-defined objects

You can define your own objects in JavaScript, but the notation can be somewhat awkward if you want to define private properties and methods (as is easily possible in Java, for example). (I will leave it to those interested to read online discussions on the “best” way to make private or protected properties and methods in JavaScript. Do a search on something like “JavaScript objects private” or similar to see these ideas.)

Define a function that serves as an object constructor.

Specify data fields and methods using the keyword “this”.

A simple example follows.

Example of a User-Defined Object

```
// COMP519        Die.js        2015.10.05 //
// Die class definition
////////////////////////////////////////////

function Die(sides)
{
   this.numSides = sides;
   this.numRolls = 0;
   this.roll = function ()
      {
          this.numRolls++;
          return Math.floor(Math.random()*this.numSides) + 1;
      }
}
```

This constructor makes an object with two properties (variables) and one method. We store this in an external file to reference where desired.

As cautioned, the properties and methods are public.

Example of a User-Defined Object (cont.)

```
<html>
<!-- COMP519  js15.html  2015.10.05 -->
<head>
       <title>Dice page</title>
       <script src="Die.js"></script>
</head>
<body>   <p>
   <script>
      die6 = new Die(6);
      die8 = new Die(8);
      roll6 = -1;     // dummy value to start loop
      roll8 = -2;     // dummy value to start loop
      while (roll6 != roll8) {
         roll6 = die6.roll();
         roll8 = die8.roll();
         document.write("6-sided: " + roll6 +
                        "    " +
                        "8-sided: " + roll8 + "<br>");
      }
      document.write("<br>Number of rolls: " +
                     die6.numRolls);
    </script>   </p>
</body>  </html>
```

view page

What next?

So far, this has been a basic introduction to JavaScript.

But how can we use JavaScript to capture and use information from a user (other than the prompt function)?

Things to introduce:

- Accessing elements on the page using JavaScript functions.
- JavaScript and forms.
- Events, capturing user input.
- The Document Object Model (DOM), and manipulating the webpage.