22AIE305: CLOUD COMPUTING

SCRIPTING LANGAUGES

- Programming languages developed to serve a particular purpose.
- JAVASCRIPT was developed to add interactivity to Web pages
- Interpreted rather than compiled
- Interpreter is built into the web browsers
- First made available as part of Netscape 2.0

What's a Scripting Language?

- Language used to write programs that compute inputs to another language processor
  - One language embedded in another
    - Embedded JavaScript computes HTML input to the browser
    - Shell scripts compute commands executed by the shell
- Common characteristics of scripting languages
  - String processing – since commands often strings
  - Simple program structure, define things "on the fly"
  - Flexibility preferred over efficiency, safety
    - Is lack of safety a good thing? (Example: JavaScript used for Web applications…)

Why JavaScript?

- "Active" web pages
- Web 2.0
  - AJAX, huge number of Web-based applications
- Some interesting and unusual features
  - First-class functions - interesting
  - Objects without classes - slightly unusual
  - Powerful modification capabilities - very unusual
    - Add new method to object, redefine prototype, …
- Many security and correctness issues
- "The world's most misunderstood prog. language"

JavaScript History

- Developed by Brendan Eich at Netscape
  - Scripting language for Navigator 2
- Later standardized for browser compatibility
  - ECMAScript Edition 3 (aka JavaScript 1.5)
- Related to Java in name only
  - "JavaScript is to Java as carpet is to car"
  - Name was part of a marketing deal
- Various implementations available
  - SpiderMonkey C implementation (from Mozilla)
  - Rhino Java implementation (also from Mozilla)

Motivation for JavaScript

- Netscape, 1995
  - > 90% browser market share
    - "I hacked the JS prototype in ~1 week in May and it showed! Mistakes were frozen early. Rest of year spent embedding in browser" -- Brendan Eich, ICFP talk, 2006
- Design goals
  - Make it easy to copy/paste snippets of code
  - Tolerate "minor" errors (missing semicolons)
  - Simplified onclick, onmousedown, etc., event handling
  - Pick a few hard-working, powerful primitives
    - First-class functions, objects everywhere, prototype-based
  - Leave all else out!

Common Uses of JavaScript

- Form validation
- Page embellishments and special effects
- Navigation systems
- Basic math calculations
- Dynamic content manipulation
- Sample applications
  - Dashboard widgets in Mac OS X, Google Maps, Philips universal remotes, Writely word processor, hundreds of others...

Client-side JavaScript

- Client-side JavaScript is the most common form of the language. The script should be included in or referenced by an HTML document for the code to be interpreted by the browser.
- It means that a web page need no longer be static HTML, but can include programs that interact with the user, control the browser, and dynamically create HTML content.
- The JavaScript code is executed when the user submits the form, and only if all the entries are valid they would be submitted to the Web Server.
- JavaScript can be used to trap user-initiated events such as button clicks, link navigation, and other actions that the user explicitly or implicitly initiates.

Advantages of JavaScript:

The merits of using JavaScript are:

- **Less server interaction:** You can validate user input before sending the page off to the server. This saves server traffic, which means less load on your server.
- **Immediate feedback to the visitors:** They don't have to wait for a page reload to see if they have forgotten to enter something.
- **Increased interactivity:** You can create interfaces that react when the user hovers over them with a mouse or activates them via the keyboard.
- **Richer interfaces:** You can use JavaScript to include such items as drag-and-drop components and sliders to give a Rich Interface to your site visitors.

Limitations with JavaScript:

- We can not treat JavaScript as a full fledged programming language. It lacks the following important features:
- Client-side JavaScript does not allow the reading or writing of files. This has been kept for security reason.
- JavaScript can not be used for Networking applications because there is no such support available.
- JavaScript doesn't have any multithreading or multiprocess capabilities.

JavaScript Development Tools:

- One of JavaScript's strengths is that expensive development tools are not usually required. You can start with a simple text editor such as Notepad.
- Since it is an interpreted language inside the context of a web browser, you don't even need to buy a compiler.
- To make our life simpler, various vendors have come up with very nice JavaScript editing tools. Few of them are listed here:
  - **Microsoft FrontPage**
  - **Macromedia Dreamweaver MX**
  - **Macromedia HomeSite 5**
  - **Yaldex Software Inc., Free JavaScript Editor**

2. JavaScript Syntax

- A JavaScript consists of JavaScript statements that are placed within the <script>... </script> HTML tags in a web page.
- You can place the <script> tag containing your JavaScript anywhere within you web page but it is preferred way to keep it within the <head> tags.
- The <script> tag alert the browser program to begin interpreting all the text between these tags as a script. So simple syntax of your JavaScript will be as follows (next slide)

Your First JavaScript Script:

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

Example 1:

```
<script language="javascript" type="text/javascript">
<!--

// This is a comment. It is similar to comments in C++

/*
 * This is a multiline comment in JavaScript
 * It is very similar to comments in C Programming
 */
//-->
</script>
```

Whitespace and Line Breaks:

- JavaScript ignores spaces, tabs, and newlines that appear in JavaScript programs.
- Because you can use spaces, tabs, and newlines freely in your program so you are free to format and indent your programs in a neat and consistent way that makes the code easy to read and understand.

