4. Client-Side Scripting

- Why Client-Side Coding?
- Introduction to Dynamic HTML (DHTML)
- Overview of XHTML Document Object Model (DOM)
- Overview of JavaScript
  - Syntax
  - Built-in Objects
  - User-defined Objects
- Manipulating DOM Objects using JavaScript

Why Client-Side Coding?

- What is client-side code?
  - Software that is downloaded from Web server to browser and then executes on the client
- Including code within a web page, leads to addition of a number of features to a Web page
  - Without the need to send information to the Web Server which takes time.
- Why client-side code?
  - Better scalability: less work done on server
  - Better performance/user experience
  - Create UI constructs not inherent in HTML (i.e., special formatting features that go beyond HTML)
    - Drop-down and pull-out menus
    - Tabbed dialogs
  - Cool effects, e.g. animation
  - Data validation

Introduction to Dynamic HTML

- Traditionally, Web page elements are static and they never change unless the Web page itself is changed
  - Appropriate for pages where the content and styling seldom change and where the visitor is merely a passive reader of page content.
  - Not appropriate for **dynamic** pages where layout, styling, and content need to change in response to visitor actions and desires.
- Examples of dynamic effects
  - A hyperlink changing its visual appearance to indicate user actions
  - A picture having its size or lightness changing, or be hidden or revealed, by user clicks on accompanying buttons.
  - A block of text changing (revealing words definitions) by moving the mouse on top of the underlined terms being defined
  - A Web page "programmed" to carry out processing tasks through interaction with the user.

Dynamic HTML (DHTML)

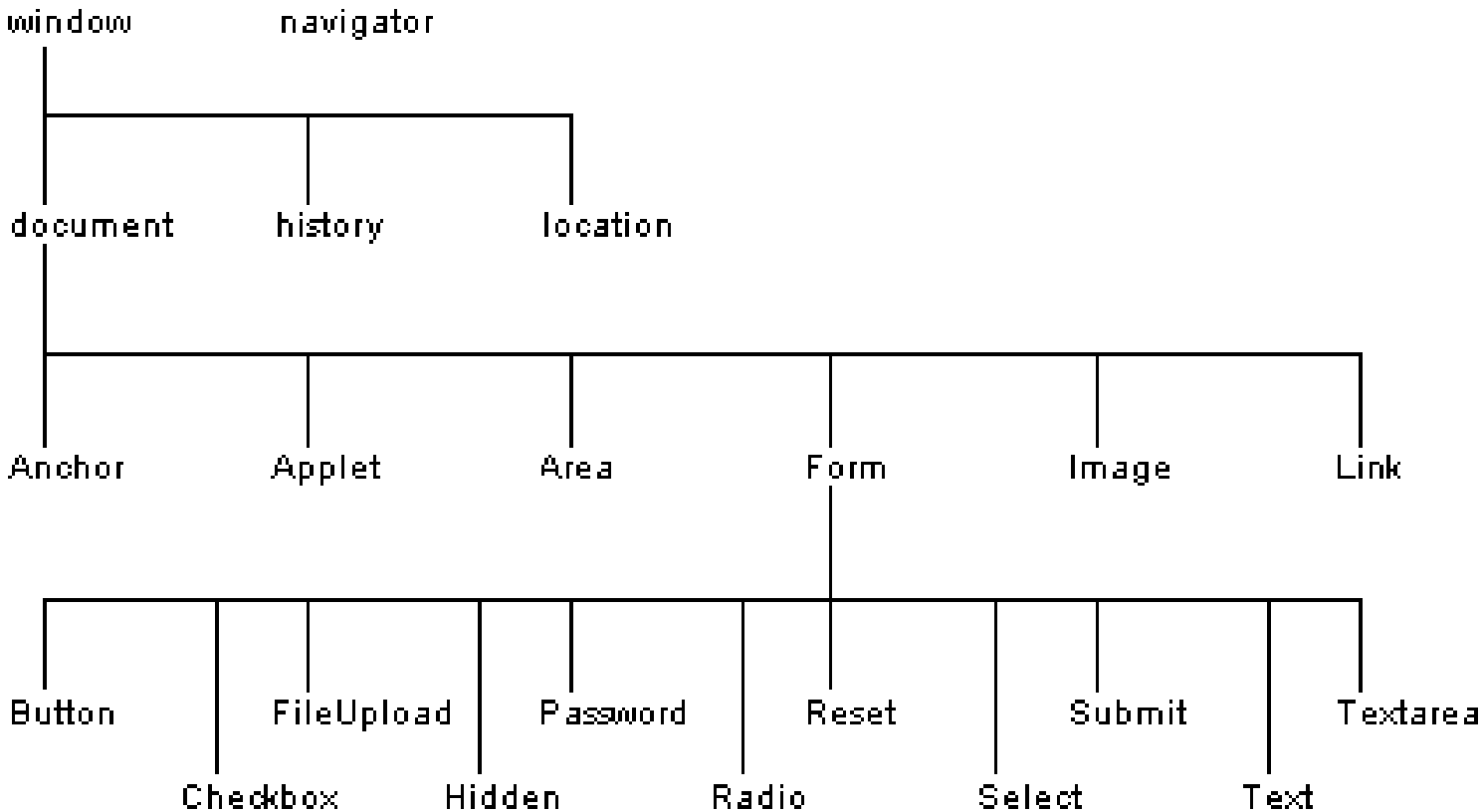
- A collection of techniques to change static Web pages into dynamic Web pages that react to events.
  - Events can be initiated by the user or by the Web page itself.
- DHTML pages requires familiarity with four main topics
  - XHTML
  - CSS
  - The browser's Document Object Model (DOM)
    - the collection of XHTML elements appearing on a Web page
  - JavaScript
- There are DOM standards (published by **W3C**) to provide common approaches to using DHTML.
  - Unfortunately, not all browsers follow these standards, and some augment the standards with additional capabilities.

