CITS1231 Web Technologies
JavaScript Math, String, Array, Number, Debugging
Last Lecture

- Introduction to JavaScript
- Variables
- Operators
- Conditional Statements
- Program Loops
- Popup Boxes
- Functions
This Lecture

- Functions continued
- Inbuilt objects
  - Math, String, Array, Number
- Debugging JavaScript
JavaScript References

• HTML, XHTML and XML (3rd edition - comprehensive), by Patrick Carey, Course Technology.

• New Perspectives on JavaScript, by Patrick Carey, Frank Canovatchel - in reserve collection.

• [http://www.w3schools.com/js/js_intro.asp](http://www.w3schools.com/js/js_intro.asp)
Creating a Function to Perform an Action

- Syntax of a JavaScript function

  ```javascript
  function function_name(parameters){
    JavaScript commands
  }
  ```

- Calling a function

  ```javascript
  function_name(parameter values)
  ```
Creating a Function to Return a Value

- For a function to return a value
  - It must include a return statement

- Syntax of a function that returns a value
  
  ```javascript
  function function_name(parameters) {
    JavaScript commands
    return value;
  }
  ```
Lifetime of Variables

• Local variables
  – When you declare a variable within a function, the variable can only be accessed within that function.
  – When you exit the function, the variable is destroyed.
  – You can have local variables with the same name in different functions, because each is recognized only by the function in which it is declared.

• Global variables
  – If you declare a variable outside a function, all the functions on your page can access it.
  – The lifetime of these variables starts when they are declared, and ends when the page is closed.
JavaScript Function Example

```javascript
<head>
<script type="text/javascript">
    var curMax = 0;
    function product(a,b) {
        var result = a*b;
        return result;
    }
    function sum(a,b) {
        var result = a+b;
        return result;
    }
    function updateMax(a) {
        if (a > curMax) {
            curMax = a;
        }
    }
</script>
</head>
<body>
<script type="text/javascript">
    var num1 = 2;
    var num2 = 5;
    document.writeln("Product of \"+num1+\" and \"+num2+\" = \"+product(num1,num2));
    document.writeln("Sum of \"+num1+\" and \"+num2+\" = \"+sum(num1,num2));
    updateMax(num1);
    updateMax(num2);
    document.writeln("Current max number i= \"+curMax);
</script>
</body>
```

Global variable: Can be accessed from any JavaScript on page.

Local variables; Same name ok in different functions.
JavaScript Control Flow

• The previous lecture has briefly touched on the JavaScript Control Flow, namely
  – Conditional statements
    • if, if else ...
    • switch
  – Iteration or looping
    • for
    • while
JavaScript Objects

- In-built Global Objects
  - Math, String, Array, Number (Rest of this lecture)
  - Date, RegExp, navigator, Event, screen (Next lecture)
- Document Objects (Subsequent lecture)
Window Object

• Window is the root class, so no need to explicitly specify the class before calling its functions.

• There are a set of functions that can create pop-up windows to issue warnings, confirm information and request for user input. See last lecture for examples.
  - alert()
  - confirm()
  - prompt()
Array

- Allows one to use one variable to hold more than one value at a time.
- Instead of:
  ```javascript
  var fruit1 = "apple";
  var fruit2 = "orange";
  var fruit3 = "banana";
  ```
- One can write:
  ```javascript
  var fruits = new Array();   // create a new array object
  fruits[0] = "apple";       // first item
  fruits[1] = "orange";      // second item
  fruits[2] = "banana";      // third item
  ```
- This allows one to use for loops to iterate through whole list.
- Extremely useful if list is very long (eg 10’s or 100’s items)
Three Ways to Create Arrays

- **Regular:**
  ```javascript
  var fruits = new Array();       // create a new array object
  fruits[0] = "apple";            // first item
  fruits[1] = "orange";           // second item
  ```

- **Condensed:**
  ```javascript
  var fruits = new Array( "apple", "orange" );
  ```