Semicolons are Optional:

- Simple statements in JavaScript are generally followed by a semicolon character, just as they are in C, C++, and Java. JavaScript, however, allows you to omit this semicolon if your statements are each placed on a separate line. For example, the following code could be written without semicolons

Case Sensitivity:

- JavaScript is a case-sensitive language. This means that language keywords, variables, function names, and any other identifiers must always be typed with a consistent capitalization of letters.
- So identifiers *Time*, *TIme* and *TIME* will have different meanings in JavaScript.
- **NOTE**: Care should be taken while writing your variable and function names in JavaScript.

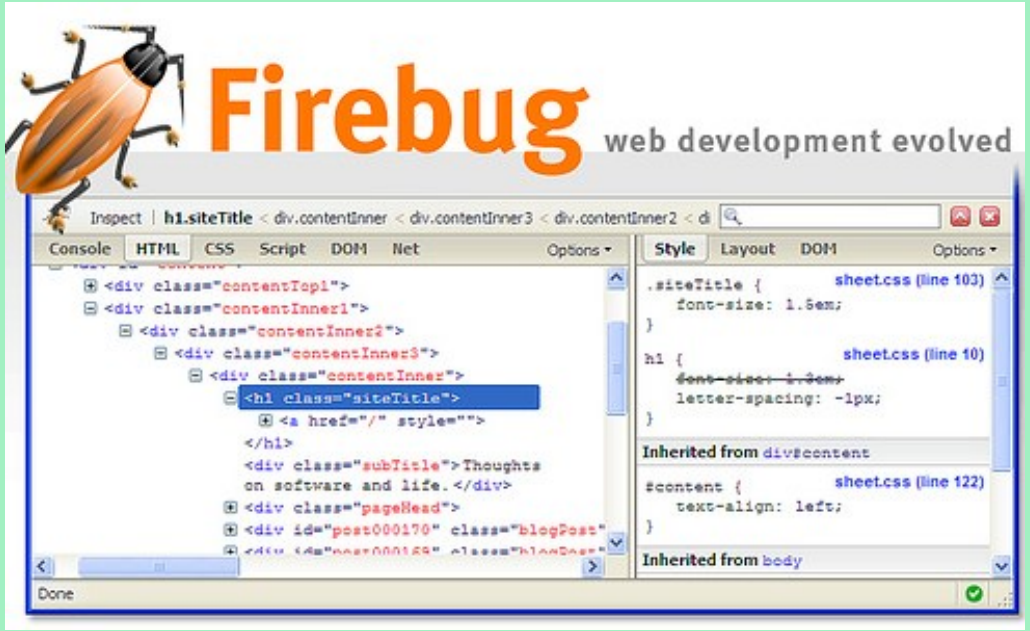
Example 2:

```
<html>
    <head>
        <script>
            function displayDate()
            {
document.getElementById("demo").innerHTML=Date();
            }
        </script>
    </head>
    <body>
        <h1>My First JavaScript</h1>
        <p id="demo">This is a paragraph.</p>

        <button type="button" onclick="displayDate()">Display
   Date</button>
    </body>
</html>
```

Running JS code in a browser

```
<html>
  <head>
    <script src="myfile.js"
            type="text/javascript"></script>
  </head>
  <body>
    <p>My web page</p> ...
  </body>
</html>
```

- Firebug extension



Running JS without a browser

- **CommonJS**: project started in 2009 to create a standard library of JS types and functions for all non-web apps
  - **Rhino** (Mozilla)
  - V8 (Google / Chrome)
  - Narwhal
  - others: Ringo, Joyent, Sprout, Persevere

- **Rhino** runtime available at
  - http://www.mozilla.org/rhino/
  - `java -jar rhino.jar` ***`JSFileName`***

JavaScript is object-based language

- JavaScript is an object-based language.
- JavaScript is based on manipulating objects by modifying an object's properties or by applying methods to an object.
  - ***objects*** are items that have a defined existence
  - each object has ***properties*** that describe its appearance, purpose, or behavior
  - each object has ***methods***, which are actions that can be performed with the object or to it

JavaScript

- **Original name**: Mocha/LiveScript for Netscape Browsers
  - By Brendon Eich; Renamed JavaScript in 1995
  - **Agreement:** Netscape to support Java; Sun allow name change
- **Versions**
  - **JavaScript:** Copyright owned by Oracle
  - **Jscript:** Microsoft's version, named to avoid trademark issues
  - JavaScript submitted by Netscape to the European Computer Manufacturers Association (ECMA) in 1996 for standardization, and ECMAScript was born
  - ActionScript created by MacroMedia for Flash. The current version is compliant with the EMAScript standard
- **Notes**
  - JavaScript, JScript, ActionScript are supersets of ECMAScript
  - http://www.ecma-international.org/publications/standards/Ecma-262.htm

Scope

- **Java: Block level Scope**
  - A new scope starts with each opening brace
- **JavaScript: Functional Scope**

– *Global Variables*: Those declared without var or outside of all functions

– Local variables: Declared with var in a function

– JavaScript allows **functions within functions**. Local variables then live beyond execution **(closure)**, and are accessible by the sub-functions **(lexical scope)**

– Scope can be dynamic in that listener methods within an outer function cannot access its variables.

