IT4409: Web Technologies and e-Services

2020-1

JavaScript

Instructor: Dr. Thanh-Chung Dao
Slides by Dr. Binh Minh Nguyen

Department of Information Systems
School of Information and Communication Technology
Hanoi University of Science and Technology

1

Content

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
  - ➢ Date, document, navigator, user-defined classes

2

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 (HTML) 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 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

3

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 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, 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

4

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
  - 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

5

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>
<!-- CS443  js01.html  16.08.06 -->

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

<body>
  <script type="text/javascript">
    // 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

`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 might also be interpreted as the end of a statement (depends upon browser)

JavaScript comments similar to C++/Java

`//` starts a single line comment

`/*…*/` enclose multi-line comments

6

JavaScript Data Types & Variables

- JavaScript has only three primitive data types

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

```
<html>
<!-- CS443  js02.html  16.08.06 -->

<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>
```

view page

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 <u>first time</u> 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 (Danger!)

JavaScript Operators & Control Statements

```
<html>
<!-- CS443 js03.html  08.10.10 -->

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

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

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

view page

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

- `+, -, *, /, %, ++, --, …`
- `==, !=, <, >, <=, >=`
- `&&, ||, !,===,!==`
- if , if-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?

*Lots of information is available online

JavaScript Math Routines

```
<html>
<!-- CS443  js04.html  08.10.10 -->

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

<body>
  <div style="text-align:center">
    <script type="text/javascript">
      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/CS443/Images/die" +
           roll1 + ".gif' alt='dice showing ' + roll1 />");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
           "~martin/teaching/CS443/Images/die" +
           roll2 + ".gif' alt='dice showing ' + roll2 />");
    </script>
  </div>
</body>
</html>
```

view page

the built-in 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 a real number in [0..1)

9

Interactive Pages Using Prompt

```
<html>
<!-- CS443  js05.html  08.10.10 -->

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

<body>
<script type="text/javascript">
 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>The rest of the page...</p>
</body>
</html>
```

view page

crude user interaction can take place using prompt

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 (a string)

if value is a number, must use parseFloat (or parseInt) to convert

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

10

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 to the function.

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

11

Function Example

```
<html>
<!-- CS443  js06.html  16.08.2006 -->

<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>
```

view page

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 (and, therefore, the function can be used later in the body of the HTML document)

12

Another Example

```
<html>
<!-- CS443  js07.html  11.10.2011 -->
<head>
  <title> Random Dice Rolls Revisited</title>
  <script type="text/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 style="text-align: center">
    <script type="text/javascript">
      roll1 = randomInt(1, 6);
      roll2 = randomInt(1, 6);

      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll2 + ".gif'/>");
</script>
  </div>
</body>
</html>
```

view page

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

13

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

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

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

14

Library Example

```
<html>
<!-- CS443  js08.html  11.10.2011 -->

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

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

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

      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll1 + ".gif'/>");
      document.write("&nbsp;&nbsp;");
      document.write("<img src='http://www.csc.liv.ac.uk/"+
                     "~martin/teaching/CS443/Images/die" +
                     roll2 + ".gif'/>");

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

view page

15

JavaScript Objects

- an object defines a new type (formally, *Abstract Data Type*)
  - encapsulates data (properties) and operations on that data (methods)
- a String object 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 (in this case) 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)
```

16

String example: Palindromes

```
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;
}

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;
}
```

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

noon   *Radar*
Madam, I'm Adam.
A man, a plan, a canal: Panama!

must strip non-letters out of the word or phrase

make all chars uppercase in order to be case-insensitive

finally, traverse and compare chars from each end

17

```
<html>
<!-- CS443  js09.html  11.10.2011 -->

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

  <script type="text/javascript">
    function strip(str)
    {
       // CODE AS SHOWN ON PREVIOUS SLIDE
    }

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

<body>
  <script type="text/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

18

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
}
```

19

Array Example

```
<html>
<!-- CS443  js10.html  11.10.2011 -->
<head>
 <title>Dice Statistics</title>
 <script type="text/javascript"
src="http://www.csc.liv.ac.uk/~martin/teaching/CS443/JS/ran
dom.js">
 </script>
</head>
<body>
 <script type="text/javascript">
    numRolls = 60000;
    diceSides = 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>
```

view page

suppose we want to simulate dice 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

20

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");
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"

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]
```

