Overview of JavaScript

- Originally developed by Netscape, as LiveScript
- Became a joint venture of Netscape and Sun in 1995, renamed JavaScript
- Now standardized by the European Computer Manufacturers Association as ECMA-262 (also ISO 16262)
- JavaScript can be divided into three categories, core (this chapter), client-side (Chapters 5 & 6), and server-side (not covered in this book)
- We'll call collections of JavaScript code *scripts*, not programs

More Basics

- JavaScript and Java are only related through syntax
- JavaScript is dynamically typed
- JavaScript's support for objects is very different (it's not really object oriented!)
- JavaScript be embedded in many different things, but its primary use is within HTML documents

Overview

- JavaScript can be used to replace some of what is typically done with applets (except graphics)
- JavaScript can be used to replace some of what is done with CGI (but no file operations or networking)
- Interacts very well with html *forms*
- The Document Object Model makes it possible to support dynamic HTML documents with JavaScript
- *Event-Driven Computation (See Chapter 5)*
  - User interactions with HTML documents in JavaScript use the event- driven model of computation
- User interactions with form elements can be used to trigger execution of scripts

HTML/JavaScript Documents

- The document *head* holds function definitions and code associated with widgets
- The document *body* holds code that is interpreted once, when found by the browser
  - This code often dynamically generates html code:

```
<html> <head> <title>JavaScript Example 1</title> </head>
<body>
<script language=javascript>
for(i=0; i<10;i ++)
  if (i%2) document.write("<br>i is ",i," and i squared is ",i*i);
  else document.write("<br><b>i is ",i," and i squared is i,"</b>");
</script>
</body> </html>
```

Object Orientation?

- JavaScript is NOT an object-oriented programming language
- Does not support class-based inheritance
  - Cannot support polymorphism
  - Has prototype-based inheritance, which is much different
- *JavaScript "Objects":*
- JavaScript objects are collections of *properties*,
  - like the members of classes in Java and C++
- Properties can be *data properties* or *method properties*
- JavaScript has primitives for simple types
- All JavaScript objects are accessed via references
- Each object appears as a list of property-value pairs
  - properties can be added or deleted dynamically
  - Syntax: objectRef.propName

General Syntax

- Typically JavaScript scripts are embedded in HTML documents
  - Either directly, as the content of the <script> tag whose language attribute is set to "JavaScript"

```
<script language = "JavaScript">
  -- JavaScript script --
  </script>
```

  - Or indirectly, as a file specified in the src attribute of <script>, as in

```
<script language = "JavaScript"
   src = "myScript.js">
</script>
```

More syntax

- *Identifiers*: begin with a letter or underscore, followed by any number of letters, underscores, and digits
  - Case sensitive
  - 25 reserved words, plus future reserved words (basically same as in Java)
- Comments: both // and /* … */

Scripts within HTML

- Scripts are often hidden from browsers that do not include JavaScript interpreters by commenting them:

```
<!--
-JavaScript script –
//-->
```

- JavaScript statements usually do not need to be terminated by semicolons, but most programmers do so

Primitives

- All primitive values have one of the five primitive types:
  - Number, String, Boolean, Undefined, or Null
- Number, String, and Boolean have wrapper “classes” (Number, String, and Boolean)
- In the cases of Number and String, primitive values and objects are coerced back and forth so that primitive values can be treated essentially as if they were objects

Primitives (cont.)

- Numeric literals – just like Java
- All numeric values are stored in double-precision floating point
- String literals are delimited by either ' or "
  - Can include escape sequences (e.g., \t)
  - Embedded variable names are NOT interpolated
  - All String literals are primitive values
  - Ex: “Ben said, \“ here\’s to you!\””

Primitives (yet more)

- **Boolean values are *true* and *false***
- **The only Null value is *null***
- **The only Undefined value is *undefined***

Dynamically Typed

- **JavaScript is dynamically typed – any variable can be used for anything (primitive value or reference to any object)**
- **The interpreter determines the type of a particular occurrence of a variable**
- **Variables can be either implicitly or explicitly declared:**

```
var sum = 0,
 today = "Monday",
 flag = false;
```

Operators

- **Numeric operators for primitives ++, --, +, -, *, /, %**
  - **All operations are double precision**
  - **Same precedence and associativity as Perl**
- **The Math Object**
  - **Provides methods that operate on Numbers**
  - **floor, round, max, min, trig functions, etc.**
  - **Ex: Math.round(x)**

Number Object

- **The Number Object**
  - **Some useful (constant) properties:**
  - **MAX_VALUE, MIN_VALUE, NaN, POSITIVE_INFINITY, NEGATIVE_INFINITY, PI**
  - **e.g., Number.MAX_VALUE**
- **An arithmetic operation that creates overflow returns *NaN***
  - **NaN is not == to any number, not even itself**
  - **Test for it with isNaN(x)**
- **Number object has the method, toString**
  - **Number.toString(x)**