– References to declared variables can be deleted: var x; delete x;

– Variable declarations are hoisted in JavaScript

– Local scope is the function where declared

```
// Works in JavaScript
function doIt()
{   var sum = 0;
    for (var i=0;i<10;i++)
    { sum += i; }
    for (i=0;i<10;i++)
    {  sum += i; }
    if (sum > 10)
    {  var z = 3; }
    alert(sum+" "+i+" "+z);
}
doIt();
```

this: the local environment existing when a function is called

JavaScript Objects

OBJECT	JAVASCRIPT OBJECT NAME
The browser window	window
A frame within the browser window	frame
The history list containing the Web pages the user has already visited in the current session	history
The Web browser being run by the user	navigator
The URL of the current Web page	location
The Web page currently shown in the browser window	document
A hyperlink on the current Web page	link
A target or anchor on the current Web page	anchor
A form on the current Web page	form

Object and Properties

OBJECT NAME	PROPERTY NAME	DESCRIPTION
window	DefaultStatus	The default message displayed in the window's status bar
	frames	An array of all the frames in the window
	length	The number of frames in the window
	name	The target name of the window
	status	A priority or temporary message in the window's status bar
frame	document	The document displayed within the frame
	length	The number of frames within the frame
	name	The target name of the frame
history	length	The number of entries in the history list
navigator	appCodeName	The code name of the browser
	appName	The name of the browser
	appVersion	The version of the browser
location	href	The URL of the location
	protocol	The protocol (HTTP, FTP, etc.) used by the location
document	bgColor	The page's background color
	fgColor	The color of text on the page
	lastModified	The date the document was last modified
	linkColor	The color of hyperlinks on the page
	title	The title of the page
link	href	The URL of the hyperlink
	target	The target window of the hyperlink (if specified)
anchor	name	The name of the anchor
form	action	The ACTION property of the <FORM> tag
	length	The number of elements in the form
	method	The METHOD property of the <FORM> tag
	name	The name of the form

Modifying a Property's Value

- The syntax for changing the value of a property is:

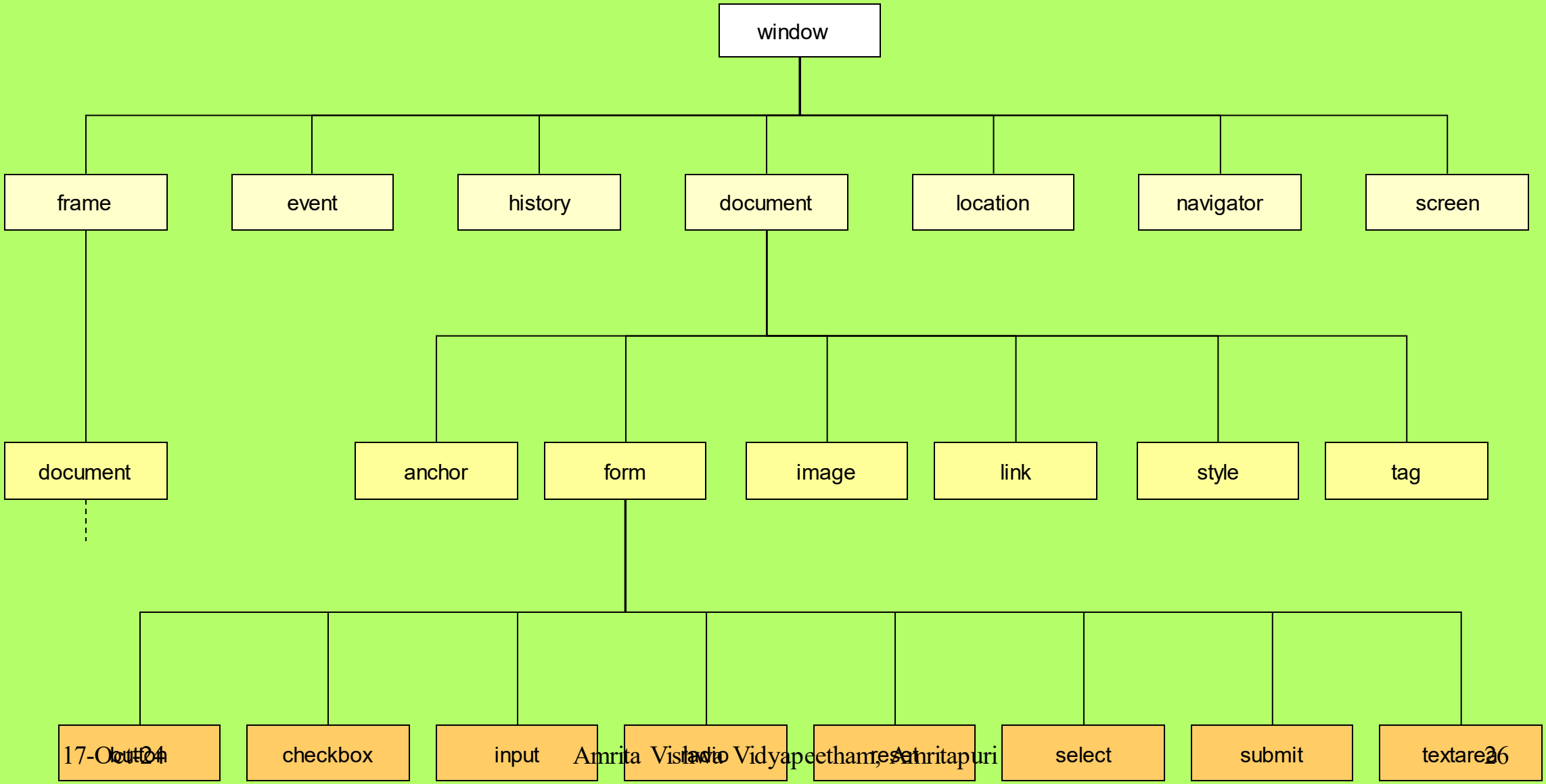
  `object.property = expression`

  – ***object*** is the JavaScript name of the object you want to manipulate

  – ***property*** is a property of that object

  – ***expression*** is a JavaScript expression that assigns a value to the property

A Part of the Document Object Model



Variables

```
var name = expression;
```

- Examples:
  - `var age = 32;`
  - `var weight = 127.4;`
  - `var clientName = "Connie Client";`

- variables are declared with `var` keyword (case sensitive)
- types not specified, but JS does have types
  - `Number`, `Boolean`, `String`, `Array`, `Object`, `Function`, `Null`, `Undefined`
  - can find out a variable's type by calling `typeof`

Numbers

```
var enrollment = 99;
var medianGrade = 2.8;
var credits = 5 + 4 + (2 * 3);
```

- integers and real numbers are the same type
  - (no `int` vs. `double`)
- same operators: `+ - * / % ++ -- = += -= *= /= %=`
  - similar precedence to Java
  - many operators auto-convert types: `"2" * 3` is 6

Number properties/methods

`Number` object "static" properties

`Number.MAX_VALUE`	largest possible number, roughly 10^{308}
`Number.MIN_VALUE`	smallest *positive* number, roughly 10^{-324}
`Number.NaN`	Not-a-Number; result of invalid computations
`Number.POSITIVE_INFINITY`	infinity; result of 1/0
`Number.NEGATIVE_INFINITY`	negative infinity; result of -1/0

`Number` instance methods

`.toString(`***`[base]`***`)`	convert a number to a string with optional base
`.toFixed(`***`digits`***`)`	fixed-point real with given # digits past decimal
`.toExponential(`***`digits`***`)`	convert a number to scientific notation
`.toPrecision(`***`digits`***`)`	floating-point real, given # digits past decimal

global methods related to numbers

`isNaN(`***`expr`***`)`	`true` if the expression evaluates to NaN
`isFinite(`***`expr`***`)`	`true` if ***expr*** is neither NaN nor an infinity

JavaScript keywords

```
break    case     catch  continue debugger
default  delete   do     else     finally
for      function if     in       instanceof
new      return   switch this     throw
try      typeof   var    void     while
with
```

- Reserved words (these don't do anything yet):

```
class   const     enum   export    extends
import  implements interface let package
private protected public static super yield
```

The Math object

```
var rand1to10 = Math.floor(Math.random() * 10 + 1);
var three = Math.floor(Math.PI);
```

- Math methods: abs, ceil, cos, floor, log, max, min, pow, random, round, sin, sqrt, tan
- properties: E, PI

Math properties/methods

`Math.E`	*e*, base of natural logarithms: 2.718...
`Math.LN10, Math.LN2, Math.LOG2E, Math.LOG10E`	natural logarithm of 10 and 2; logarithm of *e* in base 2 and base 10
`Math.PI`	π, circle's circumference/diameter: 3.14159...
`Math.SQRT1_2, Math.SQRT2`	square roots of $^1/_2$ and 2
`Math.abs(`***n***`)`	absolute value
`Math.acos/asin/atan(`***n***`)`	arc-sin/cosine/tangent of angle in radians
`Math.ceil(`***n***`)`	ceiling (rounds a real number up)
`Math.cos/sin/tan(`***n***`)`	sin/cosine/tangent of angle in radians
`Math.exp(`***n***`)`	e^n, *e* raised to the *n*th power
`Math.floor(`***n***`)`	floor (rounds a real number down)
`Math.log(`***n***`)`	natural logarithm (base *e*)
`Math.max/min(`***a***`, `***b***`...)`	largest/smallest of 2 or more numbers
`Math.pow(`***x***`, `***y***`)`	x^y, *x* raised to the *y*th power
`Math.random()`	random real number *k* in range $0 \le k < 1$
`Math.round(`***n***`)`	round number to nearest whole number
`Math.sqrt(`***n***`)`	square root

Comments (same as Java)

```
// single-line comment

/*
multi-line comment
multi-line comment
*/
```

- (identical to Java's comment syntax)

Familiar applications of Javascript

- Cascading menus
- Modify images on mouse “rollover”
- Validate forms before submitting to server
  - Num of chars, non-blank input, non-alpha numerics, etc
- Background (color, music) effects
- “Simple” applications
  - Calendars, payment calculators, etc
- “Complex” applications
  - Google docs
  - Gmail
  - Facebook

NaN and Infinity

```
var1 = 1/0; // evaluates to Infinity (a numeric type)

var1 = -1/0 // evaluates to –Infinity

var1 = 1/"x"; // evaluates to NaN (Not a Number)
```

NaN and Infinity are numeric types!