The DOM

- A Web page is made dynamic by applying JavaScript processing to the XHTML elements on that page
  - XHTML tags are also **software objects** , having properties and methods that can be programmed
  - These objects are programmed through JavaScript processing routines to make Web pages dynamic
- The DOM is the programming interface to the XHTML objects appearing on a Web page
- All XHTML elements, along with their containing text and attributes, can be accessed through the DOM.
  - The contents can be modified or deleted, and new elements can be created.
- The XHTML DOM is platform and language independent.
  - It can be used by any programming language like Java, JavaScript, and VBScript.

The DOM Hierarchy

- The DOM is organized as a hierarchy of browser components



`Window` Object

- The window object is the “master” DOM object at the top of the DOM hierarchy
- Useful properties:
  - `length`: number of frames in window
  - `frames`: an array of window objects, one for each frame
  - `parent`: Since frames are window objects, sometimes parent window is needed
- Examples:
  - `window.document` : if frameless, accesses the top level document. If frames, accesses the top frame’s document
  - `window.frame[1].document` : Access the document contained in the first frame
  - `frame[1].parent.document` : Access the document contained in the parent frame

`Window` Object Methods

- `alert`, `confirm` and `prompt` are actually methods of the window object, ex: `window.alert`
- `window.open(); /* opens a window */`
- `window.close(); /* closes window */`

`Navigator` Object

- Contains information about the browser
- Can be accessed as window.navigator or just navigator
- Useful properties:
  - `appName`: name of browser used (can be deceiving; more on this in a later class)
  - `appVersion`: version of browser used (can be deceiving; more on this in a later class)
  - `platform`: operating system in use
  - `cookieEnabled`: can the browser store cookies?

`Location` Object

- Contains information about the current URL
- Can be accessed as window.location or just location
- Useful properties:
  - `href`: retrieves entire URL
  - `host`: retrieves just the domain name (ex: yahoo.com)
  - `pathname`: retrieves just the path inside the domain (page name is at end)
  - `hash`: retrieves the anchor

`History` Object

- Contains information on the URLs that the browser has visited in this session within a window
- Can be accessed as window.history or just history
- Useful properties: next, previous (tells you the URL, but won't direct you there)
- Useful methods:
  - `back`: same as pressing the back arrow button
  - `forward`: same as pressing the forward arrow button
  - `go`: go back or forward a given number of pages; to go back 3 pages:
    - `history.go(-3);`

`Document` Object

- This is the typically the most accessed object
- You can access all items in the document window through the document object
  - Forms, tables, paragraphs, lists, images, etc.
  - Consult a reference for properties and methods
- Frameless document: Access as `window.document or document`
- Document contained in a frame: `window.frame[x].document`, where x is the number or name of the frame

Identifying DOM Objects

`navigator`	The browser itself.
`window`	The main browser window.
`window.`*`framename`*	A frame that occupies the browser window and identified by its assigned name.
`window.document`	The document appearing in the main browser window.
`window.`*`framename`*`.document`	The document appearing in a frame identified by its assigned name.
`document.getElementById("`*`id`*`")`	An XHTML element appearing in a document and identified by its assigned `id` value.
`document.all.`*`id`*	Alternate reference to an XHTML element appearing in a document and identified by its assigned `id` value.
`id`	Alternate reference to an XHTML element appearing in a document and identified by its assigned `id` value.

Referencing XHTML Elements

- An XHTML element must be assigned an **id** for a script to refer to it:

  <*tag* **id**="*idValue*"...>

  - The assigned *idValue* value must be unique and composed of alphanumerics excluding spaces

- Once an id is assigned, the XHTML object can be referenced in a script:

  document.getElementById("*idValue*")

- An alternate is the notation

  document.all.*idValue*,

- In some cases only the value itself is needed

  idValue

Getting and Setting Style Properties

- DHTML is created commonly by changing the style properties of XHTML elements

  -- Get a current style property:
  document.getElementById("*id*").**style**.***property***

  -- Set a different style property:
  document.getElementById("*id*").**style**.***property*** = *value*

- For example, given

  <h2 id="Head" style="color:blue">This is a Heading</h2>

- We can change the color property as

  document.getElementById("Head").style.color = "red"

Applying Methods

- DHTML can also be created by by activating methods built into the objects. E.g., Given

  Enter your name: <input id="Box" type="text"/>

- We automatically make the textbox gain focus by:

  document.getElementById("Box").focus()

- Gaining focus means if the page containing the above code first loads, the cursor is automatically placed in the textbox
  - The user can immediately begin typing without having to click first.

- Learning DHTML is to do with learning the numerous properties and methods available for the numerous DOM components

The JavaScript Language

- A **client-side** scripting language – i.e., the ability to run JavaScript code is built into modern desktop browsers.
  - Code embedded in Web pages along with XHTML and CSS formatting codes.
  - The browser interprets and runs scripts locally, on the PC
- JavaScript is not Java, or even related to it
  - The original name for JavaScript was “LiveScript”
  - The name was changed when Java became popular
  - Released in the Fall of 1995
- JavaScript and Java are unrelated except for minor syntactical similarities.

