Web Programming Step by Step

Chapter 7
JavaScript for Interactive Web Pages

Except where otherwise noted, the contents of this presentation are Copyright 2009 Marty Stepp and Jessica Miller.

7.1: Key JavaScript Concepts

What is JavaScript? (7.1)

- a lightweight programming language (scripting)
- used to make web pages interactive
  - insert dynamic text into HTML (ex: user name)
  - react to events (ex: page load user click)
  - get information about a user's computer (ex: browser type)
  - perform calculations on user's computer (ex: form validation)
- a web standard (but not supported identically by all browsers)
- NOT related to Java other than by name and some syntactic similarities

JavaScript vs. Java

- **interpreted**, not compiled
- more relaxed syntax and rules
  - fewer and "looser" data types
  - variables don't need to be declared
  - errors often silent (few exceptions)
- key construct is the **function** rather than the class
  - (more procedural less object-oriented)

+

=

JavaScript

- contained within a web page and integrates with its HTML/CSS content

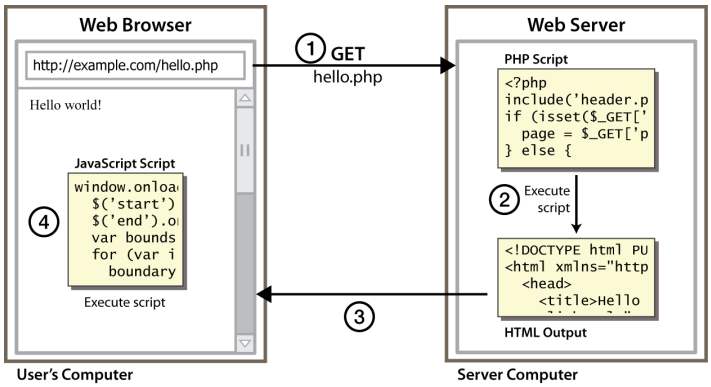
JavaScript vs. PHP

- similarities:
  - both are **interpreted**, not compiled
  - both are relaxed about syntax, rules, and types
  - both are case-sensitive
  - both have built-in regular expressions (powerful text processing)

JS <3 

- differences:
  - JS is less procedural (`verb(noun)`), more object-oriented (`noun.verb()`)
  - JS focuses on user interfaces and interacting with a document; PHP is geared toward HTML output and file/form processing
  - JS code runs on the client's browser; PHP code runs on the web server

Client-side scripting (7.1.1)



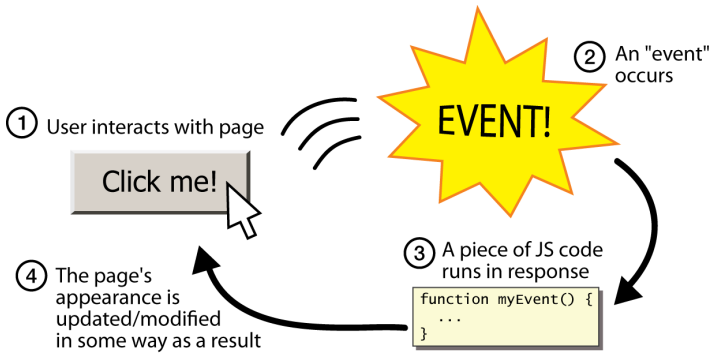
- **client-side script**: code runs in browser after page is sent back from server
  - often this code manipulates the page or responds to user actions

Why use client-side programming?

PHP already allows us to create dynamic web pages. Why also use a client-side language like JavaScript?

- PHP benefits:
  - **security**: has access to server's private data; client can't see source code
  - **compatibility**: avoids browser compatibility issues
  - **power**: fewer restrictions (can write to files, open connections to other servers, connect to databases, ...)
- JavaScript benefits:
  - **usability**: can modify a page without having to post back to the server (faster UI)
  - **efficiency**: can make small, quick changes to page without waiting for server
  - **event-driven**: can respond to user actions like clicks and key presses

Event-driven programming



- most languages' programs start with a `main` method and run sequentially
- JavaScript programs wait for user actions called **events** and respond to them
- **event-driven programming**: writing programs driven by user events

Buttons: `<button>`

the most common clickable UI control (inline)

```HTML
<button>Click me!</button>
```

Click me!