- **Literal:**
  ```javascript
  var fruits = [ "apple", "orange" ];
  ```
Accessing and Modifying Array

• One can access an array element by referring to its name and index number:

```javascript
var fruits = new Array(“apple”, “orange”);
document.writeln(“Second fruit = “ + fruits[1]);
```

Output: orange

• One can easily modify an array element:

```javascript
var fruits = new Array(“apple”, “orange”);
fruits[1] = “grape”;
document.writeln(“Second fruit = “ + fruits[1]);
```

Output: grape
Other Array Methods

- **length**: Sets or returns the number of elements in an array
- **concat()**: Joins two or more arrays, and returns a copy of the joined arrays
- **join()**: Joins all elements of an array into a string
- **pop()**: Removes the last element of an array, and returns that element
- **push()**: Adds new elements to the end of an array, and returns the new length
- **reverse()**: Reverses the order of the elements in an array
- **shift()**: Removes the first element of an array, and returns that element
- **slice()**: Selects a part of an array, and returns the new array
- **sort()**: Sorts the elements of an array
- **splice()**: Adds/Removes elements from an array
- **toString()**: Converts an array to a string, and returns the result
- **unshift()**: Adds new elements to the beginning of an array, and returns the new length
- **valueOf()**: Returns the primitive value of an array
Array concat() and length

```html
<html><body>
<pre>
<script type="text/javascript">
var fruits = new Array( "apple", "orange");
var veges = new Array( "carrot", "lettuce", "spinach");

var shopping = fruits.concat(veges);

document.writeln("Shopping list of "+shopping.length+" items:");
document.writeln(shopping);
document.writeln("\n");

document.writeln("Using for loop to list items:");
document.writeln("<ul>");
for(var i=0; i<shopping.length; i++)
{
    document.writeln("<li>"+shopping[i]+"</li>");
}
document.writeln("</ul>");

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

concat() – combines arrays

length – gives array length
Math Object

- JavaScript provides built in mathematical methods through its Math objects. The math methods are applied in the following manner,

  \[ \text{SinOfAngle} = \text{Math.sin(Angle)}; \]

- Some important methods are Math

  Math.abs(value) \hspace{1cm} \text{(absolute value)}
  Math.sin(value) \hspace{1cm} \text{(sine of angle value)}
  Math.round(value) \hspace{1cm} \text{(round value to nearest integer)}
  Math.ceil(value) \hspace{1cm} \text{(round up to next integer)}
  Math.floor(value) \hspace{1cm} \text{(round down to next integer)}. 
# Math Methods

<table>
<thead>
<tr>
<th>Math Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.abs(x)</td>
<td>Returns the absolute value of x</td>
</tr>
<tr>
<td>Math.acos(x)</td>
<td>Returns the arc cosine of x in radians</td>
</tr>
<tr>
<td>Math.asin(x)</td>
<td>Returns the arc sine of x in radians</td>
</tr>
<tr>
<td>Math.atan(x)</td>
<td>Returns the arc tangent of x in radians</td>
</tr>
<tr>
<td>Math.atan2(x, y)</td>
<td>Returns the angle between the x-axis and the point (x, y)</td>
</tr>
<tr>
<td>Math.ceil(x)</td>
<td>Returns x rounded up to the next highest integer</td>
</tr>
<tr>
<td>Math.cos(x)</td>
<td>Returns the cosine of x</td>
</tr>
<tr>
<td>Math.exp(x)</td>
<td>Returns $e^x$</td>
</tr>
<tr>
<td>Math.floor(x)</td>
<td>Returns x rounded down to the next lowest integer</td>
</tr>
<tr>
<td>Math.log(x)</td>
<td>Returns the natural logarithm of x</td>
</tr>
<tr>
<td>Math.max(x, y)</td>
<td>Returns the larger of x and y</td>
</tr>
<tr>
<td>Math.min(x, y)</td>
<td>Returns the smaller of x and y</td>
</tr>
<tr>
<td>Math.pow(x, y)</td>
<td>Returns $x^y$</td>
</tr>
<tr>
<td>Math.random()</td>
<td>Returns a random number between 0 and 1</td>
</tr>
<tr>
<td>Math.round(x)</td>
<td>Returns x rounded to the nearest integer</td>
</tr>
<tr>
<td>Math.sin(x)</td>
<td>Returns the sine of x</td>
</tr>
<tr>
<td>Math.sqrt(x)</td>
<td>Returns the square root of x</td>
</tr>
<tr>
<td>Math.tan(x)</td>
<td>Returns the tangent of x</td>
</tr>
</tbody>
</table>
# Math Constants

<table>
<thead>
<tr>
<th>Math Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math.E</td>
<td>The natural logarithm base, e (approximately 2.7183)</td>
</tr>
<tr>
<td>Math.LN10</td>
<td>The natural logarithm of 10 (approximately 2.3026)</td>
</tr>
<tr>
<td>Math.LN2</td>
<td>The natural logarithm of 2 (approximately 0.6931)</td>
</tr>
<tr>
<td>Math.LOG10E</td>
<td>The base 10 logarithm of e (approximately 0.4343)</td>
</tr>
<tr>
<td>Math.LOG2E</td>
<td>The base 2 logarithm of e (approximately 1.4427)</td>
</tr>
<tr>
<td>Math.PI</td>
<td>The value $\pi$ (approximately 3.1416)</td>
</tr>
<tr>
<td>Math.SQRT1_2</td>
<td>The value of 1 divided by the square root of 2 (approximately 0.7071)</td>
</tr>
<tr>
<td>Math.SQRT2</td>
<td>The value of the square root of 2 (approximately 1.4142)</td>
</tr>
</tbody>
</table>
Global Numeric Functions

- **isFinite()** - determines if a value is a finite, legal number
  - `isFinite(1.23);` Returns: **true**
  - `isFinite(“hello”);` Returns: **false**

- **isNaN()** - determines whether a value is an illegal number
  - `isNaN(1.23);` Returns: **false**
  - `isNaN(“hello”);` Returns: **true**

- **parseFloat()** - parses string and returns floating number
  - `parseFloat(“1.23”);` Returns: **1.23**

- **parseInt()** - Parses a string and returns an integer
  - `parseInt(“1234”);` Returns: **1234**
Number Object Functions