JavaScript versus Java

JavaScript	Java
Interpreted (not compiled) by client.	Compiled bytecodes downloaded from server, executed on client.
Object-oriented. No distinction between types of objects. Inheritance is through the prototype mechanism, and properties and methods can be added to any object dynamically.	Class-based. Objects are divided into classes and instances with all inheritance through the class hierarchy. Classes and instances cannot have properties or methods added dynamically.
Code integrated with, and embedded in, HTML.	Applets distinct from HTML (accessed from HTML pages).
Variable data types not declared (dynamic typing).	Variable data types must be declared (static typing).
Cannot automatically write to hard disk.	Cannot automatically write to hard disk.

Placement of JavaScripts

- JavaScript can be put in the `<head>` or in the `<body>` of an HTML document
  - JavaScript *functions* should be defined in the `<head>`
    - This ensures that the function is loaded before it is needed
  - JavaScript in the `<body>` will be executed as the page loads
- JavaScript can be put in a separate `.js` file
  - `<script src="myJavaScriptFile.js"></script>`
  - Put this HTML wherever you would put the actual JavaScript code
  - An external .js file lets you use the same JavaScript on multiple HTML pages
  - The external `.js` file cannot itself contain a `<script>` tag
- JavaScript can be put in HTML form object, such as a button
  - This JavaScript will be executed when the form object is used

JavaScript Functions

- A JavaScript function works just like subprograms in other languages:

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

function ChangeStyle() {
   document.getElementById("MyTag").style.fontSize = "14pt";
   document.getElementById("MyTag").style.fontWeight = "bold";
   document.getElementById("MyTag").style.color = "red";

 }

</script>


<p id="MyTag" onclick="ChangeStyle()" >This is a paragraph that
   has its styling changed.</p>
```

- The semicolon ending a line is optional unless two or more statements appear on the same line.

Mouse Event Handlers

- There are numerous page events and associated event handlers that need to be learned to create DHTML

`onclick`	The mouse button is clicked and released with the cursor positioned over a page element.
`ondblclick`	The mouse button is double-clicked with the cursor positioned over a page element.
`onmousedown`	The mouse button is pressed down with the cursor positioned over a page element.
`onmousemove`	The mouse cursor is moved across the screen.
`onmouseout`	The mouse cursor is moved off a page element.
`onmouseover`	The mouse cursor is moved on top of a page element.
`onmouseup`	The mouse button is released with the cursor positioned over a page element.

Inline Scripts

- A second way is to code a script inside the event handler itself:

```
<p id="MyTag" onclick="document.getElementById('MyTag').style.fontSize='14pt';
    document.getElementById('MyTag').style.fontWeight='bold';
    document.getElementById('MyTag').style.color='red'"> This is a paragraph that
    has its color changed.</p>
```

- Note
  - The <script> tag is not necessary in this case
  - Quoted values inside the script must be enclosed in single quotes (apostrophes) to alternate and differentiate the sets of quote marks.
  - Amount of use and convenience dictate whether to use functions or inlining
  - The paragraph “MyTag” (containing the script) refers to itself in the script

The `this` Keyword

- The preceding code can be simplified thus:

```
<p id="MyTag" onclick="this.style.fontSize='14pt';
    this.style.fontWeight='bold'; this.style.color='red'"> This is a paragraph
    that has its color changed.</p>
```

- Self reference can also be passed to a function:

```
<script type="text/javascript">
function ChangeStyle(SomeTag) {
   SomeTag.style.fontSize = "14pt";
   SomeTag.style.fontWeight = "bold";
   SomeTag.style.color = "red";
}
</script>
<p onclick="ChangeStyle(this)">Style this paragraph in 14pt bold red
    text.</p>
<p onclick="ChangeStyle(this)">Style this paragraph in the same
    way.</p>
```

JavaScript Comments

- JavaScript uses C-style comments: // and /* */
- Some old browsers do not recognize script tags
  - These browsers will ignore the script tags but will display the included JavaScript
  - To get old browsers to ignore the whole thing, use:

