Introduction to JavaScript

CSE 190 M (Web Programming) Spring 2008
University of Washington

Reading: Chapter 3 sections 3.2 - 3.3

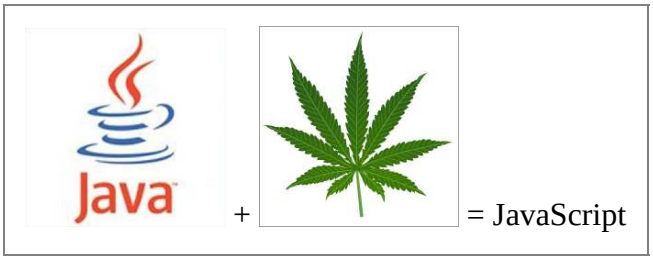
Special thanks to Hélène Martin for creating the 2007 version of these slides and presenting this material to the students.

What is JavaScript?

- 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

Differences between JavaScript and 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)
- contained within a web page and integrates with its HTML/CSS content



Linking to a JavaScript file (example)

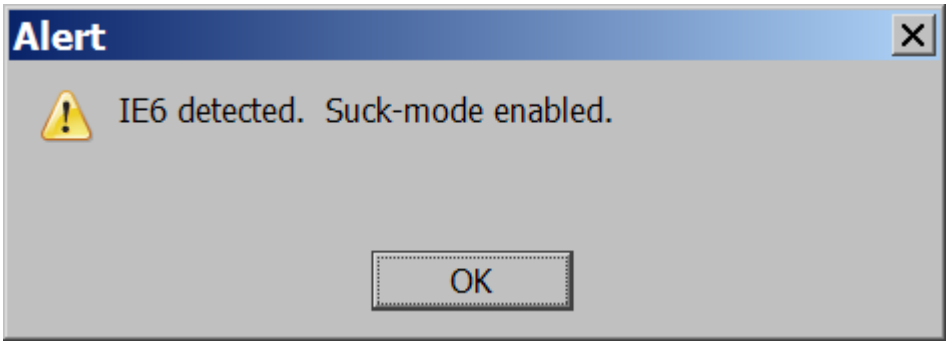
```HTML
<script src="filename" type="text/javascript"></script>
```

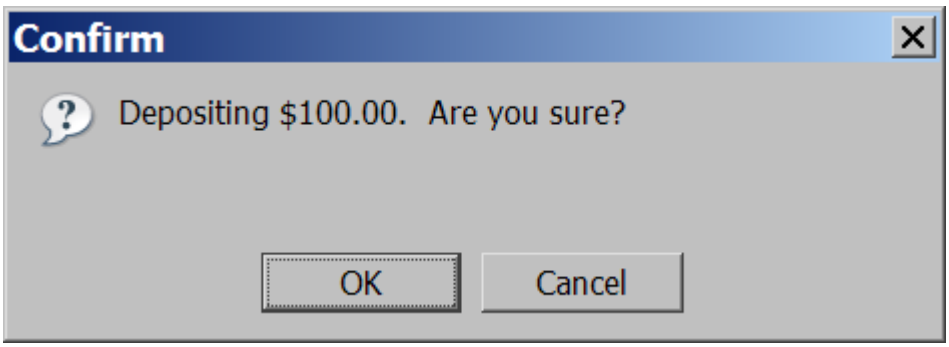
```
<script src="example.js" type="text/javascript"></script>
```

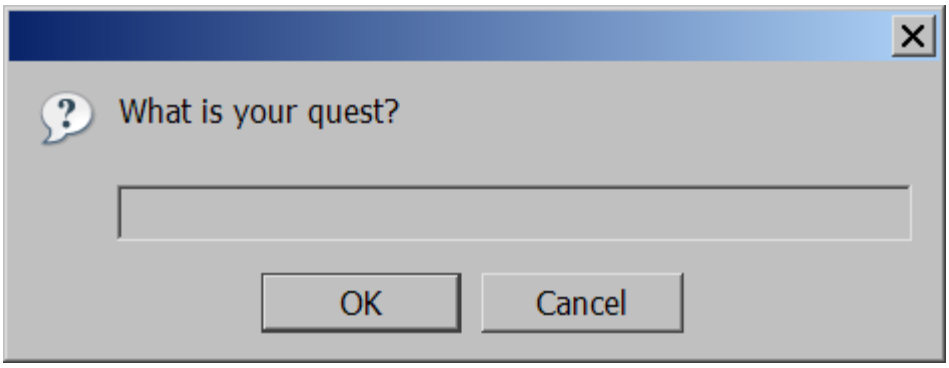
- should be placed in XHTML page's `head`
- script code is stored in a separate `.js` file

