

CSC 551: Web Programming

Fall 2001

client-side programming with JavaScript

- scripts vs. programs
 - JavaScript vs. JScript vs. VBScript
 - common tasks for client-side scripts
- JavaScript
 - data types & expressions
 - control statements
 - functions & libraries
 - strings & arrays

Client-side programming

recall: HTML is good for developing *static* pages

- can specify text/image layout, presentation, links, ...
- Web page looks the same each time it is accessed
- in order to develop interactive/reactive pages, must integrate programming

client-side programming

- programs are written in a separate programming language
 - e.g., JavaScript, JScript, VBScript
- programs are embedded in the HTML of a Web page, with tags to identify the program component
 - e.g., `<script language="JavaScript"> ... </script>`
- the browser executes the program as it loads the page, integrating the dynamic output of the program with the static content of HTML

Scripts vs. programs

a scripting language is a simple, interpreted programming language

- scripts are embedded as plain text, interpreted by application
- *simpler execution model*: don't need compiler or development environment
- *saves bandwidth*: source code is downloaded, not compiled executable
- *platform-independence*: code interpreted by any script-enabled browser
- *but*: slower than compiled code, not as powerful/full-featured

JavaScript: the first Web scripting language, developed by Netscape in 1995
syntactic similarities to Java/C++, but simpler & more flexible
(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 can (mostly) handle both JavaScript & JScript

JavaScript 1.5 & JScript 5.0 cores conform to ECMAScript standard

VBScript: client-side scripting version of Microsoft Visual Basic

Common scripting tasks

adding dynamic features to Web pages

- validation of form data
- image rollovers
- time-sensitive or random page elements
- handling cookies

defining programs with Web interfaces

- utilize buttons, text boxes, clickable images, prompts, frames

limitations of client-side scripting

- since script code is embedded in the page, 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 designed to run on any machine platform, scripts do not contain platform specific commands
- script languages are not full-featured
e.g., JavaScript objects are crude, not good for large project development

JavaScript

JavaScript code can be embedded in a Web page using SCRIPT tags

- the output of JavaScript code is displayed as if directly entered in HTML

```
<html>
<!-- Dave Reed  js01.html  9/06/01  -->

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

<body>
<script language="JavaScript">
  // silly code to demonstrate output

  document.write("Hello world!");

  document.write("<p>How are <br>" +
                 "<i>you</i>?");
</script>
</body>
</html>
```

[view page in browser](#)

document.write displays text in page

text to be displayed can include HTML tags

the tags are interpreted by the browser when the text is displayed

as in C++/Java, statements end with ;

JavaScript comments similar to C++/Java

// starts a single line comment

/*...*/ enclose multi-line comments

JavaScript data types & variables

JavaScript has only three primitive data types

```
strings : "foo"  'howdy do'  "I said 'hi'."  ""
numbers : 12    3.14159    1.5E6
booleans: true  false
```

```
<html>
<!-- Dave Reed  js02.html  9/06/01  -->

<head>
<title>Data Types and Variables</title>
</head>

<body>
<script language="JavaScript">
  x = 1024;
  document.write("<p>x = " + x);

  x = "foobar";
  document.write("<p>x = " + x);
</script>
</body>
</html>
```

[view page in browser](#)

assignments are as in C++/Java

```
message = "howdy";
pi = 3.14159;
```

variable names are sequences of letters, digits, and underscores: *start with a letter*

variables names are case-sensitive

you don't have to declare variables, will be created the first time used

variables are loosely typed, can assign different types of values

JavaScript operators & expressions

standard C++/Java operators are provided in JavaScript

numeric: + - * / % (remainder)
strings: + (concatenation)
relational: == != < <= > >=
logical: && || !

```
<html>
<!-- Dave Reed js03.html 9/06/01 -->

<head>
  <title>Operators and Expressions</title>
</head>

<body>
  <script language="JavaScript">
    x = 5;
    document.write("x+1 = " + x+1 + "<br>");
    document.write(x + 1 + " = x+1<br>");
    document.write("x+1 = " + (x+1) + "<br>");
  </script>
</body>
</html>
```

