CSE417: WEB ENGINEERING

Daffodil International University

Learning Outcome

- ✓ Understand Javascript
- ✓ Apply basic JS

Contents

Client-Side Programming

- 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 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 tags to identify the program component

    e.g., `<script type="text/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
  - could allow the user to input information and process it, might be used to validate input before it's submitted to a remote server

Scripts vs. Programs

- a scripting language is a simple, <u>interpreted</u> 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 in some respects, limited in others

(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, and Firefox 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

Common Scripting Tasks

- adding dynamic features to Web pages
  - validation of form data (probably the most commonly used application)
  - image rollovers
  - time-sensitive or random page elements
  - handling cookies

- defining programs with Web interfaces
  - utilize buttons, text boxes, clickable images, prompts, etc

- 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, database

    since they are 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 very 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>
<!-- CSE391  js01.html   -->

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

<body>
  <script type="text/javascript">
    // silly code to demonstrate output

    document.write("Hello world!");

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

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

`document.write` displays text in the 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 `;`
but a line break is also interpreted as the end of a statement

JavaScript comments similar to C++/Java

`//` starts a single line comment

`/*…*/` enclose multi-line comments

JS Basics…

- Anatomy of <script> tag:
    - //MIME: Multipurpose Internet Mail Extensions
    - <script type=”MIME_type”> ...script content… </script>

- Display data:
    - Writing into an HTML element, using innerHTML.
    - Writing into the HTML output using document.write().
    - Writing into an alert box, using window.alert().
    - Writing into the browser console, using console.log().

JavaScript Data Types & Variables

- JavaScript has only three primitive data types

String :	`"foo"`	`'howdy do'`	`"I said 'hi'."`	`""`
Number:	`12`	`3.14159`	`1.5E6`	
Boolean :	`true`	`false`	`*Find info on Null, Undefined`	

```
<html>
<!-- CSE391  js02.html   -->

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

<body>
  <script type="text/javascript">
    var x, y;
    x= 1024;

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

assignments are as in C++/Java

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

variable names are sequences of letters, digits, and underscores that *start with a letter or an underscore*

variables names are <u>case-sensitive</u>

you don't have to declare variables, will be created the first time used, but it's better if you use var statements

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

variables are loosely typed, can be assigned different types of values

Primitive Data types, Variables, Constant

- Primitive Data types:
    - JavaScript supports the following data types: Booleans,Integers, Floating-point numbers, Strings, Arrays, null, undefined
    - JavaScript also supports objects — an unordered container of various data types.
    - //var person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
    - Dynamically and weakly/loosely typed.
- Variables:
    - Case sensitive and can contain a mix of letters and numbers
    - Example: var x = 20; var firstname = “Steve”;
    - Note that variables with global scope transcend <script> sections; that is, variables declared within one <script> section are accessible in other <script> sections within the document.
    - Keyword let: { let x = 2; }// x can NOT be used outside the scope
- Constants:
    - const country = ‘Bangladesh’;

Type Conversion

- typeof operator to find the data type of a JavaScript variable.
- using JavaScript function
  - global method, String(123), 123 converted to string
  - global method Number() can convert strings to numbers
- Automatically by JavaScript itself
  - "loosely typed" language, **whenever an operator or statement is expecting a particular data-type, JavaScript will automatically convert the data to that type**.
  - The if() statement in the first example expects a boolean value, therefore whatever you define in the brackets will be converted to a boolean. The same is true for while() and do...while() statements.

JavaScript Operators & Control Statements

```
<html>
<!-- CSE391 js03.html   -->

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

<body>
 <script type="text/javascript">
    distanceToSun = 93.3e6*5280*12;
    thickness = .002;

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

standard C++/Java operators & control statements are provided in JavaScript

- `+, -, *, /, %, ++, --, …`
- `==, !=, <, >, <=, >=`
- `&&, ||, !,===,!==`
- if-then, if-then-else, switch
- while, for, do-while, …

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?