Popup boxes

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







JavaScript keywords

```
abstract boolean break byte case catch char class const continue debugger
default delete do double else enum export extends false final finally float
for function goto if implements import in instanceof int interface long
native new null package private protected public return short static super
switch synchronized this throw throws transient true try typeof var void
volatile while with
```

- new ones we'll use most: `var`, `function`

Variables and types

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

```
var clientName = "Connie Client";
var age = 32;
var weight = 137.4;
```

- declaring variables:
  - explicitly, by `var` keyword (case sensitive)
  - implicitly, by assignment (give it a value, and it exists!)
- types: not specified, but JS does have types ("loosely typed")
  - common types: Number, Boolean, String, Null, Undefined
  - can find out a variable's type by calling `typeof`

Number type

```
var enrollment = 99;
var median142Grade = 2.8;
```
JS

- integers and real numbers are the same type (no `int` vs. `double`)
- converting a `String` into a `Number`

```
var name = parseInt("String");
var name = parseFloat("String");
```
JS

  - `parseInt("123hello")` returns `123`
  - `parseInt("booyah")` returns `NaN` (not a number)

Operators and expressions

```
var credits = 5 + 4 + (2 * 3);
```
JS

- operators:
  `+ - * / % ++ -- = += -= *= /= %=`
  `> < >= <= && || ! == != === !==`
  - `==` just checks value (`"5.0" == 5` is `true`)
  - `===` also checks type (`"5" === 5` is `false`)
- similar precedence to Java
- many operators auto-convert: `"2" * 3` is `6`, `5 < "7"` is `true`

Comments (same as Java)

```
// single-line comment

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

- identical to Java's comment syntax
- recall: 3 comment syntaxes
  - XHTML: `<!-- comment -->`
  - CSS/JS: `/* comment */`
  - JS: `// comment`

`for` loop (same as Java)

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

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

- performs a cumulative sum
- minor difference: variables declared still exist after the loop
  - in JS, there is only global scope and function scope

`String` type

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

- methods: `charAt`, `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 `' '` (useful later)

More about `String`

- escape sequences behave as in Java
  - `\' \" \& \n \t \\`
- converting a number to a `String`
  - `var s1 = String(myNum);`
  - `var s2 = count + " bananas, ah ah ah!";`
- accessing the letters of a `String`
  - `var firstLetter = s[0];`
  - `var lastLetter = s.charAt(s.length - 1);`
  - (indexing using array-like syntax doesn't work in IE6)

Math object

```
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

if/else statement (same as Java)

```
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*
  (see next slide)

Boolean type

```
var iLike190M = true;
var ieIsGood = "IE6" > 0;   // false
```
JS

- any value can be used as a Boolean
  - `if ("Marty is great") {   // true, of course!`
  - "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` loop (same as Java)

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

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

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

Defining functions

```
function name(parameterName, ..., parameterName) {
  statements;
}
```
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 an `undefined` value
- 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 get an `undefined` value

Scope: global and local variables

```
var count = 1;

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

f2();   // this is "main"
f1();
```
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?)

Special values: `undefined` and `null`

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

// at this point in the code,
//   ned is null
//   benson is 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?

Arrays