String operators

- String catenation operator: +
- Coercions
  - Catenation coerces numbers to strings
    - Ex: 3 + "bob"
  - Numeric operators (other than +) coerce strings to numbers
    - Ex: 3 * "4"
  - Conversions from strings to numbers that do not work return NaN

String properties & methods

- length e.g., var len = str1.length; (a property)
- charAt(position) e.g., str.charAt(3)
- indexOf(string) e.g., str.indexOf('B')
- substring(from, to) e.g., str.substring(1, 3)
- toLowerCase() e.g., str.toLowerCase()

More operations

- *Conversion functions* (not called through string objects, because they are not methods)
  - parseInt(string) and parseFloat(string)
  - The string must begin with a digit or sign and have a legal number; otherwise NaN is returned
  - Not often needed because of implicit coersions
- *The* typeof *operator*
  - Returns "number", "string", or "boolean" for primitives; returns "object" for objects and null
  - Ex: typeof(x)
- *Assignment statements* – just like C++ and Java

Output

- **The JavaScript model for the HTML document is the Document object**
- **The model for the browser display window is the Window object**

Screen (browser) output

- The Window object has two properties, *document* and *window*, which refer to the Document and Window objects, respectively
- The Document object has a method, *write*, which dynamically creates content
  - The parameter is a string, often catenated from parts, some of which are variables:

```
document.write("Answer: " + result +"<br>");
```

  - The parameter is sent to the browser, so it can be anything that can appear in an HTML document (e.g. <br>, but not \n)

Dialog boxes

- **The Window object has three methods for creating dialog boxes:**
  - **alert, confirm, and prompt**
- **The default object is the current window, so the object need not be included in the call to any of these three**

Alert dialog box

- alert("Hey! \n");
- **Parameter is plain text, not HTML**
- **Opens a dialog box that displays the parameter string and an OK button**
- **It waits for the user to press the OK button**

Confirm dialog box

- confirm("Do you want to continue?");
- **Opens a dialog box and displays the parameter and two buttons, OK and Cancel**
- **Returns a Boolean value, depending on which button was pressed (it waits for one)**

Prompt dialog boxes

- prompt("What is your name?", "");
- **Opens a dialog box and displays its string parameter, along with a text box and two buttons, OK and Cancel**
- **The second parameter is for a default response if the user presses OK without typing a response in the text box (waits for OK)**
- http://goshawk.emich.edu/%7Esverdlik/JavaScript3.html

Control Statements

- **Syntax is similar to C, Java, and C++**
- **Compound statements are delimited by braces, but compound statements are not blocks (cannot declare local variables)**

Conditional expressions

- **Three kinds: primitive, relational, compound**

- **1. *Primitive values***
  - **If it is a string, it is *true* unless it is empty or "0"**
  - **If it is a number, it is *true* unless it is zero**

Relational conditionals

- ***The usual six*: ==, !=, <, >, <=, >=**

- **Operands are coerced if necessary**
  - **If one is a string and one is a number, it attempts to convert the string to a number. If one is Boolean and the other is not, the boolean operand is coerced to a number (1 or 0)**

- ***The unusual two*: === and !==**
  - **Same as == and !=, except that no coercions are done (operands must be identical)**
  - **Comparisons of references to objects are not useful (addresses are compared, not values)**

Compound Conditionals

- The usual logical operators: &&, | |, and !
- The primitive values, true and false, must not be confused with the Boolean object properties
- If a Boolean object is used in a conditional expression, it is false only if it is null or undefined
  - Instead, use code something like x == Boolean.true
- The Boolean object has a method, toString, to allow them to be printed (true or false)

Selection statements

- The usual if-then-else statements
- *Switch:*

```
switch (expression) {
  case value_1:
    // value_1 statements
  case value_2:
    // value_2 statements
  ...
  [default:
    // default statements]
}
```

- The statements can be either statement sequences or compound statements
- In most situations, the cases end with break
- The control expression can be a number, a string, or a Boolean

Iterations

- The usual:
  - while (…) { … }
  - do { …} while (…)
  - for(x; y; z) { … }

Object Creation

- **Objects can be created with new**
- **The most basic object is one that uses the Object constructor, as in**

  var myObject = new Object();
- **The new object has <u>no properties.</u> It is a blank object**
- **Properties can be added to an object, any time**

Object modification

```
var myAirplane = new Object();
myAirplane.make = "Cessna";
myAirplane.model = "Centurian";
```

- **Objects can be nested, so a property could be itself another object, created with *new***
- **Properties can be accessed by dot notation or in array notation, as in**

```
var property1 = myAirplane["model"];
property1 = myAirplane.model;
```