An explicit conversion is required to turn a string to a number

```
var1 = parseInt("10");
var1 = parseFloat("3.14");
var1 = parseFloat("abc");
```

The numeric value NaN is assigned when a string cannot be parsed.

The opposite conversion from a numeric value to a String is done via:

```
var1 = 100;   // var1 contains numeric value 100
var1 = String(100); // var1  contains "100"
```

Strings

```
var s = "Connie Client";
var firstName = s.substring(0, s.indexOf(" "));
var len = s.length;              // 13
var s2 = 'Melvin Merchant';      // can use "" or ''
```

- String **methods**: `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  - `charAt` returns a one-letter string (there is no `char` type)
  - `length` is a property (not a method as in Java)
- **concatenation** with + : `1 + 1` is 2, but `"1" + 1` is `"11"`
- strings can be **compared** with <, <=, ==, !=, >, >=

String methods

`String.fromCharCode(`***expr***`)`	converts ASCII integer → String
`.charAt(`***index***`)`	returns character at index, as a String
`.charCodeAt(`***index***`)`	returns ASCII value at a given index
`.concat(`***str...***`)`	returns concatenation of string(s) to this one
`.indexOf(`***str[,start]***`)` `.lastIndexOf(`***str[,start]***`)`	first/last index at which given string begins in this string, *optionally* starting from given index
`.match(`***regexp***`)`	returns any matches for this string against the given string or regular expression ("regex")
`.replace(`***old, new***`)`	replaces first occurrence of old string or regular expr. with new string (use regex to replace all)
`.search(`***regexp***`)`	first index where given regex occurs
`.slice(`***start, end***`)` `.substring(`***start, end***`)`	substr. from start (inclusive) to end (exclusive)
`.split(`***delimiter[,limit]***`)`	break apart a string into an array of strings
`.toLowerCase()` `.toUpperCase()`	return new string in all upper/lowercase

More about Strings and numbers

- escape sequences behave as in Java: `\' \" \& \n \t \\`

- convert string to number with `parseInt`, `parseFloat`:

```
var count = 10;
var s1 = "" + count;                    // "10"
var s2 = count + " bananas, ah ah ah!";
var n1 = parseInt("42 is the answer");  // 42
var n2 = parseInt("0x2A", 16);          // 42
var n3 = parseFloat("3.1415");          // 3.1415
var bad = parseInt("booyah");           // NaN
```

- access the letters of a String with `[ ]` or `charAt`:

```
var firstLetter = s[0];
var firstLetter = s.charAt(0);
var lastLetter  = s.charAt(s.length - 1);
```

Arrays in Javascript

```
var myArray; // untyped and undefined
myArray = new Array(); // size not needed
myArray[0] = “Hello”; // array size now 1
myArray[1] = 1; // array size now 2
myArray[2]= new Array(); // array element 3 is another array
```

If an array size is specified, and the number of elements assigned to the array exceeds the size of the array, the array grows itself!

More on arrays

```
var myArray = [0,1,2,3]; // declared & initialized
myArray[4]= -1; // now [0,1,2,3,-1]
myArray[6]= 8; // now [0,1,2,3,-1, undefined, 8]
myArray[0]= “hello”; // legal!
```

The for loop (same as Java)

```
for (initialization; test; update) {
    statements;
}

for (var i = 0; i < 10; i++) {
    print(i + "\n");
}

var s1 = "hi, there!!!",  s2 = "";
for (var i = 0; i < s1.length; i++) {
    var c = s1.charAt(i);
    if (c >= "a" && c <= "z") {
        s2 += c + c;
    }
}
// s2 stores "hhiitthheerree"
```

Logical operators

```
> < >= <= && || ! == != === !==
```

- most logical operators automatically convert types:
  - `5 < "7"` is `true`
  - `42 == 42.0` is `true`
  - `"5.0" == 5` is `true`

- `===` , `!==` are strict equality tests; checks type *and* value
  - `"5.0" === 5` is `false`

The if/else statement

```
if (test) {
    statements;
} else if (test) {
    statements;
} else {
    statements;
}
```

- identical structure to Java's `if/else` statement...
  - but JavaScript allows almost any value as a test!

Boolean type

```
var iLike341 = true;
var ieIsGood = "IE6" > 0;       // false
if ("JS is great") { ... }      // true
if (0 || "") { ... }            // false
```

- any value can be used as a test
  - "falsey" values: `0`, `0.0`, `NaN`, `""`, `null`, and `undefined`
  - "truthy" values: anything else

- converting a value into a boolean explicitly:

```
var boolValue = Boolean(otherValue);
var boolValue = !!(otherValue);
```

&& and || in depth

- `a && b` is a binary operator that returns:
  - if a is truthy, then b, else a
  - *(this turns out to be a truthy/falsey value in the right cases)*
- `a || b` is a binary operator that returns:
  - if a is truthy, then a, else b
  - *(this turns out to be a truthy/falsey value in the right cases)*
- Examples:
  - `0 || 42 || 12 || -1` returns 42 (truthy)
  - `NaN || null || ""` returns "" (falsey)
  - `1 + 1 && 6 && 9` returns 9 (truthy)
  - `3 && 4 && null && 5 && 6` returns null (falsey)

null vs. undefined

```
var ned = null;
var benson = 9;
var caroline;
```

- at this point in the code:
  - `ned` is `null`
  - `benson` is `9`
  - `caroline` is `undefined`

- `undefined`: has not been declared, does not exist
- `null`: exists, but specifically assigned an empty value
  - Why does JavaScript have both of these?