[view page in browser](#)

as in C++/Java, precedence rules apply to expressions

(* / %) → (+ -) → (&& || !)

operators are left-associative (evaluated in left-to-right order)

must be careful when mixing strings and numbers

number + number → addition
string + string → concatenation
string + number →
convert number to string, then concatenation

JavaScript control statements

C++/Java control statements are provided in JavaScript

conditional execution:

```
if (BOOLEAN TEST) {
  STATEMENTS;
}

if (BOOLEAN TEST) {
  STATEMENTS;
} else {
  STATEMENTS;
}
```

conditional looping:

```
while (BOOLEAN TEST) {
  STATEMENTS;
}
```

counter-driven looping:

```
for (INITIALIZE; TEST; UPDATE) {
  STATEMENTS;
}
```

JavaScript example

```
<html>
<!-- Dave Reed  js04.html  9/06/01 -->

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

<body>
  <script language="JavaScript">
    distanceToSun = 93300000*5280*12;
    thickness = .002;

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

[view page in browser](#)

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?

Note: shorthand expressions are provided as in C++/Java

*= /= += -= ++ --

JavaScript Math routines

```
<html>
<!-- Dave Reed  js05.html  9/06/01 -->

<head>
  <title>Mystery Program</title>
</head>

<body>
  <script language="JavaScript">
    maxRange = 100;

    for(i = 1; i <= maxRange; i++) {
      if (Math.pow(Math.floor(Math.sqrt(i)),2) == i) {
        document.write(i + "<br>");
      }
    }
  </script>
</body>
</html>
```

[view page in browser](#)

the predefined Math object contains routines and constants

Math.sqrt
Math.pow
Math.abs
Math.max
Math.min
Math.floor
Math.ceil
Math.round

Math.PI
Math.E

QUESTION: what does this program do?

Random page elements

```
<html>
<!-- Dave Reed  js06.html  9/06/01 -->

<head>
  <title>Random Dice Rolls</title>
</head>

<body>
  <div align="center">
    <script language="JavaScript">
      roll1 = Math.floor(Math.random()*6) + 1;
      roll2 = Math.floor(Math.random()*6) + 1;

      document.write("<img src='http://www.creighton.edu/"+
        "~csc551/Images/die" + roll1 + ".gif'>");
      document.write("&nbsp;&nbsp;&nbsp;");
      document.write("<img src='http://www.creighton.edu/"+
        "~csc551/Images/die" + roll2 + ".gif'>");
    </script>
  </div>
</body>
</html>
```

`Math.random` function returns a pseudo-random number in the range [0..1)

can alter the range using other Math routines

useful for generating dynamic page elements

[view page in browser](#)

Interactive pages using prompt

```
<html>
<!-- Dave Reed  js07.html  9/06/01 -->

<head>
  <title>Interactive page</title>
</head>

<body>
  <script language="JavaScript">
    userName = prompt("What is your name?", "");

    document.write("Hello " + userName +
      ", welcome to my Web page.");
  </script>
</body>
</html>
```

somewhat crude interaction with the user can take place using the `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 function returns the value entered by the user in the dialog box

[view page in browser](#)

forms will provide a better interface for user interaction (later)

Prompting for numbers

```
<html>
<!-- Dave Reed  js08.html  9/06/01  -->

<head>
  <title>Prompting for numbers</title>
</head>

<body>
  <script language="JavaScript">
    num1 = prompt("Enter the first number", "1");
    num1 = parseFloat(num1);

    num2 = prompt("Enter the second number", "2");
    num2 = parseFloat(num2);

    document.write("The sum of the numbers is " +
                   (num1 + num2));

  </script>
</body>
</html>
```

Note: prompt always returns a string

if the user enters the number 12 at the prompt, the string "12" is returned

recall: + applied to strings gives concatenation

if numbers are to be read using prompt, they must be explicitly converted to numbers using parseFloat

[view page in browser](#)

User-defined functions

function definitions are similar to C++/Java, except:

- no return type for the function (since variables are loosely typed)
- no types for parameters (since variables are loosely typed)
- by-value parameter passing only (parameter gets copy of argument)