```
<script type="text/javascript">
  <!--
      document.write("Hello World!")
  //-->
  </script>
```

  - The `<!--` introduces an HTML comment
  - To get JavaScript to ignore the HTML close comment, `-->`, the `//` starts a JavaScript comment, which extends to the end of the line

Primitive data types

- JavaScript has three "primitive" types: `number`, `string`, and `boolean`
  - Everything else is an object
- Numbers are always stored as floating-point values
  - Hexadecimal numbers begin with `0x`
  - Some platforms treat `0123` as octal, others treat it as decimal
- Strings may be enclosed in single quotes or double quotes
  - Strings can contains `\n` (newline), `\"` (double quote), etc.
- Booleans are either `true` or `false`
  - `0`, `"0"`, empty strings, `undefined`, `null`, and `NaN` are `false`, other values are `true`

Variables

- Variables are declared with a var statement:
  - `var pi = 3.1416, x, y, name = "Dr. ABC" ;`
  - Variables names must begin with a letter or underscore
  - Variable names are case-sensitive
  - Variables are *untyped* (they can hold values of any type)
  - The word `var` is optional (but it's good style to use it)
- Variables declared within a function are local to that function (accessible only within that function)
- Variables declared outside a function are global (accessible from anywhere on the page)

Operators, I

- Because most JavaScript syntax is borrowed from C (and is therefore just like Java), we won't spend much time on it
- Arithmetic operators:
  `+` `-` `*` `/` `%` `++` `--`
- Comparison operators:
  `<` `<=` `==` `!=` `>=` `>`
- Logical operators:
  `&&` `||` `!` (`&&` and `||` are short-circuit operators)
- Bitwise operators:
  `&` `|` `^` (XOR) `~` (NOT) `<<` `>>` (Shifts binary bits to right, discarding bits shifted off) `>>>` (Shifts binary bits to right, discarding bits shifted off and shifting in zeros from left.)
- Assignment operators:
  `+=` `-=` `*=` `/=` `%=` `<<=` `>>=` `>>>=`
  `&=` `^=` `|=`

Operators, II

- String operator:
  `+`
- The conditional operator:
  `condition ? value_if_true : value_if_false`
- Special equality tests:
  - `==` and `!=` try to convert their operands to the same type before performing the test
  - `===` and `!==` consider their operands unequal if they are of different types

    Using x=3 and y="3": 1) x==y Result: returns true
    2) x===y Result: returns false
- Additional operators:
  `new` `typeof` `void` `delete`

Statements, I

- Most JavaScript statements are also borrowed from C
  - Assignment: `greeting = "Hello, " + name;`
  - Compound statement:

    `{ statement; ...; statement }`
  - If statements:

    `if (condition) statement;`

    `if (condition) statement; else statement;`
  - Familiar loop statements:

    `while (condition) statement;`

    `do statement while (condition);`

    `for (initialization; condition; increment) statement;`

Statements, II

- The switch statement:

```
switch (expression){
      case label :
         statement;
         break;
      case label :
         statement;
         break;
      ...
      default : statement;
}
```

- Other familiar statements:
  - `break;`
  - `continue;`
  - The empty statement, as in `;;` or `{ }`

Exception handling, I

- Exception handling in JavaScript is almost the same as in Java
- `throw` *`expression`* creates and throws an exception
  - The *`expression`* is the value of the exception, and can be of any type (often, it's a literal String)
- ```
  try {
      statements to try
  } catch (e) {    // Notice: no type declaration for e
      exception-handling statements
  } finally {      // optional, as usual
      code that is always executed
  }
  ```
  - With this form, there is only one catch clause
  ```

Exception handling, II

- ```
  try {
      statements to try
  } catch (e if test1) {
      exception-handling for the case that test1 is true
  } catch (e if test2) {
      exception-handling for when test1 is false and test2 is true
  } catch (e) {
      exception-handling for when both test1 and test2 are false
  } finally {          // optional, as usual
      code that is always executed
  }
  ```

- Typically, the test would be something like

  ```
  e == "InvalidNameException"
  ```
  ```

Basic Input and Output

- Programming languages need to start with some data and manipulate it
- `Confirm` asks a yes or no question in a dialog box
- `Prompt` prompts the user to type in some information into a text field inside the dialog box
- Sources of data can include:
  - Files
  - Databases
  - User (keyboard & mouse typically)
  - Variable assignments (ex: pi=3.14159)
  - Javascript objects
    - Example: date object
- Example:
  - `User_name = prompt("What is your name?", "Enter your name here");`

Output

- After a program manipulates the input data with various statements it usually creates an output of some kind

- Source of output may include:
  - Files
  - Database
  - Display or Printer
  - Devices (sound card, modems etc)
  - Javascript Objects
    - Via Object Methods

Simple Input/Output - 1

```
<script type="text/javascript">
function Multiply() {
  var No1 = prompt("Enter the first number:", "");
  var No2 = prompt("Enter the second number:", "");
  var Product = No1 * No2;

  Str = No1 +" * "+ No2 + " = " ;
  alert(Str + Product.toFixed(2));
}
</script>
<input type="button" value="Get Number"
   onclick="Multiply()"/>
```

Simple Input/Output - 2

```
<script type="text/javascript">
 function Subtract() {
   document.getElementById("Output").value =
    document.getElementById("FirstNo").value -
    document.getElementById("SecondNo").value;
 }
</script>
<input id="FirstNo" type="text" value="10" style="width:50px"/>
<input id="SecondNo" type="text" value="20" style="width:50px"/>
<input type="button" value=" = " onclick="Subtract()"/>
<input id="Output" type="text" style="width:50px"/>
```

Simple Input/Output - 3