The while loop (same as Java)

```
while (test) {
    statements;
}

do {
    statements;
} while (test);
```

- `break` and `continue` keywords also behave as in Java

Example 1: Add Two Numbers

```
<html>
    …
 <p>  … </p>
<script>
    var num1, num2, sum
    num1 = prompt("Enter first number")
    num2 = prompt("Enter second number")
    sum = parseInt(num1) + parseInt(num2)
    alert("Sum = " + sum)
</script>
    …
</html>
```

Functions

```
function name(paramName, ..., paramName) {
    statements;
}

function myFunction(name) {
    print("Hello, " + name + "!\n");
    print("How are you?\n");
}
```

- unlike in Java, functions are *first-class* (can be stored as variables, passed as parameters, returned, ...)

Functions

- Functions are objects with method called "( )"
  - A property of an object may be a function (=method)
    - function max(x,y) { if (x>y) return x; else return y;};
    - max.description = "return the maximum of two arguments";
  - Local declarations may appear in function body
- Call can supply any number of arguments
  - functionname.length : # of arguments in definition
  - functionname.arguments.length : # arguments in call
  - Basic types are passed by value, objects by reference
- "Anonymous" functions
  - (function (x,y) {return x+y}) (2,3);

Examples of Functions

- Curried functions
  - function CurriedAdd(x) { return function(y){ return x+y} };
  - g = CurriedAdd(2);
  - g(3)
- Variable number of arguments
  - function sumAll() {

```
var total=0;
for (var i=0; i< sumAll.arguments.length; i++)
    total+=sumAll.arguments[i];
return(total); }
```

  - sumAll(3,5,3,5,3,2,6)

Javascript supports the concept of **global variables**

```
var x = "I'm global!";
function myMethod(param1, param2) {
  "use strict";
  var localVar = …
  return localVar + x; // x is visible here
}
```

- Global variables are visible everywhere
- Local variables are only visible in their function scope

Scoping and hoisting

```
function myMethod(param1) {
  "use strict";
  // var x is accessible here (but undefined) due to
  hoisting
  x = 5;
  const z = 3;
  if( param1 > value ) {
      var x = 1; // x is function scoped
     let y = 2; // y is block scoped
  }
  // var x is accessible here; y is NOT
}
```

- JS < 6 does not support block scope

Anonymous functions

```
var adder = function(a, b) {
  "use strict";
  return a+b;
}
var sum = adder(1,3); // called like a normal function!
```

- Note: Recursion is not possible with anonymous functions since internally the function cannot refer to itself by name

Fat arrow functions (JS6)

```
var add = (a,b)=>a+b;
var sum = add(1,2); // called like a normal function!
```

- Similar to Java lambda function

Anonymous Functions

- Anonymous functions very useful for callbacks
  - setTimeout(function() { alert("done"); }, 10000)
  - Evaluation of alert("done") delayed until function call
- Simulate blocks by function definition and call
  - var u = { a:1, b:2 }
  - var v = { a:3, b:4 }
  - (function (x,y) {

```
        var tempA = x.a; var tempB =x.b;  // local variables
        x.a=y.a; x.b=y.b;
        y.a=tempA; y.b=tempB
    }) (u,v)   // Works because objs are passed by ref
```

Built-in Objects

No huge class library, like Java's

- **Number:** Contains various formatting methods
- **Date:** Many methods for time and date
- **Math:** Various methods for common calculations
- **String:** Many methods to manipulate strings
- **Array:** A variety of methods for manipulating arrays
- **RegExp:** search or test against regular expressions
- **DOM based objects:** described in the next set of slides
- **Device:** Part of the HTML 5 specification

JavaScript global functions

Not contained in any Object

1. parseInt(), parseFloat()
2. isNaN(), isFinite()
3. eval()
4. deCodeURI(), enCodeURI()

   Purpose: Replace special characters except: ":", "/", ";", and "?" with %xx, where xx is the corresponding hexadecimal number

   var uri="my test.asp?name=ståle&car=saab";
   document.write(encodeURI(uri)+ "
");

   Output: **my%20test.asp?name=st%C3%A5le&car=saab**

5. escape(), unescape()

   Purpose: Spaces, punctuation, accented characters, and any other non-ASCII characters are replaced with %xx encoding, where xx is the corresponding hexadecimal number. Not good for URI encoding.

   document.write(escape("Need tips? Visit W3Schools!"));

   Output: **Need%20tips%3F%20Visit%20W3Schools%21**

Anonymous Callback Function

Callback function: passed as an argument and called when an event occurs

```
window.addEventListener("load",
function (event)  // Function with an argument (undefined if not there)
{     "use strict";  // Strict syntax verification (ex: no undeclared variables)
      event.preventDefault(); // Prevent browser default behavior
      var element = document.getElementsByTagName("article")[0];
      var family = ["bill", "joe", "mary"], num = family.length, data  = "";
      var i;            // Note: all variables are declared on top (no hoisting)
      for (i=0; i<num; i++)  // Use num, not family.length (much faster)
            data += "<p>" + family[i] + "</p>";
      element.innerHTML = data;
}, false);
```