More object modification

- **If you try to access a property that does not exist, you get *undefined***
- **Properties can be deleted with delete, as in**

```
delete myAirplane.model;
```

Iteration over properties

- **for (*identifier* in *object*) *statement or compound***

```
for (var prop in myAirplane)
  document.write(myAirplane[prop] +
  "<br>");
```

Arrays

- **Objects with some special functionality**
- **Elements can be primitive values or references to other objects**
- **Length is dynamic. The *length* property stores the length**
- **Array objects can be created in two ways, with new, or by assigning an array literal**

```
var myList = new Array(24, "bread", true);
var myList2 = [24, "bread", true];
var myList3 = new Array(24);
```

Arrays (cont)

- **The length of an array is the highest subscript to which an element has been assigned, plus 1**

  myList[122] = "bitsy"; // length is 123

- **Because the length property is writeable, you can set it to make the array any length you like, as in**

  myList.length = 150;

- **This can also shorten the array (if the new length is less than the old length)**
- **Only assigned elements take space (sparse representation)**
- See insert_names.html

Array operators & methods

- **join – e.g., var listStr = list.join(", ");**

- **reverse**
- **sort -- Coerces elements to strings and puts them in alphabetical order**
- **concat – e.g., newList = list.concat(47, 26);**
- **slice**
  - **listPart = list.slice(2, 5);**
  - **listPart2 = list.slice(2);**

- **toString -- Coerce elements to strings, if necessary, and catenate them together, separated by commas (exactly like join(", "))**
- **push, pop, unshift, and shift**
- **See nested_arrays.html**

Functions

```
function function_name([formal_parameters]) {
  -body –
 }
```

- **Return value is the parameter of function's *return***
- **If there is no return, or if the return has no parameter or if the end of the function is reached, undefined is returned**
- **Functions are objects, so variables that reference them can be treated as other object references (can be passed as parameters, assigned to variables, and be elements of an array)**

More functions

- **If fun is the name of a function,**

```
   ref_fun = fun;
  /* Now ref_fun is a reference to fun */
 ref_fun();  /* A call to fun */
```

- **We place all function definitions in the head of the the HTML document, and all calls in the body**
- **All variables that are either implicitly declared or explicitly declared outside functions are global**
- **Variables explicitly declared in a function are local**
- **Functions can be nested, but why make life complicated!?**

Function Parameters

- **Parameters are passed by value, but when a reference variable is passed, the semantics are pass-by-reference. This is identical to the way objects are passed in Java.**
- **There is no type checking of parameters, nor is the number of parameters checked (excess actual parameters are ignored, excess formal parameters are set to undefined)**
- **All parameters are sent through a property array, *arguments*, which has the length property**
- **See parameters.html**

Primitive parameters

- **There is no clean way to send a scalar by reference. One dirty way is to put the value in an array and send the array's name:**

```
function by10(a) {  /* a is an array */
    a[0] *= 10;
}
...
var listx = new Array(1);  /*serves as wrapper around primitive*/
...
listx[0] = x;
by10(listx);
x = listx[0];
```

Example functions

- **To sort something other than strings into alphabetical order, write a 2-argument function that performs the comparison and provide it to the *sort* method**
- **This comparison function, f(a,b), must return a negative number, zero, or a positive number to indicate whether a<b, a=b, or a>b**
- **For example, to sort numbers we could define a simple comparison function, *num_order*, as**

  function num_order(a, b) {return a   b;}

- **Now, we can sort an array named num_list with:**

  num_list.sort(num_order);

An Example

Function `median`: Given an array of numbers, return the median of the array

```
function median(list) { /* Use anonymous function to sort */
  list.sort(function (a, b) {return a-b;});
  var list_len = list.length;

// Use the modulus operator to determine whether the array's
// length is odd or even.
// Use Math.floor to truncate numbers
// Use Math.round to round numbers

  if ((list_len % 2) == 1) /* take the middle number */
    return list[Math.floor(list_len / 2)];
  else /* take average of middle two numbers */
    return Math.round((list[list_len / 2 + 1]
                   + list[list_len / 2]) / 2);
}  // end of function median
```

Constructors

- *new* is always followed by name of a constructor.
- Several constructors are pre-defined (Object, Array, etc.)
- In constructors, *this* is a reference to the object being created

```
function plane(newMake, newModel, newYear){
  this.make = newMake;
  this.model = newModel;
  this.year = newYear;
}

myPlane = new plane("Cessna", "Centurnian", "1970");
```

Method properties

- **Objects can also have method properties**

```
function displayPlane() { /* Method */
  document.write("Make: ", this.make, "<br />");
  document.write("Model: ", this.model,"<br />");
  document.write("Year: ", this.year, "<br />");
```