```
<script type="text/javascript">
  function TextSize() {
    var ReturnedValue = window.confirm("Larger text?");
    if (ReturnedValue == true) {
         document.body.style.fontSize = "12pt";
    } else {
         document.body.style.fontSize = "10pt";
    }
  }
</script>
<input type="button" value="Set Text" onclick="TextSize()">
```

Some Built-in DOM Objects

- The DOM includes built-in objects besides those associated with specific elements of a Web page.
  - Number
  - Boolean
  - Math
  - String
  - Date
  - Array

- http://msconline.maconstate.edu/tutorials/JSDHTML/default.htm

Numbers

- In JavaScript, all numbers are floating point
- Special predefined numbers:
  - `Infinity, Number.POSITIVE_INFINITY`
    - the result of dividing a positive number by zero
  - `Number.NEGATIVE_INFINITY`
    - the result of dividing a negative number by zero
  - `NaN, Number.NaN` (Not a Number)
    - the result of dividing 0/0
    - `NaN` is unequal to everything, even itself
    - There is a global `isNaN()` function
  - `Number.MAX_VALUE`
    - the largest representable number
  - `Number.MIN_VALUE`
    - the smallest (closest to zero) representable number

Boolean

- The boolean values are `true` and `false`
- When converted to a boolean, the following values are also `false`:
  - `0`
  - `"0"` and `'0'`
  - the empty string, `''` or `""`
  - `undefined`
  - `null`
  - `NaN`

`Math` Object

- Can be accessed as Math.property, ex:
  - `x=Math.pow(3,3); // x=27`
- Allows many common mathematical calculations including (all prefixed with Math as above):
  - `abs(x)` : absolute value
  - `ceil(x)` and `floor(x)` : smallest integer not less than x and largest integer not greater than x
  - `cos(x)`, `exp(x)`, `log(x)`, `sin(x)`, `tan(x)` : trigonometric and log rhythmic functions
  - `min(x,y)` or `max(x,y)` : returns the minimum or maximum of values x and y
  - `pow(x,y)` : raises x to the power y
  - `round(x)` : rounds to nearest integer
  - `sqrt(x)` : Square root

Strings and characters

- In JavaScript, string is a primitive type
- Strings are surrounded by either single quotes or double quotes
- There is no “character” type
- Special characters are:

`\0`	NUL	`\v`	vertical tab
`\b`	backspace	`\'`	single quote
`\f`	form feed	`\"`	double quote
`\n`	newline	`\\`	backslash
`\r`	carriage return	`\xDD`	Unicode hex *DD*
`\t`	horizontal tab	`\xDDDD`	Unicode hex *DDDD*

Some string methods

Method	Description
`bold()`	Changes the text in a string to bold.
`italics()`	Changes the text in a string to italic.
`strike()`	Changes the text in a string to strike-through characters.
`sub()`	Changes the text in a string to subscript.
`sup()`	Changes the text in a string to superscript.
`toLowerCase()`	Changes the text in a string to lower-case.
`toUpperCase()`	Changes the text in a string to upper-case.
`fixed()`	Changes the text in a string to fixed (monospace) font.
`fontcolor ("color")`	Changes the color of a string using color names or hexadecimal values.
`fontsize("n")`	Changes the size of a string using font sizes 1 (smallest) - 7 (largest).
`link("href")`	Formats a string as a link.

More string methods

Method	Description
`charAt(`*`index`*`)`	Returns the character at position *index* in the string.
`charCodeAt(`*`index`*`)`	Returns the Unicode or ASCII decimal value of the character at position *index* in the string.
`indexOf("`*`chars`*`")`	Returns the starting position of substring "*chars*" in the string. If "*chars*" does not appear in the string, then -1 is returned.
`lastIndexOf("`*`chars`*`")`	Returns the starting position of substring "*char*" in the string, counting from end of string. If "*chars*" does not appear in the string, then -1 is returned.
`slice(`*`index1`*`[,`*`index2`*`])`	Returns a substring starting at position *index1* and ending at (but not including) position *index2*. If *index2* is not supplied, the remainder of the string is returned.
`split(`*`delimiter`*`)`	Splits a string into separate substrings which are copied as individual elements into a new array object. The *delimiter* identifies the separator character for splitting the string but it is not included in the substrings. The array object does not need to be prior declared.
`substr(`*`index`*`[,`*`length`*`])`	Returns a substring starting at position *index* and including *length* characters. If no length is given, the remaining characters in the string are returned.
`substring(`*`index1`*`,`*`index2`*`)`	Returns a substring starting at position *index1* and ending at (but not including) position *index2*.
`toString()`	Converts a value to a string.
`toFixed(`*`n`*`)`	Returns a string containing a number formatted to *n* decimal digits.
`toPrecision(`*`n`*`)`	Returns a string containing a number formatted to *n* total digits.

`Date` Object

- Permits you to work with date and time settings taken from the system clock of the user's PC.
- By default creates an object with the computer's current date and time, ex:
  - `now = new Date(); // variable now contains current date and time`
  - Note: months are expressed 0-11, 0 being January, 11 being December
- Dates are actually stored as an integer representing the number of milliseconds since January 1st, 1970
  - Negative values indicate dates before this date
- Once you have a date object you can set the date, or read the date in a number of useful formats
  - `now.setFullYear(2003, 0, 31); /* Jan 31st, 2003 */`
  - `now.setHours(13, 13, 13); /* 1:13:13 PM, local time zone */`

`Date` Methods

Method	Description
`getDate()`	Returns the day of the month.
`getDay()`	Returns the numeric day of the week (Sunday = 0).
`getMonth()`	Returns the numeric month of the year (January = 0).
`getYear()` `getFullYear()`	Returns the current year.
`getTime()`	Returns the number of milliseconds since January 1, 1970.
`getHours()`	Returns the military hour of the day.
`getMinutes()`	Returns the minute of the hour.
`getSeconds()`	Returns the seconds of the minute.
`getMilliseconds()`	Returns the milliseconds of the second.

Building an XHTML Table with JavaScript

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

function BuildArray()
{
  var SquareRoots = new Array();
  for (i=0; i<10; i++) {
    SquareRoots[i] = Math.sqrt(i + 1).toFixed(3);
  }

  var TableOut = ""
  TableOut += "<table border='1' style='width:100px'>";
  TableOut += "<caption>SquareRoots</caption>";
  for (i=0; i < SquareRoots.length; i++) {
    TableOut += "<tr><td style='text-align:center'>" +
      SquareRoots[i] + "</td></tr>";
  }
  TableOut += "</table>";
  document.getElementById("Output").innerHTML = TableOut;
}

</script>

<input type="button" value="Build Array" onclick="BuildArray()"/>
<p><div id="Output"></div></p>
```