```javascript
var num = 123.456;

• toExponential(x) - converts number to exponential form. num is optional and specifies # of digits after decimal
  ```javascript
  num.toExponential();
  Returns: 1.23456e+2
  num.toExponential(1);
  Returns: 1.2e+2
  ```

• toFixed(x) - formats number with x digits after decimal
  ```javascript
  num.toFixed(2);
  Returns: 123.45
  ```

• toPrecision(x) - formats a number to x length
  ```javascript
  num.toPrecision(4);
  Returns: 123.4
  ```
```
Strings Object

• String objects are used to manipulate text.

• Example of how to create a string:

    var first_name = "John";

• length property:

    document.writeln(first_name.length);

    Output: 4

• Method toUpperCase():

    document.writeln(first_name.toUpperCase());

    Output: JOHN

• Method toLowerCase():

    document.writeln(first_name.toLowerCase());

    Output: john
Strings Object Continued

```javascript
var first_name = "John";

• charAt(index) returns character at the index:
  document.writeln(first_name.charAt(2));  // Output: h

• concat(str) returns copy of given string joined with str:
  document.writeln(first_name.concat("ny"));  // Output: Johnny

• replace(str1, str2) – replaces substring str1 with str2:
  document.write(first_name.replace("hn","e"));  // Output: Joe