21

Date Object

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

22

Date Example

```
<html>
<!-- CS443  js11.html  16.08.2006 -->

<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
```

view page

Another Example

```
<html>
<!-- CS443  js12.html  12.10.2012 -->

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

<body>
  <p>Elapsed time in this year:
  <script type="text/javascript">
    now = new Date();
    newYear = new Date(2012,0,1);

    secs = Math.round((now-newYear)/1000);

    days = Math.floor(secs / 86400);
    secs -= days*86400;
    hours = Math.floor(secs / 3600);
    secs -= hours*3600;
    minutes = Math.floor(secs / 60);
    secs -= minutes*60

    document.write(days + " days, " +
                   hours + " hours, " +
                   minutes + " minutes, and " +
                   secs + " seconds.");
  </script>
  </p>
</body>
</html>
```

you can add and subtract Dates: the result is a number of milliseconds

here, determine the number of seconds since New Year's day (<u>note</u>: January is month 0)

divide into number of days, hours, minutes and seconds

view page

Document Object

Internet Explorer, Firefox, Opera, etc. allow you to access information about an HTML document using the `document` object

```
<html>
<!-- CS443  js13.html  2.10.2012 -->

<head>
  <title>Documentation page</title>
</head>

<body>
  <table width="100%">
    <tr>
      <td><i>
        <script type="text/javascript">
            document.write(document.URL);
        </script>
      </i></td>
      <td style="text-align: right;"><i>
        <script type="text/javascript">
            document.write(document.lastModified);
        </script>
      </i></td>
    </tr>
  </table>
</body>
</html>
```

`document.write(…)`
method that displays text in the page

`document.URL`
property that gives the location of the HTML document

`document.lastModified`
property that gives the date & time the HTML document was last changed

view page

25

User-Defined Objects

- can define new objects, but the notation can be somewhat awkward
  - simply define a function that serves as a constructor
  - specify data fields & methods using `this`
  - no data hiding: can't protect data or methods

```
// CS443      Die.js      11.10.2011 //
// Die class definition
////////////////////////////////////////////

function Die(sides)
{
   this.numSides = sides;
   this.numRolls = 0;
   this.roll = roll;   // define a pointer to a function
}

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

define `Die` function (i.e., the object's constructor)

initialize data fields in the function, preceded with "this"

similarly, assign method to separately defined function (which uses this to access data)

26

Object Example

```
<html>
<!-- CS443  js15.html  11.10.2011 -->

<head>
  <title>Dice page</title>

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

<body>
 <script type="text/javascript">
    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 +
                     "&nbsp;&nbsp;&nbsp;&nbsp;" +
                     "8-sided: " + roll8 + "<br />");
    }

    document.write("<br />Number of rolls: " +
                   die6.numRolls);
  </script>
</body>
</html>
```

create a Die object using new (similar to String and Array)

here, the argument to Die initializes numSides for that particular object

each Die object has its own properties (numSides & numRolls)

Roll(), when called on a particular Die, accesses its numSides property and updates its NumRolls

view page

27

JavaScript and HTML validators

•In order to use an HTML validator, and not get error messages from the JavaScript portions, you must "mark" the JavaScipt sections in a particular manner. Otherwise the validator will try to interpret the script as HTML code.

•To do this, you can use a markup like the following in your inline code (this isn't necessary for scripts stored in external files).

```
<script type="text/javascript">
   //  <![CDATA[

  document.write("<p>The quick brown fox jumped over the lazy dogs.</p>");
            //    **more code here, etc.

   //  ]]>
</script>
```

28

•Since the (new) XHTML standard is written as an XML application, validators such as the one from the W3C are actually attempting to check an XML document for the correct structure.

•The two tags <![CDATA[and]]> together form an XML directive, meaning to interpret the data between them as literal (non-parsed) "character data". An XML validator will effectively ignore the data between these two tags, meaning that any symbols that would result in an invalid document structure are ignored and do not result in an error message from the validator.

•Because we are using these tags inside of a JavaScript block, and they are not JavaScript commands, we precede each of them with a (JavaScript) comment marker, hence the two forward slashes before each tag.

29

More to learn…

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

30

email: chungdt@soict.hust.edu.vn

Q&A

31