output

- button's text appears inside tag; can also contain images
- To make a responsive button or other UI control:
  1. choose the control (e.g. button) and event (e.g. mouse click) of interest
  2. write a JavaScript function to run when the event occurs
  3. attach the function to the event on the control

Linking to a JavaScript file: `script`

```HTML
<script src="filename" type="text/javascript"></script>
```

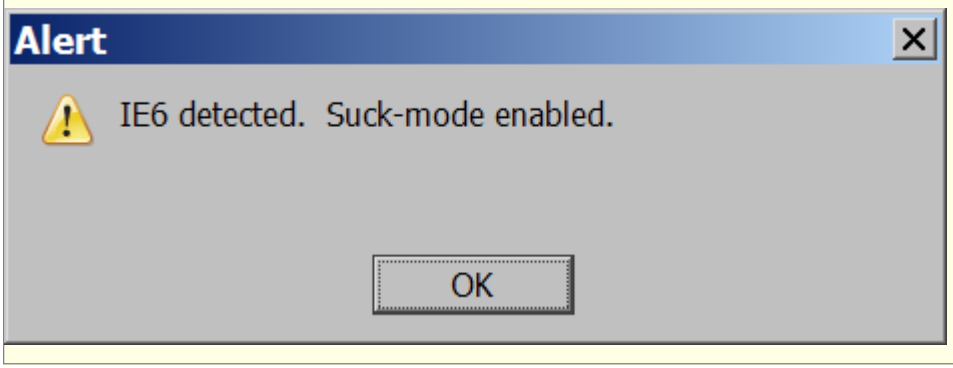
```HTML
<script src="example.js" type="text/javascript"></script>
```

- `script` tag should be placed in HTML page's `head`
- script code is stored in a separate `.js` file
- JS code can be placed directly in the HTML file's `body` or `head` (like CSS)
  - but this is bad style (should separate content, presentation, and behavior)

A JavaScript statement: `alert`

```
alert("message");
```
JS

```
alert("IE6 detected.  Suck-mode enabled.");
```
JS



output

- a JS command that pops up a dialog box with a message

JavaScript functions

```
function name() {
  statement ;
  statement ;
  ...
  statement ;
}
```
JS

```
function myFunction() {
  alert("Hello!");
  alert("How are you?");
}
```
JS

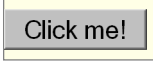
- the above could be the contents of `example.js` linked to our HTML page
- statements placed into functions can be evaluated in response to user events

Event handlers

```HTML
<element attributes onclick="function();">...
```

```HTML
<button onclick="myFunction();">Click me!</button>
```

output

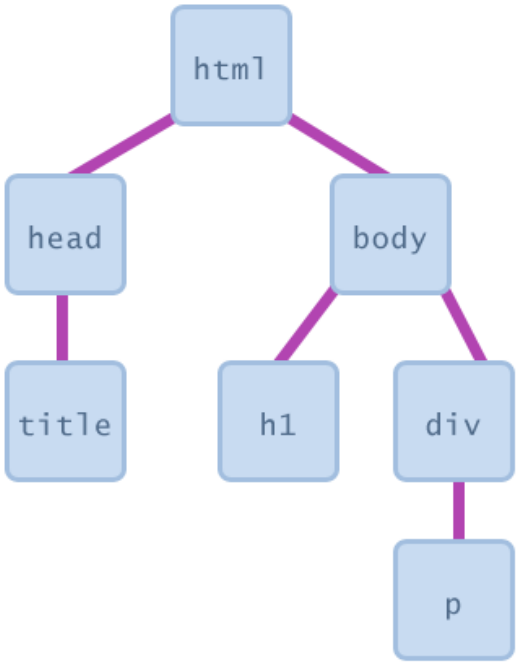


- JavaScript functions can be set as **event handlers**
  - when you interact with the element, the function will execute
- `onclick` is just one of many event HTML attributes we'll use

Document Object Model (DOM) (7.1.4)

a set of JavaScript objects that represent each element on the page

- most JS code manipulates elements on an HTML page
- we can examine the state of the elements, e.g. whether a box is checked
- we can change state, e.g. putting text into a `div`
- we can change styles, e.g. make a paragraph red



Accessing elements: `getElementById`

```JS
var name = document.getElementById("id");
```

```HTML
<button onclick="myFunction2();">Click me!</button>
<span id="output">replace me</span>
```

```JS
function myFunction2() {
  var span = document.getElementById("output");
  span.innerHTML = "Hello, how are you?";
}
```

