Javascript

CSE 190 M (Web Programming) Spring 2007
University of Washington

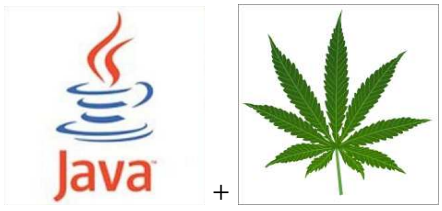
Reading: Sebesta Ch. 4 sections 4.1 - 4.6.4, 4.8, 4.9.1 - 4.9.3, 4.14

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



Injecting Dynamic Text: `document.write()`

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

- prints specified text to page
- can be used to display HTML
- argument can be a literal string in quotes or a variable

Variables

```
var name = value;
```

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

- type is not specified, but Javascript does have types
  - (a "loosely typed" language)
  - values are often converted between types automatically as needed
- variable names are case sensitive
- explicitly declared using `var` keyword
- implicitly declared through assignment (give it a value and it exists!)

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
```

Operators

- `+ - * / % ++ -- = += -= *=`
  `/= %= == != > < >= <= && || !`

- `==` just checks value (`"5.0" == 5` is `true`)
- `===` also checks type (`"5" === 5` is `false`)
- many operators auto-convert types: `5 < "7"` is `true`
- similar precedence hierarchy to Java

`for` loop

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

```
for (var i = 0; i < 10; i++) {
    document.write("<p>" + i + " squared = " +
                   (i * i) + "</p>");
}
```

Inserting Javascript in HTML

- Javascript code can be added to a web page in three ways:
  1. in the page's `body` (runs when page loads)
  2. in the page's `head` (runs when events occur)
  3. in a link to an external `.js` script file

Javascript in HTML body (example)

```
<body>
    ...
    <script type="text/javascript">
    Javascript code
    </script>
    ...
</body>
```

- always runs on page load
- useful for generating dynamic text

Practice problem: Hello World!

1. Write a page that displays "Hello World!" using Javascript.
2. Make "Hello World!" appear 1000 times.
3. Make it so there's only one "Hello World!" per line.

Javascript in HTML head (example)

```
<head>
    ...
    <script type="text/javascript">
    Javascript code
    </script>
    ...
</head>
```

- does not run unless functions are explicitly called
- useful for event-triggered actions
  - pop up an alert message when a user clicks a given element
  - display a greeting message on refresh

Linking to a Javascript file (example)

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

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

- can be placed in page's `head` or `body`
- script is stored in a `.js` file
- the preferred way to write scripts for this course

`String` type

```
var clientName = "Connie Client";
```

- methods
  - `charAt`, `indexOf`, `lastIndexOf`, `replace`, `split`, `substring`, `toLowerCase`, `toUpperCase`
  - `var fName = s.substring(0, s.indexOf(" "));`
  - the `charAt` method returns a value of type `String` (there is no `char` type in Javascript)
- `length` property
  - `clientName.length` is `13`
- can be specified with `" "` or `' '` (this is useful later)

More about `String`

- escape sequences behave as in Java
  - `\' \" \& \n \t \\`
- converting a number to a `String`
  - `var s = String(myNum);`
  - `var s = count + " bananas, ah ah ah!"`
  - many other operators such as `<` automatically convert

Number type

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

- integers and real numbers are the same type
  - stored as 64-bit floating point
- converting a `String` into a `Number`
  - `var integerValue = parseInt("String");`
  - `var floatValue = parseFloat("String");`
  - `parseInt("123hello")` returns `123`
  - `parseInt("booyah")` returns `NaN` (not a number)

`if/else` statement

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

- identical structure to Java's `if/else` statement
- Javascript is more forgiving about what it allows as a condition
  (see `Boolean` type on next slide)

Boolean type

```
var iLike190M = true;
```

- any value can be used as a `Boolean`
  - ```
    if ("Marty is great") {   // true, of course!
        ...
    }
    ```
  - `0`, `NaN`, `""`, `null`, and `undefined` are all `false`
  - all else are `true`
- converting a value into a `Boolean` explicitly
  - `var boolValue = Boolean(otherValue);`
    ```

while loop

```
while (condition) {
    statements;
}

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

- break and continue keywords also behave as in Java

Math object

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

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

Comments

```
// single-line comment

/*
multi-line comment
*/
```

- identical to Java's comment syntax

Practice problem: Random image

A "Person of the Day" page has been made for this class. Today's winner is Marty, but you have two favorite pictures of him!

Randomly choose between these two pictures on each page refresh.

Functions

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

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

- parameter types and 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 (only exist in that function)

Calling functions

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

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

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

Global and local variables