- **Now add the following to the constructor:**

```
  this.display = displayPlane;
```

- **Now this "method" can be invoked:**

```
 var dp = new Plane(); …
 dp.display();
```

Pattern Matching

- **Patterns are based on those of Perl**
  - **Patterns are usually surrounded by '/' characters.**
  - **Each pattern is a regular expression**
  - **Ex: /abc/, /[abc]de/, /a.*b/**
- **JavaScript has two approaches to pattern-matching operations, but we will cover just one: pattern-matching operations as methods of the String object**

Regular expressions

- */pattern/modifier*
  - Modifier: g = global, i=ignore case, m=multiline
- Normal characters match themselves
- Metacharacters are "wildcards":
  - |(){}[]
  - $^
  - *+?.
  - The \ operator can convert a metacharacter into a normal character:
    - /Match an asterisk with */

Pattern-matching functions

- **There are four basic pattern-matching operators: search, replace, match, split**
- **1. search(pattern)**
- **Returns the position of *pattern* in the object string (position is relative to zero); -1 if failure**
- **After, $1 will be the substring that matched *pattern***

```
var str = "Gluckenheimer";
 var position = str.search(/[nm]/);
    /* position is now 6, $1 is "n" */
```

Replace (patterns)

- **2. replace(pattern, string)**
  - **Finds a substring in object string that matches *pattern* and replaces it with *string* (g (global) modifier can be used)**

```
var str = "Some Rabbits are rabid";
 str.replace(/rab/ig, "tim");
```

- ***str* is now "Some timbits are timid"**
- **$1 is "Rab" and $2 is "rab"**
  - **$n are global vars, set after each pattern function**

The *match* pattern function

- **match(pattern)**

- **The most general pattern-matching method (and slowest)**
- **With the g modifier, returns an array of the substrings that matched**

```
var str = "My 3 kings beat your 2 aces";
var matches = str.match(/[ab]./g);
```

- ***matches* is set to ["be", "at", "ac"]**

More *match*

- **Without the g modifier, first element of the returned array is the matched substring, the other elements are the substrings that matched any parenthesized expressions in *pattern***

```
var str = “I have 20 dollars and 15 cents";
var matches = str.match(/(\d+)([^\d]+)(\d+)/);
```

Afterward, *matches* = [“20 dollars and 15”, “20”, “ dollars and ”, “15”]

The split operator

- **split(parameter)**
- **Like the Perl split operator**
- **The parameter could be a string or a pattern.**
  - **"," and /,/**
- **In either case, it is used to split the string into substrings and return an array of them**

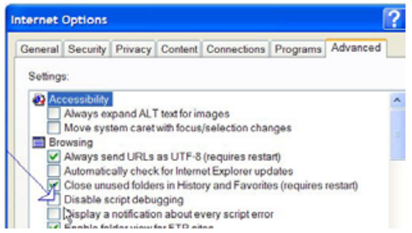
  var str = "128.4.64.127";

  matches = str.split(/\./);
- **Now, *matches*=["128","4","64","127"]**
- **See forms_check.html**

Debugging JavaScript in IE

1. **Select *Internet Options* from the *Tools* menu**
2. **Choose the *Advanced* tab**
3. **Uncheck the *Disable script debugging* box**
4. **Check the *Display a notification about every script error* box**

- **Now, a script error causes a small window to be opened with an explanation of the error**



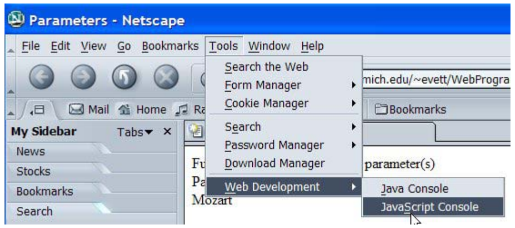
Settings:

Accessibility
- [ ] Always expand ALT text for images
- [ ] Move system caret with focus/selection changes

Browsing
- [✓] Always send URLs as UTF-8 (requires restart)
- [ ] Automatically check for Internet Explorer updates
- [✓] Close unused folders in History and Favorites (requires restart)
- [ ] Disable script debugging
- [ ] Display a notification about every script error

Debugging in Netscape

- Select *Tools*, *Web Development* and *JavaScript Console*
- A small window appears for displaying script errors
- Remember to Clear the console after dealing with an error message – avoids confusion


Parameters - Netscape
File Edit View Go Bookmarks Tools Window Help
Search the Web
Form Manager
Cookie Manager
Search
Password Manager
Download Manager
Web Development
Java Console
JavaScript Console
mich.edu/~evett/WebProgra
Mail Home
Bookmarks
My Sidebar Tabs
News
Stocks
Bookmarks
Search
parameter(s)
Mozart