Click me! replace me

output

- `document.getElementById` returns DOM object for an element with a given `id`
- can change the text inside most elements by setting `innerHTML` property

7.2: JavaScript Syntax

- 7.1: Key JavaScript Concepts
- **7.2: JavaScript Syntax**
- 7.3: Program Logic
- 7.4: Advanced JavaScript Syntax

Variables and types (7.2.1, 7.2.3)

```
var name = expression;
```
JS

```
var clientName = "Connie Client";
var age = 32;
var weight = 127.4;
```
JS

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

`Number` type (7.2.2)

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

- 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`

Comments (same as Java) (7.2.4)

```
// single-line comment

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

- identical to Java's comment syntax
- recall: 4 comment syntaxes
  - HTML: `<!-- comment -->`
  - CSS/JS/PHP: `/* comment */`
  - Java/JS/PHP: `// comment`
  - PHP: `# comment`

DOM object properties (7.2.5)

```
<div id="main" class="foo bar">
  <p>Hello, <em>very</em> happy to see you!</p>
  <img id="icon" src="images/borat.jpg" alt="Borat" />
</div>
```
HTML

```
var div = document.getElementById("main");
var image = document.getElementById("icon");
```
JS

- `tagName`: element's HTML tag, capitalized; `div.tagName` is `"DIV"`
- `className`: CSS classes of element; `div.className` is `"foo bar"`
- `innerHTML`: HTML content inside element; `div.innerHTML` is `"\n <p>Hello, <em>very</em> happy to ..."`
- `src`: URL target of an image; `image.src` is `"images/borat.jpg"`

DOM properties for other elements

```
<input id="studentid" type="text" size="7" maxlength="7" />
<input id="freshman" type="checkbox" checked="checked" /> Freshman?
```
HTML

```
var sid = document.getElementById("studentid");
var frosh = document.getElementById("freshman");
```
JS

output

 Freshman?

- `value`: the text in an input control
  - `sid.value` could be `"1234567"`
- `checked`, `disabled`, `readOnly`: whether a control is selected/disabled/etc.
  - `frosh.checked` is `true`

Debugging common errors (7.2.6)

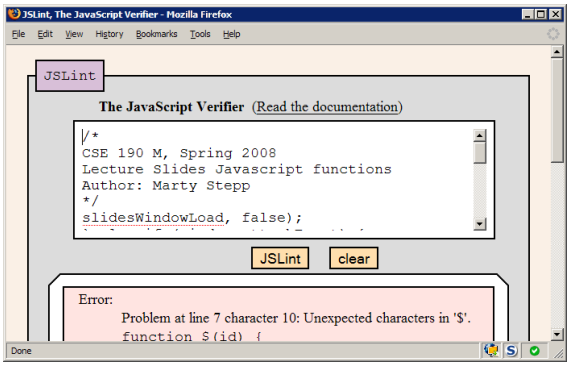
- JavaScript's syntax is looser than Java's, but its errors are meaner
  - most errors produce no visible output or error message!
- some common error symptoms:
  - "My program does nothing." (most errors produce no output)
  - "It just prints `undefined`." (many typos lead to undefined variables)
  - "I get an error that says, `foo` has no properties."

Debugging JS code in Firebug

- Firebug JS debugger can set breakpoints, step through code, examine values (Script tab)
- interaction pane for typing in arbitrary JS expressions (Console tab; Watch tab within Script tab)

JSLint

- **JSLint**: an analyzer that checks your JS code, much like a compiler, and points out common errors
  - Marty's version
  - original version, by Douglas Crockford of Yahoo!
- when your JS code doesn't work, paste it into JSLint first to find many common problems



Debugging checklist

- Are you sure the browser is even loading your JS file at all?
  Put an `alert` at the top of it and make sure it appears.
- When you change your code, do a **full browser refresh (Shift-Ctrl-R)**
- Check bottom-right corner of Firefox for Firebug syntax errors.
- Paste your code into our JSLint tool to find problems.
- Type some test code into Firebug's console or use a breakpoint.

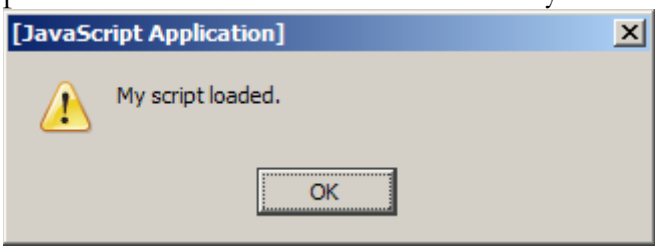
"My program does nothing"

Since Javascript has no compiler, many errors will cause your Javascript program to just "do nothing." Some questions you should ask when this happens:

- Is the browser even loading my script file?
- If so, is it reaching the part of the file that I want it to reach?
- If so, what is it doing once it gets there?

Is my JS file loading?

- put an `alert` at the VERY TOP of your script:

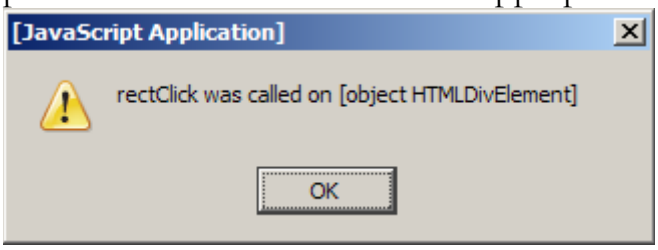


- if it shows up, good!
- if it doesn't show up:
  - maybe your HTML file isn't linking to the script properly
    - double-check file names and directories
  - maybe your script has a syntax error
    - check bottom-right for Firebug error text 

    - comment out the rest of your script and try it again
    - run your script through JSLint to find some syntax problems

Is it reaching the code I want it to run?

- put an `alert` at the start of the appropriate function:



  - write a descriptive message, not just `"hello"` or `"here"`
- if it shows up, good!
- if it doesn't show up:
  - if it's an event handler, maybe you didn't attach it properly
  - maybe your script has a syntax error; run JSLint

Object 'foo' has no properties

- these errors mean you are trying to utilize an undefined value:
  - `Object foo has no properties`
  - `ReferenceError: foo is not defined`
  - `TypeError: foo.bar is not a function`
- possible causes:
  - you're trying to access a variable that is out of scope
  - you're trying access a DOM element with an invalid `id`
  - you've run off the bounds of an array
  - you've spelled the variable's name incorrectly

Common bug: bracket mismatches