```
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[9] = "Curly";     // stooges.length is 10
```
JS

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

Array methods

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

- 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`

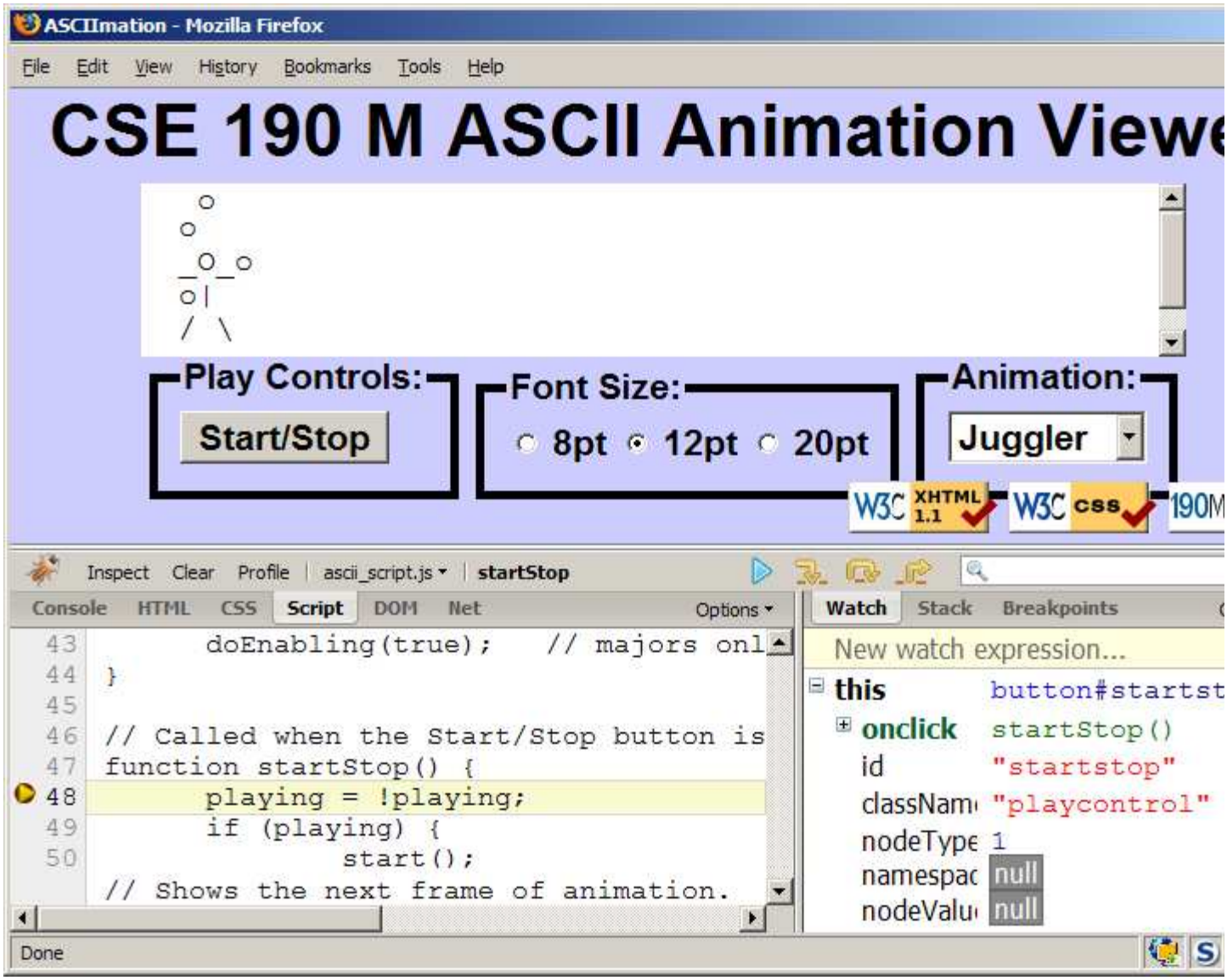
```
var s = "the quick brown fox";
var a = s.split(" ");          // [the,quick,brown,fox]
a.reverse();                   // [fox,brown,quick,the]
s = a.join("!");               // "fox!brown!quick!the"