```
function isPrime(n)
// Assumes: n > 0
// 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;
  }
}
```

can limit variable scope

if the first use of a variable is preceded with var, then that variable is local to the function

for modularity, should make all variables in a function local

Function example

```
<html>
<!-- Dave Reed  js09.html  9/06/01  -->
<head>
  <title>Prime Tester</title>
  <script language="JavaScript">
    function isPrime(n)
      // Assumes: n > 0
      // Returns: true if n is prime
      {
        // CODE AS SHOWN ON PREVIOUS SLIDE
      }
  </script>
</head>
<body>
  <script language="JavaScript">
    testNum = prompt("Enter a positive integer", "7");
    testNum = parseFloat(testNum);

    if (isPrime(testNum)) {
      document.write(testNum + " <b>is</b> a prime number.");
    }
    else {
      document.write(testNum + " <b>is not</b> a prime number.");
    }
  </script>
</body>
</html>
```

function definitions go in the HEAD

HEAD is loaded first, so the function is defined before code in the BODY is executed

[view page in browser](#)

```
<html>
<!-- Dave Reed  js10.html  9/06/01  -->
<head>
  <title> Random Dice Rolls Revisited</title>
  <script language="JavaScript">
    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 align="center">
    <script language="JavaScript">
      roll1 = randomInt(1, 6);
      roll2 = randomInt(1, 6);

      document.write("<img src='http://www.creighton.edu/~csc551/Images/die' + roll1 + ".gif'>");
      document.write("&nbsp;&nbsp;&nbsp;");
      document.write("<img src='http://www.creighton.edu/~csc551/Images/die' + roll2 + ".gif'>");
    </script>
  </div>
</body>
</html>
```

Another example

recall the dynamic dice page

could define a function for generating random numbers in a range, then use whenever needed

easier to remember, promotes reuse

[view page in browser](#)

JavaScript libraries

better still: if you define functions that may be useful to many pages, store in a separate library file and load the library when needed

the file at <http://www.creighton.edu/~csc551/JavaScript/random.js> contains definitions of the following functions:

<code>randomNum(low, high)</code>	returns random real in range [low..high)
<code>randomInt(low, high)</code>	returns random integer in range [low..high)
<code>randomChar(string)</code>	returns random character from the string
<code>randomOneOf([item1, ..., itemN])</code>	returns random item from list/array

Note: as with style sheet files, no tags in the JavaScript library file

load a library using the SRC attribute in the SCRIPT tag (nothing between the tags)

```
<SCRIPT LANGUAGE="JavaScript"
      SRC="http://www.creighton.edu/~csc551/JavaScript/random.js">
</SCRIPT>
```

Library example

```
<html>
<!-- Dave Reed  js11.html  9/06/01  -->

<head>
  <title> Random Dice Rolls Revisited</title>

  <script language="JavaScript"
    SRC="http://www.creighton.edu/~csc551/JavaScript/random.js"
  </script>
</head>

<body>
  <div align="center">
    <script language="JavaScript">
      roll1 = randomInt(1, 6);
      roll2 = randomInt(1, 6);

      document.write("<img src='http://www.creighton.edu/" +
        "~csc551/Images/die" + roll1 + ".gif'">");
      document.write("&nbsp;&nbsp;&nbsp;");
      document.write("<img src='http://www.creighton.edu/" +
        "~csc551/Images/die" + roll2 + ".gif'">");
    </script>
  </div>
</body>
</html>
```

[view page in browser](#)

JavaScript strings

strings are objects in JavaScript (i.e., instances of a class)

- each string has properties (data fields) and methods (member functions)

string properties include

`length` : stores the number of characters in the string

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

properties/methods are called exactly as in C++/Java

`word.length` `word.charAt(0)`

String example (pt. 1)

suppose we want to test whether a word or phrase is a palindrome

e.g., *radar* *Bob* *noon*

```
function IsPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
  str = 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;
}
```

must traverse the string,
comparing characters
from front to back

should be case-
insensitive, so make all
letters uppercase before
testing

String example (pt. 2)

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

function IsPalindrome(str)
// Assumes: str is a string
// Returns: true if str is a palindrome, else false
{
  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;
}
```

better yet, we would like to be able to test phrases

Madam, I'm Adam.