```
function foo() {
  ...   // missing closing curly brace!

function bar() {
  ...
}
```
JS

- JS doesn't always tell us when we have too many/few brackets
  - (it is legal in JavaScript to declare one function inside another)
- **symptom**: script becomes (fully or partially) non-functional
- **detection**: bracket matching in TextPad (highlight bracket, press Ctrl-M); using an Indenter tool; JSLint

`String` type (7.2.7)

```
var s = "Connie Client";
var fName = s.substring(0, s.indexOf(" "));   // "Connie"
var len = s.length;                           // 13
var s2 = 'Melvin Merchant';
```
JS

- methods: `charAt`, `charCodeAt`, `fromCharCode`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  - `charAt` returns a one-letter `String` (there is no `char` type)
- `length` property (not a method as in Java)
- Strings can be specified with `""` or `''`
- concatenation with + :
  - `1 + 1` is 2, but `"1" + 1` is `"11"`

More about `String`

- escape sequences behave as in Java: `\' \" \& \n \t \\`
- converting between numbers and `String`s:

```
var s1 = String(myNum);
var s2 = count + " bananas, ah ah ah!";
var n1 = parseInt("42 is the answer");  // 42
var n2 = parseFloat("booyah");  // NaN
```
JS

- accessing the letters of a `String`:

```
var firstLetter = s[0];  // doesn't work in IE
var lastLetter = s.charAt(s.length - 1);
```
JS

for loop (same as Java) (7.2.8)

```
for (initialization; condition; update) {
  statements;
}
```
JS

```
var sum = 0;
for (var i = 0; i < 100; i++) {
  sum = sum + i;
}
```
JS

```
var s1 = "hello";
var s2 = "";
for (var i = 0; i < s.length; i++) {
  s2 += s1.charAt(i) + s1.charAt(i);
}
// s2 stores "hheellllloo"
```
JS

Math object (7.2.9)

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

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

Special values: `null` and `undefined` (7.2.10)

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

// at this point in the code,
//   ned is null
//   benson's 9
//   caroline is undefined
```
JS

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

7.3: Program Logic

- 7.1: Key JavaScript Concepts
- 7.2: JavaScript Syntax
- **7.3: Program Logic**
- 7.4: Advanced JavaScript Syntax