for (var i = 0; i < a.length; i++) {
  alert(a[i]);
}
```
JS

- `split` breaks apart a string into an array using a delimiter
- `join` merges an array of strings into a single string, placing the delimiter between them

Common JavaScript errors

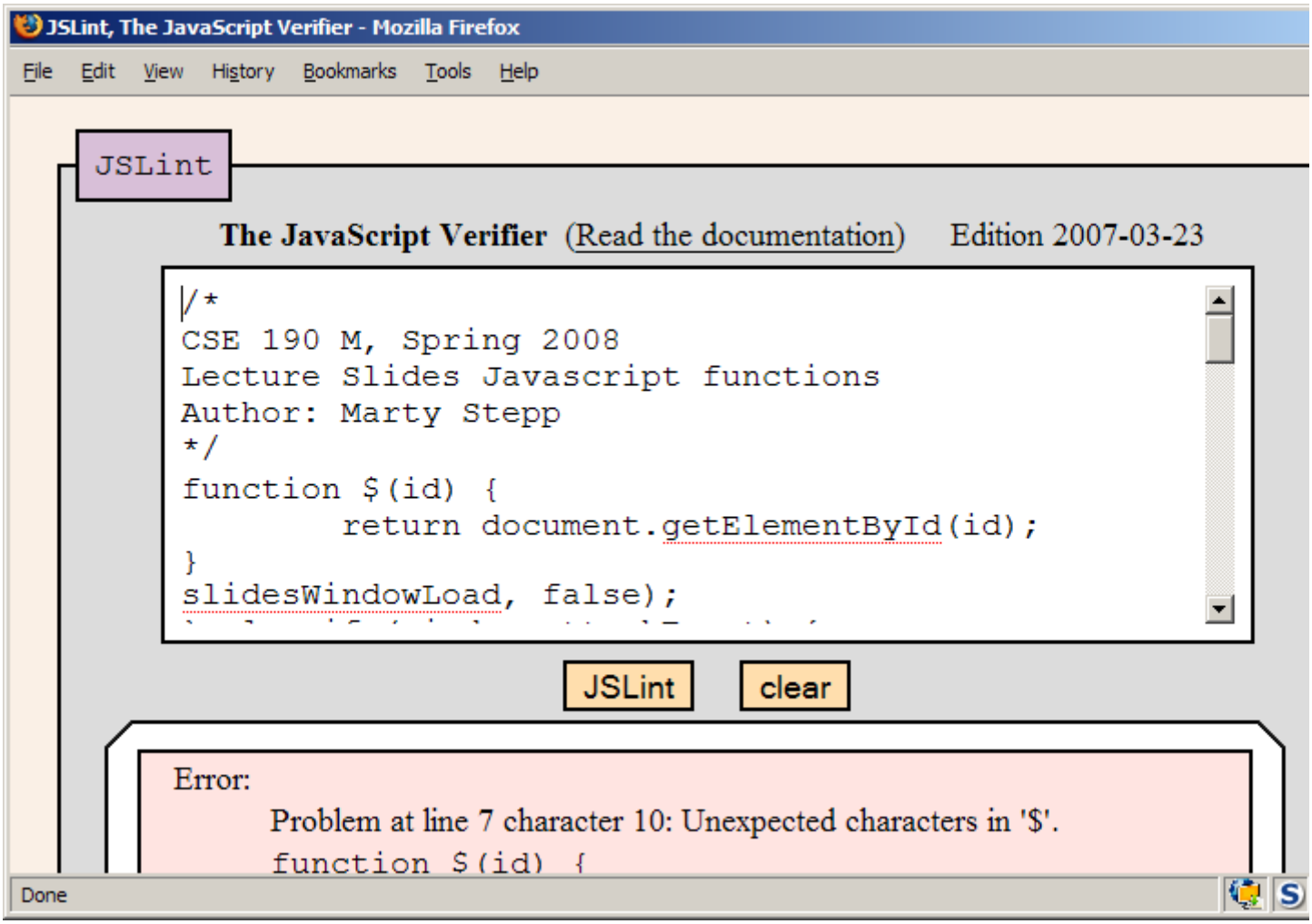
- most common student errors:
  - "My program does nothing." (most errors produce no output)
  - "It just prints `undefined`." (many typos lead to undefined variables)
- quick debugging tips:
  - Are you sure the browser is even loading your JS file at all?
    Put an `alert` at the top of it and make sure it appears.
  - When you change your code, do a **full browser refresh (Shift-Ctrl-R)**
  - Check the bottom-right corner of Firefox for syntax errors.
  - Paste your code into our JSLint tool to find problems.
  - Type some test code into Firebug's console or use a breakpoint.

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 enhanced version
  - original version, by Douglas Crockford
- when your JS code doesn't work, paste it into JSLint first to find many common problems