*Find more info on this subject

JavaScript Math Routines

```
<html>
<!-- CSE391   js04.html    -->

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

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

      document.write("<img src='Images/die" +
                roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='Images/die" +
                roll2 + ".gif'/>");
    </script>
  </div>
</body>
</html>
```

the Math object contains functions and constants

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

Math.PI
Math.E
```

`Math.random` function returns number in [0..1)

User-Defined Functions

- function definitions are similar to C++/Java, except:
  - no return type 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)

```
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>
<!-- js06.html   -->

<head>
  <title>Prime Tester</title>

  <script type="text/javascript">
    function isPrime(n)
    // Assumes: n > 0
    // Returns: true if n is prime
    {
      // CODE AS SHOWN ON PREVIOUS SLIDE
    }
  </script>
</head>

<body>
 <script type="text/javascript">
    testNum = parseFloat(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>
</body>
</html>
```

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

<head> section is loaded first, so then the function is defined before code in the <body> is executed

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

- Assume a file named `random.js[code, next slide]`

  contains definitions of the following functions:

`RandomNum(low, high)`	returns random real in range [low..high)
`RandomInt(low, high)`	returns 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

load a library using the SRC attribute in the SCRIPT tag (nothing between the beginning and ending tag)

```
<script type="text/javascript"
        src="JS/random.js">
</script>
```

random.js

```
// File:   random.js
// Author:
// Date:
//
// This file contains several routines for generating
random values
//////////////////////////////////////////////////////////

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

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

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

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

Library Example

```
<html>
<!--js08.html   -->

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

  <script type="text/javascript"
    src="JS/random.js">
  </script>
</head>

<body>
  <div align="center">
    <script type="text/javascript">
      roll1 = RandomInt(1, 6);
      roll2 = RandomInt(1, 6);

      document.write("<img src=CSE391/Images/die" +
                     roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src=CSE391/Images/die" +
                     roll2 + ".gif'/>");

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

JavaScript Strings

- a *class* defines a new type (formally, *Abstract Data Type*)
  - encapsulates data (properties) and operations on that data (methods)

- a String encapsulates a sequence of characters, enclosed in quotes

properties include

`length`	: stores 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

to create a string, assign using `new` or just make a direct assignment (`new` is implicit)

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

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

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

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 given, 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` property stores the number of items in the array

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

More on Arrays…

var points = new Array(40, 100);
  - Creates an array with two elements (40 and 100)

Var points = new Array(40);
  - Creates an array with 40 undefined elements !

- JavaScript arrays can contain a mix of the various data types and are declared using the new operator in your declaration statement, as in the following:
  - A = new Array();
  - To ensure use, a = new String(); // for string type

- Arrays with named indexes are called associative arrays (or hashes).
  - **JS doesn’t support arrays with named indexes**, always numbered index
- **Objects use named indexes**

Date Class

- String & Array are the most commonly used classes in JavaScript
  - other, special purpose classes & objects also exist

- the Date class can be used to access the date and time
  - to create a Date object, use new & supply year/month/day/… as desired

```
today = new Date();           // sets to current date & time

newYear = new Date(2002,0,1); //sets to Jan 1, 2002  12:00AM
```

  - methods include:

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

can access individual components of a date

Date Example

```
<html>
<!-- CSE391  js11.html   -->

<head>
  <title>Time page</title>
</head>

<body>
  Time when page was loaded:
  <script type="text/javascript">
    now = new Date();

    document.write("<p>" + now + "</p>");

    time = "AM";
    hours = now.getHours();
    if (hours > 12) {
        hours -= 12;
        time = "PM"
    }
    else if (hours == 0) {
        hours = 12;
    }
    document.write("<p>" + hours + ":" +
                   now.getMinutes() + ":" +
                   now.getSeconds() + " " +
                   time + "</p>");
  </script>
</body>
</html>
```

by default, a date will be displayed in full, e.g.,

```
Sun Feb 03 22:55:20 GMT-0600
(Central Standard Time) 2002
```

can pull out portions of the date using the methods and display as desired

here, determine if "AM" or "PM" and adjust so hour between 1-12

```
10:55:20 PM
```

Exercise

- **Exercise**
  - Write a function which finds the fibonacci series for n terms.
  - Test a word if it is a palindrome or not using JS String library.
  - Write short notes on JS promt.
- **READINGS**
  - M Schafer: Ch. 19, 20

Acknowledgement

- This module is designed and created with the help from following sources-
    - https://cgi.csc.liv.ac.uk/~ullrich/COMP519/
    - http://www.csc.liv.ac.uk/~martin/teaching/comp519/