Searching An Array of Objects

```
<script type="text/javascript">
var Contacts = new Array();
function LoadArray(){
  Contacts[0] = "M. Al-Turki, 555-1111, mturki@kfupm.edu.sa";
 Contacts[1] = "I. Katebah, 555-2222, ikatebah@kfupm.edu.sa";
 Contacts[2] = "U. Jan, 555-3333, ujan@kfupm.edu.sa";
 Contacts[3] = "L. Al-Awami, 555-4444, lawami@kfupm.edu.sa";
 Contacts[4] = "H. Al-Ramadi, 555-5555, hramadi@kfupm.edu.sa";
 Contacts[5] = "H. Al-Helal, 555-6666, hhelal@kfupm.edu.sa";
 Contacts[6] = "S. Al-Awfi, 555-7777, sawfi@kfupm.edu.sa";
}
function Find(){
LoadArray();
 document.getElementById("Output").innerText = "";

 var SearchName = document.getElementById("FindString").value.toLowerCase();
 for (i=0; i < Contacts.length; i++) {
   var ArrayName = Contacts[i].toLowerCase();
   if (ArrayName.indexOf(SearchName) != -1) {
     var InfoArray = Contacts[i].split(",");
     document.getElementById("Output").innerHTML += "<br/>" +
       "<b>" + InfoArray[0] + "</b><br/>" +
       InfoArray[1] + "<br/>" +
       "<a href='mailto:'" + InfoArray[2] + "'>" + InfoArray[2] + "</a><br/>";
   }  }}
</script>

Find: <input id="FindString" type="text" style="width:100px" value="gold"/>
<input type="button" value="Find" onclick="Find()"/><br/>
<span id="Output"></span>
```

User-Defined Objects

- You can create complex data structures by creating your own objects
  - JavaScript allows you, although it is not a full-featured OO language
- As usual, objects are created with a **constructor** function

```
function Employee(IDValue, NameValue, PayRateValue) {
    this.ID = IDValue;
    this.Name = NameValue;
    this.PayRate = PayRateValue.toFixed(2);
    this.Hours = 0;
    this.Pay = 0;
}
```

- JavaScript's constructors are like its other functions
- Such a method can be viewed as a *class* definition, providing the model for creating objects

Creating an Object Array

```
<script type="text/javascript">
var EmployeeDB = new Array();
function Employee(IDValue, NameValue, PayRateValue){
  this.ID = IDValue;
  this.Name = NameValue;
  this.PayRate = PayRateValue.toFixed(2);
  this.Hours = 0;
  this.Pay = 0;
}
function AddEmployees(){
  EmployeeDB[EmployeeDB.length] = new Employee("11111", "A. Katebah", 10.00);
  EmployeeDB[EmployeeDB.length] = new Employee("22222", “H. Al-Helal", 15.00);
  EmployeeDB[EmployeeDB.length] = new Employee("33333", “M. Araman", 20.00);
  EmployeeDB[EmployeeDB.length] = new Employee("44444", “F. Nabulsi", 25.00);
  EmployeeDB[EmployeeDB.length] = new Employee("55555", “Y. Al-Amer", 30.00);
}
</script>
```

Adding Methods to Your Objects

- Suppose you defined methods `ShowRecord()` and `ComputePay()` (with or without parameters)

- You add them to your object as follows

```
function Employee(IDValue, NameValue, PayRateValue){
    this.ID = IDValue;
    this.Name = NameValue;
    this.PayRate = PayRateValue.toFixed(2);
    this.Hours = 0;
   this.Pay = 0;
   this.ShowRecord = ShowRecord;
   this.ComputePay = ComputePay;
}
```

Using Your Object's Methods

```
function ShowRecord() {
    // code not shown
    return s;
}
function ComputePay(hours) {
    this.Hours = document.getElementById(hours).value;
    this.Pay = this.PayRate * this.Hours;
    this.Pay = this.Pay.toFixed(2);
}
function ShowEmployees() {
  var OutString = ""
  // code not shown
  for (i=0; i < EmployeeDB.length; i++) {
     OutString += EmployeeDB[i].ShowRecord();
  }
 // code not shown
}
function EnterHours() {
  for (i=0; i<EmployeeDB.length; i++) {
    if (EmployeeDB[i].ID == document.getElementById("EmployeeID").value) {
      EmployeeDB[i].ComputePay("EmployeeHours");
      break;
    }
  }
// code not shown
}
```

Sorting Array Elements

- To sort an array alphabetically:
  - `myArray.sort()`
- To sort an array sorts numerically:
  - `myArray.sort(function(a, b) { return a - b; })`

```
function SortDESC(){
  EmployeeDB.sort(function(a,b){return b.ID - a.ID});
  ShowEmployees();
}
function SortASC(){
  EmployeeDB.sort(function(a,b){return a.ID - b.ID});
  ShowEmployees();
}
```

The Window Object

- The **navigator** object at the top of the DOM hierarchy represents the browser.
  - has properties used to get information about the browser, version, OS platform etc
- An instance of a **window** object is created when a browser is launched
  - Its properties become available for inspection/use as: **window**.*property* or **self**.*property* or only *property*
- Common window properties:

Property	Description and Setting
`defaultStatus`	Sets the text to display in the status bar appearing at the bottom of the browser window. `top.defaultStatus=` (No message set; default is blank or "Done" when refreshing a page.)
`length`	Gets the number of frames in the window `top.length = 2`
`location`	Gets or sets the URL of the document in the window. `top.location=ttp://msconline.maconstate.edu/tutorials/JSDHTML/default.htm`
`screenLeft` `screenTop`	Gets or sets the pixel position of the window relative to the top-left corner of the screen. `top.screenLeft=26` `top.screenTop = 120`

The `location` Window Property

- The **location** property is extremely useful in setting up scripted links.
  - Has the same effect as the <a> tag but with scripted control.
  - Can create links using it along with other DHTML settings (see example below)

```
<b>A scripted link:</b>

<span style="color:blue; text-decoration:underline; cursor:hand"

  onmouseover="this.style.fontWeight='bold'"

  onmouseout="this.style.fontWeight='normal'"

  onmousedown="this.style.color='red'"

  onmouseup="this.style.color='blue'"

  onclick="location='http://webcourses.kfupm.edu.sa'">

    Link to WebCT (KFUPM's)

</span>
```

The Window Object's Methods

- Like other objects, the Window object has methods, too, that become available when a new window is opened
  - Example: `alert()`, `prompt()`, `focus()`, `open()`, `close()`, `resizeTo(h,v)`, `print()`, `createPopup()`, etc
- Window timers:
  - two sets of windows methods relate to setting up timing devices to control the automatic display of pages.
    - one set introduces a delay before showing a page;
    - the other set defines a continuous interval during which activities are repeated.
- Delay timer
  - established with `setTimout()` and cleared with `clearTimout()` methods.
  - `setTimeout()` causes the script to pause for a specified number of milliseconds.
    - **setTimeout**("*statement*", *milliseconds*)
- Interval timer
  - established with `setInterval()` and cleared with `clearInterval()` methods.

Delay Timer Example

```
<script type="text/javascript">
var SlideWindow;
function SlideShow(){
  SlideWindow = open(“slide1.jpg", "", "width=300,height=200");
  SlideWindow.moveTo(400,400);
  setTimeout("SlideWindow.location=‘slide2.jpg'", 2000);
  setTimeout("SlideWindow.location=‘slide3.jpg'", 4000);
  setTimeout("SlideWindow.location=‘slide4.jpg'", 6000);
  setTimeout("SlideWindow.location=‘slide5.jpg'", 8000);
  setTimeout("SlideWindow.location=‘slide6.jpg'", 10000);
  setTimeout("SlideWindow.close()", 12000);
}
</script>
<input type="button" value="Slide Show" onclick="SlideShow()"/>
```

Interval Timer Example

```
<script type="text/javascript">
var SlideCount;
var SlideWindow;
function SlideShow(){
  SlideCount = 1;
  SlideWindow = open("Slide1.jpg", "", "width=300,height=200");
  SlideWindow.moveTo(400,400);
  SlideTimer = setInterval("ShowNextSlide()",2000);
}
function ShowNextSlide(){
  SlideCount ++;
  if (SlideCount <= 5 && SlideWindow.closed != true) {
    SlideWindow.location = "Slide" + SlideCount + ".jpg";
  }
  else {
    clearInterval(SlideTimer);
  SlideWindow.close();
 }
}
</script>
<input type="button" value="Show Slides" onclick="SlideShow()"/>
```

The Form Object

- **Forms** are devices for collecting information from users and submitting it for processing.
  - Used to interact with a Web page and through which server and browser scripts respond to user needs.
  - Web forms contain various types of **controls** like textbox, button, text area, drop-down list etc
- Form controls, as a group, often are enclosed inside **<form>** tags.
  - <form> tags can contain **action** and **method** attributes governing submission of form values for processing by server scripts.
  - <form> tags are *not* required when form controls are used for input to *browser* scripts.
- Example:

```
<input type="text" value="Change this text."
 onchange="document.getElementById("MSG").innerText=
      'You changed the text.'"/>
<span id="MSG"></span>
```

Validating Form Data

- JavaScript can be used to validate input data in XHTML forms before sending off the content to a server.

- Form data that typically are checked by a JavaScript could be:
  - has the user left required fields empty?
  - has the user entered a valid e-mail address?
  - has the user entered a valid date?
  - has the user entered text in a numeric field?

Example: Validating Empty Field

```
<script type="text/javascript">
<!--
function CheckNull(){
  document.getElementById("MSG").innerHTML = "";
  if (document.getElementById("MyField").value == "") {
    document.getElementById("MSG").innerHTML = "Missing data!";
    document.getElementById("MyField").focus();
  }
  else {
     document.getElementById("MyField").focus();
  }
}
//-->
</script>
</head>
<body>
<input type="Text" id="MyField"/>
<input type="button" value="Submit" onclick="CheckNull()"/>
<body onload="document.getElementById('MyField').focus()" />
<span id="MSG"></span>
```

Regular Expressions

- A notational convention used to match a word, a number or any other string of text within another
- Introduced in JavaScript 1.2
  - Mainly used in form validation
- A regular expression can be written statically or dynamically
  - Within slashes, such as `re = /ab+c/`
  - With a constructor, such as `re = new RegExp("ab+c")`
    - `Used when pattern to match is taken as user input, for example`
- Regular expressions are almost the same as in Perl or Java (only a few unusual features are missing)
- Examples
  - var pattern = /[0-9]/
    - Matches an integer
  - var pattern = /[A-Za-z]/
    - Matches a string of letters
- Can specify more complex regular expressions to match phone numbers of the form `abc-def-ghij` where `a,b,…,j` are digits