Logical operators (7.3.1, 7.3.4)

- `> < >= <= && || ! == != === !==`
- most logical operators automatically convert types:
  - `5 < "7"` is `true`
  - `42 == 42.0` is `true`
  - `"5.0" == 5` is `true`
- `===` and `!==` are strict equality tests; checks both type and value
  - `"5.0" === 5` is `false`

`if/else` statement (same as Java) (7.3.2)

```
if (condition) {
  statements;
} else if (condition) {
  statements;
} else {
  statements;
}
```
JS

- identical structure to Java's `if/else` statement
- JavaScript allows almost anything as a *condition*

Boolean type (7.3.3)

```
var iLike190M = true;
var ieIsGood = "IE6" > 0;   // false
if ("web dev is great") {  /* true */ }
if (0) {  /* false */ }
```
JS

- any value can be used as a `Boolean`
  - "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);`

while loops (same as Java) (7.3.5)

```
while (condition) {
  statements;
}
```
JS

```
do {
  statements;
} while (condition);
```
JS

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

7.4: Advanced JavaScript Syntax

Scope, global and local variables (7.4.1)

```
// global code; like "main"
var count = 1;
f2();
f1();

function f1() {
  var x = 999;
  count = count * 10;
}
function f2() { count++; }
```

JS

- variable `count` above is **global** (can be seen by all functions)
- variable `x` above is **local** (can be seen by only `f1`)
- both `f1` and `f2` can use and modify `count` (what is its value?)

Function parameters/return (7.4.3)

```
function name(parameterName, ..., parameterName) {
  statements;
  return expression;
}
```
JS

```
function quadratic(a, b, c) {
  return -b + Math.sqrt(b * b - 4 * a * c) / (2 * a);
}
```
JS

- parameter/return types are not written
  - `var` is *not* written on parameter declarations
  - functions with no `return` statement return `undefined`
- any variables declared in the function are local (exist only in that function)

Calling functions (same as Java)

```
name(parameterValue, ..., parameterValue);
```
JS

```
var root = quadratic(1, -3, 2);
```
JS

- if the wrong number of parameters are passed:
  - too many? extra ones are ignored
  - too few? remaining ones are given `undefined` value

Common bug: spelling error

```
function foo() {
  Bar();   // capitalized wrong
...
function bar() {
  ...
}
```
JS

- if you misspell an identifier, the value `undefined` is used
- if you set `undefined` as an event handler, nothing happens (fails silently)
- **symptom**: function doesn't get called, or a value is unexpectedly `undefined`
- **fix**: JSLint warns you if you use an undeclared identifier

Arrays (7.4.2)

```
var name = [];                          // empty array
var name = [value, value, ..., value];  // pre-filled
name[index] = value;                    // store element
```
JS

```
var ducks = ["Huey", "Dewey", "Louie"];

var stooges = [];         // stooges.length is 0
stooges[0] = "Larry";     // stooges.length is 1
stooges[1] = "Moe";       // stooges.length is 2
stooges[4] = "Curly";     // stooges.length is 5
stooges[4] = "Shemp";     // stooges.length is 5
```
JS

- two ways to initialize an array
- `length` property (grows as needed when elements are added)

Array methods

```
var a = ["Stef", "Jason"];    // Stef, Jason
a.push("Brian");              // Stef, Jason, Brian
a.unshift("Kelly");           // Kelly, Stef, Jason, Brian
a.pop();                      // Kelly, Stef, Jason
a.shift();                    // Stef, Jason
a.sort();                     // Jason, Stef
```
JS

- array serves as many data structures: list, queue, stack, ...
- methods: `concat`, `join`, `pop`, `push`, `reverse`, `shift`, `slice`, `sort`, `splice`, `toString`, `unshift`
  - `push` and `pop` add / remove from back
  - `unshift` and `shift` add / remove from front
  - `shift` and `pop` return the element that is removed

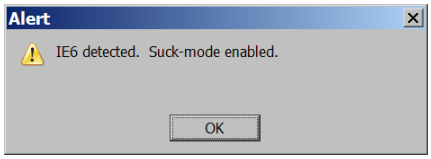
Splitting strings: `split` and `join`

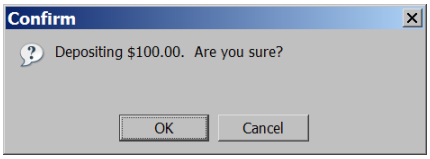
```
var s = "the quick brown fox";
var a = s.split(" ");          // ["the", "quick", "brown", "fox"]
a.reverse();                   // ["fox", "brown", "quick", "the"]
s = a.join("!");               // "fox!brown!quick!the"
```
JS