```
var count = 1;

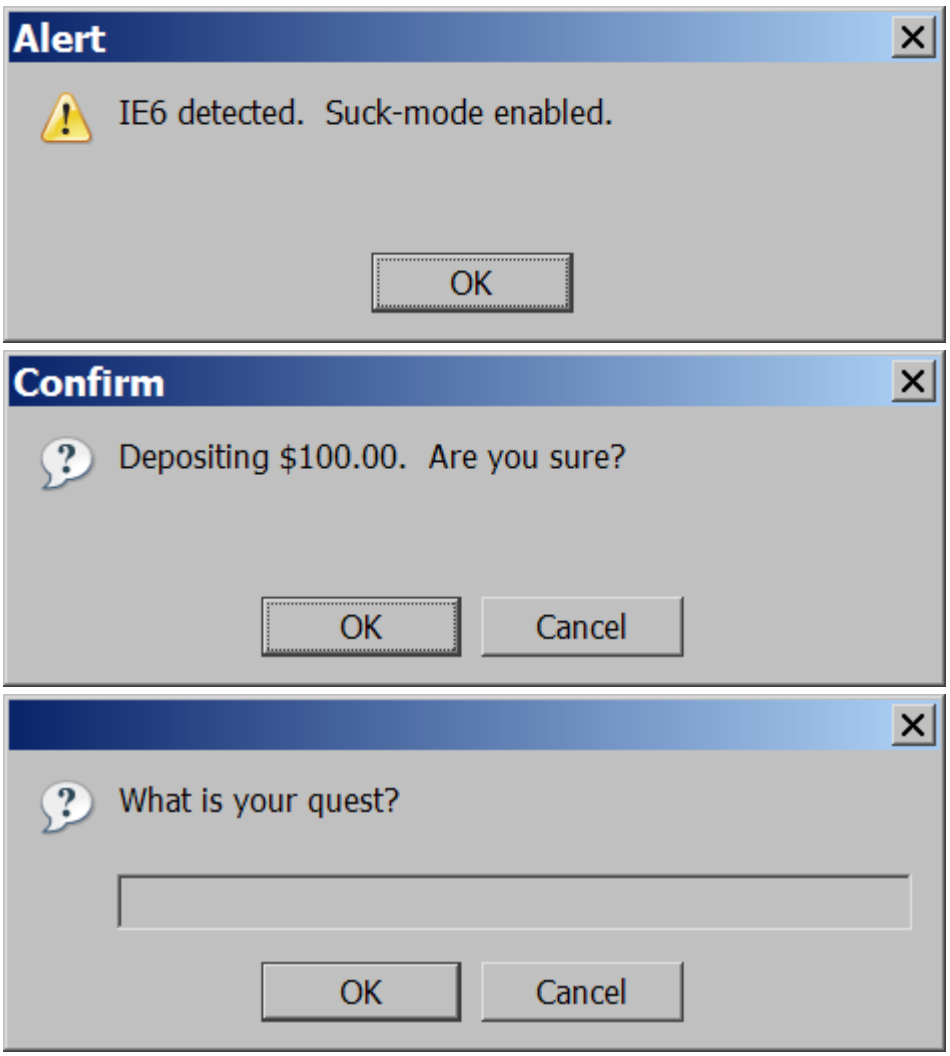
function f1() {
    var x = 999;
    count *= 10;
}
function f2() { count++; }

f2();
f1();
```

- 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?)

Popup boxes

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



Date object

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

- 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)

Event handlers