A man, a plan, a canal: Panama!

must strip non-letters out of the phrase, then test as before

to handle phrases, must be able to strip out non-letters

```
<html>
<!-- Dave Reed  js12.html  9/06/01  -->

<head>
<title>Palindrome Checker</title>

<script language="JavaScript">
  function Strip(str)
  {
    // CODE AS SHOWN ON PREVIOUS SLIDE
  }

  function IsPalindrome(str)
  {
    // CODE AS SHOWN ON PREVIOUS SLIDE
  }
</script>
</head>

<body>
<script language="JavaScript">
  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>
</body>
</html>
```

[view page in browser](#)

JavaScript arrays

arrays store a sequence of items, accessible via an index

since JavaScript is loosely typed, elements do not have to be the same type

- to create an array, allocate space using `new` (or can assign directly)

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

- to access an array element, use `[]` (as in C++/Java)

```
for (i = 0; i < 10; i++) {
    items[i] = 0;           // stores 0 at each index
}
```

- the `length` attribute stores the number of items in the array

```
for (i = 0; i < items.length; i++) {
    document.write(items[i] + "<BR>"); // displays elements
}
```

```
// File: random.js
// Author: Dave Reed
// Date: 9/10/01
////////////////////////////////////

function randomNum(low, high)
// Assumes: low <= high
// Returns: a random number in the range [low, high)
{
    return Math.random()*(high-low) + low;
}

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

function randomChar(str)
// Assumes: str is a nonempty string
// Returns: a random character from the string
{
    return str.charAt(randomInt(0, str.length-1));
}

function randomOneOf(list)
// Assumes: list is a nonempty list (array)
// Returns: a random item from the list
{
    return list[randomInt(0, list.length-1)];
}
```

Array example

the `randomOneOf`
function in the
`random.js` library

takes an array as input

selects a random index
(based on `list.length`)

returns item at that index
(using `[]`)

```

<html>
<!-- Dave Reed  js13.html  9/06/01  -->

<head>
<title>Die Statistics</title>

<script language="JavaScript"
  src="http://www.creighton.edu/~csc551/JavaScript/random.js">
</script>
</head>

<body>
<script language="JavaScript">
  numRolls = 60000;
  dieSides = 6;

  rolls = new Array(dieSides+1);
  for (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>
</body>
</html>

```

Another array example

suppose we want to simulate die rolls and verify even distribution

keep an array of counters:

initialize each count to 0

each time you roll X, increment rolls[X]

display each counter

[view page in browser](#)

Useful function

often, you will want to read a sequence of input values

- e.g., homework grades, a multi-word name or phrase, ...

entering each value at a separate prompt is tedious, lots of variables

also, may not know exactly how many values are to be entered

SOLUTION: allow the user to enter multiple values in a single prompt dialog

- must define a function which takes the string and separates the pieces

```

function Arrayify(str)
// Assumes: str is a sequence of words, separated by spaces
// Returns: an array containing the individual words

```

e.g.,

```

grades = "80 85 95";
gradeArray = Arrayify(grades); // assigns ["80","85","95"]

```

Arrayify

takes entire input as a string

first creates empty array & initializes the item count

while input string is not empty, traverses to find the first non-space (or the end)

if not at end of input string, traverses and appends together all chars up to a space (or the end)

stores the resulting item in the array, adds to count

removes those chars from the input string

finally, returns the array

```
function Arrayify(str)
// Assumes: str is a sequence of words, separated by spaces
// Returns: an array containing the individual words
{
  var items = new Array();
  var count = 0;

  while (str != "") {
    var index = 0;
    while (index < str.length && str.charAt(index) == " ") {
      index++;
    }

    if (index < str.length) {
      var item = "";
      while (index < str.length && str.charAt(index) != " ") {
        item += str.charAt(index);
        index++;
      }

      items[count] = item;
      count++;
    }

    str = str.substring(index+1, str.length);
  }

  return items;
}
```

Next week...

more JavaScript

- JavaScript objects (e.g., document, Date)
- object-based programming
- event-driven programs

read Chapters 15 and 16

as always, be prepared for a quiz on

- today's lecture (moderately thorough)
- the reading (superficial)