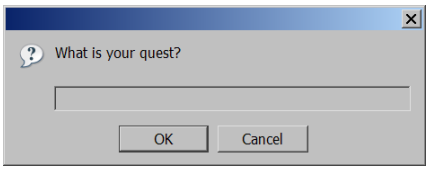
- `split` breaks apart a string into an array using a delimiter
  - can also be used with **regular expressions** (seen later)
- `join` merges an array into a single string, placing a delimiter between them

Popup boxes (7.4.4)

```
alert("message");      // message
confirm("message");    // returns true or false
prompt("message");     // returns user input string
```
JS







Extra random JavaScript stuff

- 7.1: Key JavaScript Concepts
- 7.2: JavaScript Syntax
- 7.3: Program Logic
- 7.4: Advanced JavaScript Syntax
- **Extra random JavaScript stuff**

JavaScript in HTML `body` (example)

```HTML
<script type="text/javascript">
  JavaScript code
</script>
```

- JS code can be embedded within your HTML page's `head` or `body`
- runs as the page is loading
- this is considered *bad style* and shouldn't be done in this course
  - mixes HTML content and JS scripts (bad)
  - can cause your page not to validate

The `typeof` function

```JS
typeof(value)
```

- given these declarations:
  - `function foo() { alert("Hello"); }`
  - `var a = ["Huey", "Dewey", "Louie"];`

- The following statements are `true`:
  - `typeof(3.14) === "number"`
  - `typeof("hello") === "string"`
  - `typeof(true) === "boolean"`
  - `typeof(foo) === "function"`
  - `typeof(a) === "object"`
  - `typeof(null) === "object"`
  - `typeof(undefined) === "undefined"`

The `arguments` array

```
function example() {
  for (var i = 0; i < arguments.length; i++) {
    alert(arguments[i]);
  }
}

example("how", "are", "you");   // alerts 3 times
```
JS

- every function contains an array named `arguments` representing the parameters passed
- can loop over them, print/alert them, etc.
- allows you to write functions that accept varying numbers of parameters

The "for each" loop

```
for (var name in arrayOrObject) {
  do something with arrayOrObject[name];
}
```
JS

- loops over every index of the array, or every property name of the object
- using this is actually discouraged, for reasons we'll see later

Associative arrays / maps

```
var map = [];
map[42] = "the answer";
map[3.14] = "pi";
map["champ"] = "suns";
```
JS

- the indexes of a JS array need not be integers
- this allows you to store *mappings* between an index of any type ("keys") and value
- similar to Java's `Map` collection or PHP's associative arrays

Date object

```
var today = new Date();                // today
var midterm = new Date(2007, 4, 4);    // May 4, 2007
```
JS

- methods
  - `getDate`, `getDay`, `getMonth`, `getFullYear`, `getHours`, `getMinutes`, `getSeconds`, `getMilliseconds`, `getTime`, `getTimezoneOffset`, `parse`, `setDate`, `setMonth`, `setFullYear`, `setHours`, `setMinutes`, `setSeconds`, `setMilliseconds`, `setTime`, `toString`
- quirks
  - `getYear` returns a 2-digit year; use `getFullYear` instead
  - `getDay` returns day of week from 0 (Sun) through 6 (Sat)
  - `getDate` returns day of month from 1 to (# of days in month)
  - `Date` stores month from 0-11 (not from 1-12)

Injecting Dynamic Text: `document.write`

```
document.write("message");
```
JS

- prints specified text into the HTML page
- this is very bad style; this is how newbs program JavaScript:
  - putting JS code in the HTML file's `body`
  - having that code use `document.write`
  - (this is awful style and a poor substitute for server-side PHP programming)

The `eval` (evil?) function

```
eval("JavaScript code");
```
JS

```
eval("var x = 7; x++; alert(x / 2);");  // alerts 4
```
JS

- `eval` treats a String as JavaScript code and runs that code
- this is occasionally useful, but usually a very *bad idea*
  - strings from user input can cause arbitrary code execution
  - leads to bugs and security problems; do not use