• substr(start,len) – extracts substring of len characters long starting at index start
  document.write(first_name.substr(1,2));  // Output: oh
```
Other String Methods

- `charCodeAt()` Returns the Unicode of the character at the specified index
- `fromCharCode()` Converts Unicode values to characters
- `indexOf()` Returns position of first found occurrence of specified value in string
- `lastIndexOf()` Returns position of last found occurrence of specified value in string
- `match()` Searches for match between regular expression and string, and returns the matches
- `search()` Searches for a match between a regular expression and a string, and returns the position of the match
- `slice()` Extracts a part of a string and returns a new string
- `split()` Splits a string into an array of substrings
- `substring()` Extracts the characters from a string, between two specified indices
- `valueOf()` Returns the primitive value of a String object
String HTML Wrapper Methods

- HTML wrapper methods return the string wrapped inside an HTML tag. Examples:
  
  ```javascript
  var str = “blah”;
  ``

  - anchor(anchorName) creates an anchor:
    ```javascript
    str.anchor(“theAnchor”);
    ```
    
    Output: `<a name=”theAnchor”>blah</a>`

  - link(url) creates link:
    ```javascript
    str.link(“http://theLink.com”);
    ```
    
    Output: `<a href=”http://theLink.com”>blah</a>`

  - bold() creates a string embedded in `<b>` tag:
    ```javascript
    str.bold();
    ```
    
    Output: `<b>blah</b>`
Other HTML Wrapper Methods

- `big()` Displays a string using a big font
- `blink()` Displays a blinking string
- `fixed()` Displays a string using a fixed-pitch font
- `fontcolor()` Displays a string using a specified color
- `fontsize()` Displays a string using a specified size
- `italics()` Displays a string in italic
- `small()` Displays a string using a small font
- `strike()` Displays a string with a strikethrough
- `sub()` Displays a string as subscript text
- `sup()` Displays a string as superscript text
Debugging Your JavaScript Programs

- **Load-time error**
  - Occurs when script is first loaded by JavaScript interpreter
- **Run-time error**
  - Occurs after script has been successfully loaded and is being executed
- **Logical errors**
  - Free from syntax and structural mistakes, but result in incorrect results
Illustrating a Logical Error

**Staff Directory**

<table>
<thead>
<tr>
<th>Name</th>
<th>Phone</th>
<th>E-Mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Catherine Adler</td>
<td>555-3100</td>
<td>mpl.gov@cadler</td>
</tr>
<tr>
<td>Library Director</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Michael Li</td>
<td>555-3145</td>
<td>mpl.gov@mikeli</td>
</tr>
<tr>
<td>Head of Adult Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Kate Howard</td>
<td>555-4389</td>
<td>mpl.gov@khoward</td>
</tr>
<tr>
<td>Head of Technical Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Robert Hope</td>
<td>555-7811</td>
<td>mpl.gov@rhope</td>
</tr>
<tr>
<td>Head of Children's Services</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Wayne Lewis</td>
<td>555-9001</td>
<td>mpl.gov@wlewis</td>
</tr>
<tr>
<td>Circulation Services Supervisor</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bill Forth</td>
<td>555-9391</td>
<td>mpl.gov@bforth</td>
</tr>
<tr>
<td>Interlibrary Loan</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Usernames and e-mail server names are displayed in the wrong order.
Common Mistakes

• Common errors
  – Misspelling a variable name
  – Mismatched parentheses or braces
  – Mismatched quotes
  – Missing quotes

• Debugging
  – Process of searching code to locate a source of trouble
Debugging Tools and Techniques

• Writing modular code
  – Breaking up a program’s different tasks into smaller, more manageable chunks

• Alert dialog box
  – Dialog box generated by JavaScript that displays a text message with an OK button

• Microsoft Script Debugger
  – Can help to create and debug JavaScript programs
Microsoft Debugger Window
Tips for Writing Good JavaScript Code

• Apply layout techniques to make code more readable
• Use descriptive variable names
• Be consistent in how you apply uppercase and lowercase letters to your variable names
• Add comments to your code
• Create customized functions
• Break up long and complicated functions into smaller functions
• Use the debugging tools available
• In case of a logical error, use alert boxes