Browser default behavior
- Navigate on link click
- Right click: context menu
- Cause a form submission
- Jumps to other pages

Note: InnerHTML does not actually create DOM elements for the tags

Creating a simple function

```
function showNames()
{   "use strict";  // Strict syntax verification
    var family = ["bill", "joe", "mary"],
    var num = family.length,
    var data  = "";
    var i;          // Note all variables declared on top
    for (i=0; i<num; i++)  // Use num, not family.length
        data += "<p>" + family[i] + "</p>";
    return data;
}
```

Anonymous Function Returned

Functions declared as variables and called without their name

```
function paint(type, color)
{    var str = "The"+type+" is"+ color;
     var tellMe = function()
     {   document.write(str+".<br> ");  }
     return tellMe;
}
var say1 = paint("rose",  "red");
var say2 = paint("sun", "yellow");
say1();  say2();
```

Note: The variable, str, still exists after the call to paint Completes because it is needed by subsequent calls to the anonymous function

Closure: a function that remembers the environment in which it was created.

Output:

function () { document.write(str + "."); }

The rose is red. The sun is yellow.

Closure-based object creation

Goal: Define StringBuffer class for fast concatenation (like in Java)

Advantage: Looks more like Java with private/public instance variables and method

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

StringBuffer  = function()  // Siulated Class declaration
{  var buffer = [];         // Private variable
   this.append = function(s)  {buffer.push(s);}
   this.toString = function() { return buffer.join("");};
}

var buffer = new StringBuffer();
buffer.append("Happy ");
buffer.append("Day!");
document.write("<h3>" + buffer.toString() + "</h3>");
</script></body></html>
```

Output: Happy Day!

Prototypes, not Classes

Java is class-based; JavaScript is a Prototype-based

- There are no classes present. Prototype objects are enhanced dynamically at run-time
- JavaScript prototypes are of polymorphic associative arrays of functions and properties

```
var object = { x:10, doIT: function() { /** some code **/} or
var object = []; object["x"] = 10; object["doIt"] = function() { … };
alert(object.x); or alert(object.doIt());
```

- **I**nheritance: achieved by adding constructors to existing objects
- Augmented objects serve as prototypes for even newer objects.

Prototype-based properties

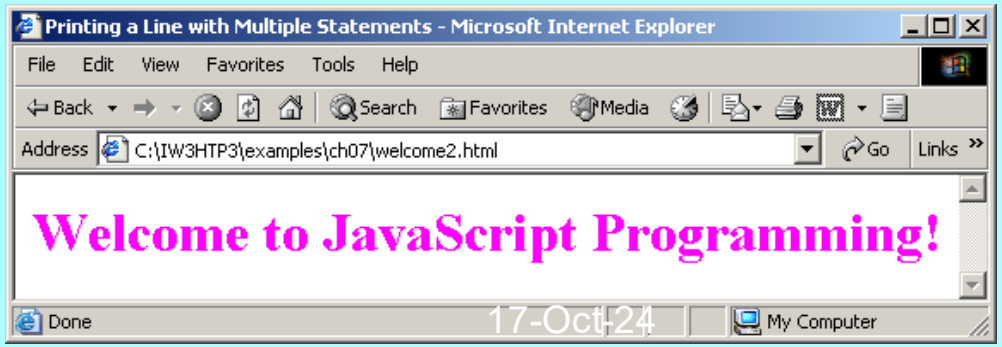
Output: 1 Value = 1 undefined

```
var MyClass = function(){};
MyClass.x = 2; // Add a property to future objects
var my = new MyClass();  // Create an instance
MyClass.prototype.y = 1; // Add a prototype property

// Add a toString() prototype method
MyClass.prototype.toString = function()
{ return "Value="+this.x + " " + this.y; }

document.writeln(my.toString());
```

Note: Every object in JavaScript has a prototype object; adding to it affects all instances, even those previously instantiated

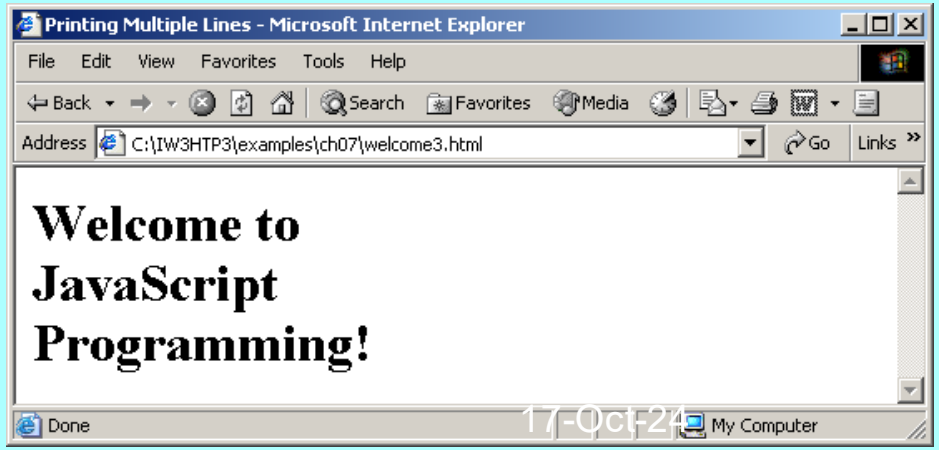
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 7.2: welcome2.html                    -->
6  <!-- Printing a Line with Multiple Statements -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Printing a Line with Multiple Statements</title>
11
12       <script type = "text/javascript">
13          <!--
14          document.write( "<h1 style = \"color: magenta\">" );
15          document.write( "Welcome to JavaScript " +
16             "Programming!</h1>" );
17          // -->
18       </script>
19
20    </head><body></body>
21 </html>
```