Special Characters in Regular Expressions

Token	Description
^	Match at the start of the input string
$	Match at the end of the input string
*	Match 0 or more times
+	Match one or more times
?	Match 0 or 1 time
a|b	Match a or b
{n}	Match the string n times
\d	Match a digit
\D	Match a non-digit

… Special Characters in Regular Expressions

Token	Description
\w	Match any alphanumeric character or underscore
\W	Match anything except alphanumeric characters or underscores
\s	Match a whitespace character
\S	Match anything except for whitespace characters
[…]	Creates a set of characters, one of which must match if the operation is to be successful. If you need to specify a range of characters then separate the first and the last with a hyper: [0-9] or [P-X]
[^…]	Creates a set of characters which must not match. If any character in the set matches then the operation has failed. This fails if any lowercase letter d to q is matched: [^d-q]

Example: Validating ID and Password

```
<script language="JavaScript">
<!--
function is6DigitInt(elm) {
    if (elm.value == "") {
        return false;
    }
    for (var i = 0; i < elm.value.length; i++) {
        if (elm.value.charAt(i) < "0" || elm.value.charAt(i) >
   "9") {
            return false;
        }
    }
    return true;
}
function is6DigitInt2(elm){
    var pattern = /[^0-9]/;
    if(pattern.test(elm.value))
         return false;
    else return true;
}
// continued …
```

…Example: Validating ID and Password

```
function is8CharacterString(elm){
    var pattern = /[A-Za-z0-9]/;
    if(pattern.test(elm.value))
         return true;
    else return false;
}
function isReady(iField, pField) {
    var iResult = iField.value.length != 6?false:is6DigitInt(iField);
    var pResult = pField.value.length !=
    8?false:is8CharacterString(pField);
    if (iResult == false) {
        alert("Please enter 6-digit integer.");
        iField.focus();
        return false;
    }
    if (pResult == false) {
        alert("Please enter 8-character string.");
        pField.focus();
        return false;
    }
    return true;
}
//-->
</script>
```

Example: Validation Phone and E-Mail

```
<script language="JavaScript">
<!--
function isValidKFUPMPhone(elm){
    var pattern = /0\d-?\d{3}-?\d{4}/;
    if(pattern.test(elm.value))
        return true;
    else return false;
}
function isValidEmail(elm){
    var pattern = /\w+\@\w+(.\w+)+/
    if(pattern.test(elm.value))
        return true;
    else return false;
}
// continued …
```

… Example: Validation Phone and E-Mail

```
function isReady(pField, eField) {
    var pResult = isValidKFUPMPhone(pField);
    var eResult = isValidEmail(eField);
    // display an alert when either of the above is false
    return true;
}
//-->
</script>
</head>
<body>
<h1>Validating Phone and E-mail:</h1>
<form name="idValid" onSubmit="return
   isReady(this.phone,this.email);" method="post" action="">
Phone : <input type="text" id="phone" /> <br />
E-Mail: <input type="text" id="email" /> <br />
<input type="submit" value="Submit" />
<input type="reset" value="Reset" />
</form>
</body>
</html>
```

Example: Sliding Tabs

```
<script language="JavaScript">
// Function to slide the tab into the visible portion of
   the browser window
function showLayer() {
    var hiddenLayer = document.getElementById("TabLayer");
    var layerPosition = parseInt(hiddenLayer.style.left);
    if (layerPosition < 0) {
        hiddenLayer.style.left = (layerPosition + 5) +
   "px";
        setTimeout("showLayer()", 20);
    }
}

// Function to hide the tab again
function hideLayer() {
    var hiddenLayer = document.getElementById("TabLayer");
    hiddenLayer.style.left = "-75px";
}
</script>
```

… Example: Sliding Tabs

```
<<body>
<div id="TabLayer"
     style="position:absolute; left:-75px; top:50px;
            width:115px; height:200px; z-index:1;
            background-color: #CCCCCC; layer-background-
   color: #CCCCCC;">
  <p align="right" class="hideshow">
    <a href="javascript:hideLayer();"
   class="hideshow">&lt;&lt;hide</a> |
    <a href="javascript:showLayer();"
   class="hideshow">show&gt;&gt;</a>
  </p>
  <p align="left" style="margin-left: 5px;">
    <a href="#">Quizzes</a><br>
    <a href="#">Majors</a><br>
    <a href="#">Project</a><br>
    <a href="#">Final</a>
  </p>
</div>
</body>
```

Debugging

- If you mess up on the syntax you will get a Javascript Error
  - Netscape
    - You will see a notification of an error on the status bar in the bottom left corner
    - You type “javascript:” in the URL field to pinpoint the error
  - Internet Explorer
    - By default a tiny little Javascript error message appears at the bottom left corner of the browser in yellow. Usually you won’t see it.
    - Can be explicitly disabled under Tools/Internet Options
    - Recommend under Tools/Internet Options/Advanced/Browsing to uncheck “Disable Script Debugging” and to check “Display a Notification about every script error” while doing development

Fixing Javascript Errors

- If possible use the debugging tool to locate the line containing the error
- Errors can be hard to find and fix
  - “code a little, test a little” strategy
- Often errors are due to things that are easy to overlook, like not closing a quote

References

- http://devedge-temp.mozilla.org/library/manuals/2000/javascript/1.3/guide/intro.html

- http://msconline.maconstate.edu/tutorials/JSDHTML/default.htm

- http://www.javascript.com