←Escape character in combination with quotation mark: \" will result in insertion of a quotation mark in the string that is actually written by JavaScript



```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 7.3: welcome3.html    -->
6  <!-- Printing Multiple Lines    -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head><title>Printing Multiple Lines</title>
10
11       <script type = "text/javascript">
12          <!--
13          document.writeln( "<h1>Welcome to<br />JavaScript" +
14             "<br />Programming!</h1>" );
15          // -->
16       </script>
17
18    </head><body></body>
19 </html>
```

←New line of the html document in a browser is determined by an html
 element



```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 7.4: welcome4.html                  -->
6  <!-- Printing multiple lines in a dialog box -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head><title>Printing Multiple Lines in a Dialog Box</title>
10
11       <script type = "text/javascript">
12          <!--
13             window.alert( "Welcome to\nJavaScript\nProgramming!" );
14          // -->
15       </script>
16
17    </head>
18
19    <body>
20       <p>Click Refresh (or Reload) to run this script again.</p>
21    </body>
22 </html>
```

←`alert` method of the `window` object displays a Dialog box

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 7.8: Addition.html -->
6  <!-- Addition Program        -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>An Addition Program</title>
11
12       <script type = "text/javascript">
13          <!--
14          var firstNumber,   // first string entered by user
15              secondNumber,  // second string entered by user
16              number1,       // first number to add
17              number2,       // second number to add
18              sum;           // sum of number1 and number2
19
20          // read in first number from user as a string
21          firstNumber =
22             window.prompt( "Enter first integer", "0" );
23
```

```
24          // read in second number from user as a string
25          secondNumber =
26             window.prompt( "Enter second integer", "0" );
27
28          // convert numbers from strings to integers
29          number1 = parseInt( firstNumber );
30          number2 = parseInt( secondNumber );
31
32          // add the numbers
33          sum = number1 + number2;
34
35          // display the results
36          document.writeln( "<h1>The sum is " + sum + "</h1>" );
37          // -->
38       </script>
39
40    </head>
41    <body>
42       <p>Click Refresh (or Reload) to run the script again</p>
43    </body>
44 </html>
```

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 7.16: welcome6.html    -->
6  <!-- Using Relational Operators -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Using Relational Operators</title>
11
12       <script type = "text/javascript">
13          <!--
14          var name, // string entered by the user
15             now = new Date(),      // current date and time
16             hour = now.getHours(); // current hour (0-23)
17
18          // read the name from the prompt box as a string
19          name = window.prompt( "Please enter your name", "GalAnt" );
20
21          // determine whether it is morning
22          if ( hour < 12 )
23             document.write( "<h1>Good Morning, " );
24
```

"now" is a new instance of JavaScript native object `Date`. It can invoke all the methods of that object class

Note that conversion to integer type was not needed when the value was returned by the `getHours` method

```
25          // determine whether the time is PM
26          if ( hour >= 12 )
27          {
28             // convert to a 12 hour clock
29             hour = hour - 12;
30
31             // determine whether it is before 6 PM
32             if ( hour < 6 )
33                document.write( "<h1>Good Afternoon, " );
34
35             // determine whether it is after 6 PM
36             if ( hour >= 6 )
37                document.write( "<h1>Good Evening, " );
38          }
39
40          document.writeln( name +
41             ", welcome to JavaScript programming!</h1>" );
42          // -->
43       </script>
44
45    </head>
46
```

Basic Object Features

- Use a function to construct an object
  - function car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year; }
- Objects have prototypes, can be changed
  - var c = new car("Ford","Taurus",1988);
  - car.prototype.print = function () {
    return this.year + " " + this.make + " " + this.model;}
  - c.print();

JavaScript in Web Pages

- Embedded in HTML page as <script> element
  - JavaScript written directly inside <script> element
    - <script> alert("Hello World!") </script>
  - Linked file as src attribute of the <script> element

    <script type="text/JavaScript" src="functions.js"></script>
- Event handler attribute

  <a href="http://www.yahoo.com " onmouseover="alert('hi');">
- Pseudo-URL referenced by a link

  Click me

We are looking at JavaScript as a language; ignore BOM, DOM, AJAX

Javascript Apps

- Using Apache Cordova, it is possible to package JavaScript apps as mobile apps.
- AppJS can package a JavaScript app as a desktop app.
- Docker solves the problem of Node portability.
- apps may exist entirely in the browser and use cloud APIs.
- React Native framework allows cross-platform mobile app building, where developers can use a universal front-end for Android and IOS platforms.