```
<h2 onclick="myFunction();">Click me!</h2>
```

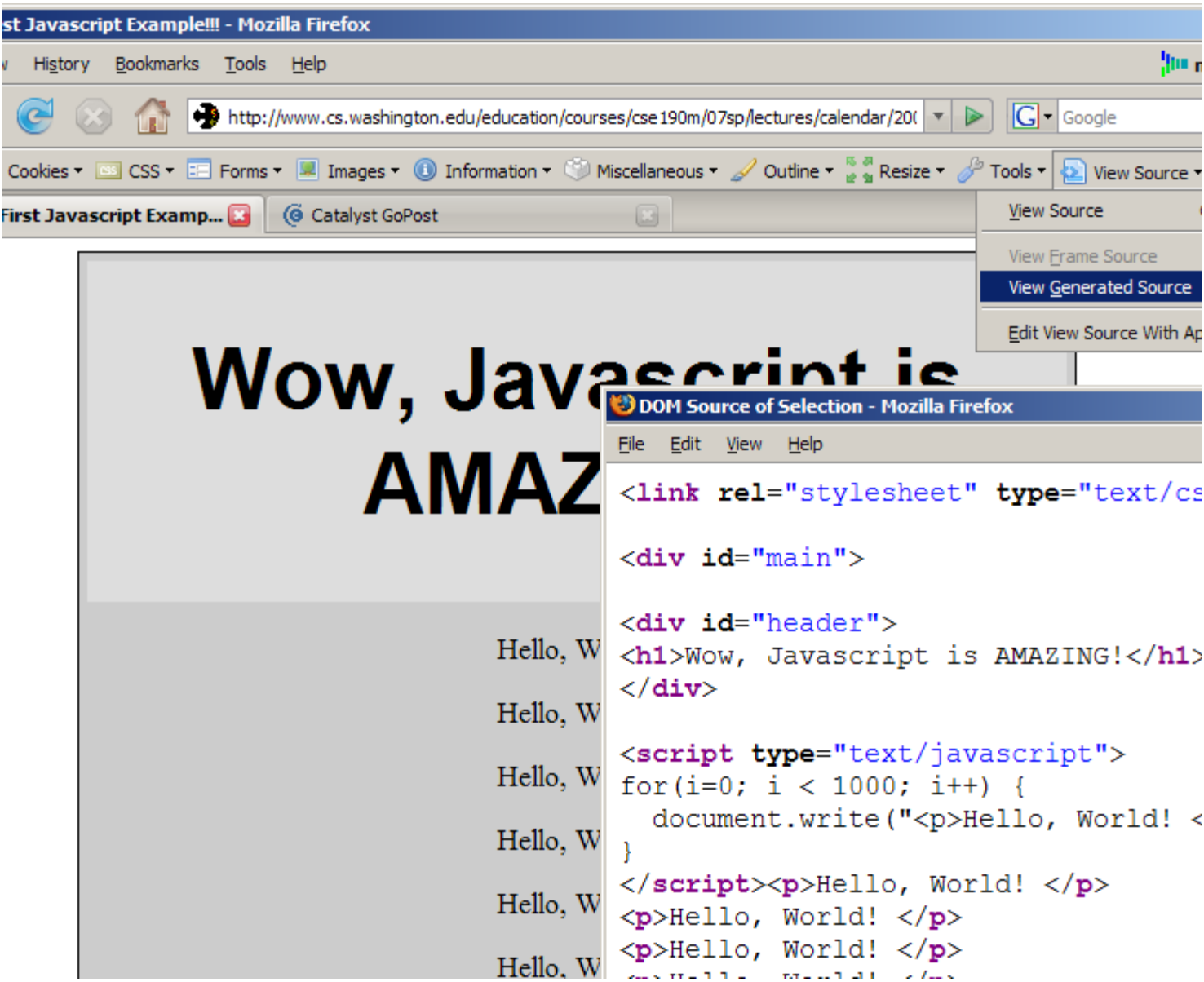
Click me!

- HTML elements have special attributes called events
- Javascript functions can be set as event handlers
  - when you interact with the element, the function will execute
  - an example of event-driven programming
- onclick is just one of many event HTML attributes we'll see later (complete list)

Practice problem: Birthdays

1. Modify the "Person of the Day" page so that the number of days until Marty's birthday are displayed in an alert box when his picture is clicked.
2. When the previous people of the day's names are clicked the number of days until their birthday should be displayed.

Firefox Web Developer extension



Arrays

```
var stooges = new Array();
stooges[0] = "Larry";
stooges[1] = "Moe";
stooges[2] = "Curly";

var stooges = new Array("Larry", "Moe", "Curly");

var stooges = ["Larry", "Moe", "Curly"];
```

- three ways to initialize an array

Array methods

- methods:
  - concat, join, pop, push, reverse, shift, slice, sort, splice, toString, unshift
- properties:
  - length

Arrays as lists

```
var a = new Array();
a.push("Morgan");           // Brian
a.push("Brian");            // Morgan,Brian
a.unshift("Kenneth");       // Kenneth,Morgan,Brian
a.push("Helene", "Jeff");   // Kenneth,Morgan,Brian,Helene,Jeff
a.shift();                  // Morgan,Brian,Helene,Jeff
a.pop();                    // Morgan,Brian,Helene
a.sort();                   // Brian,Helene,Morgan
```

- `push` and `pop` add and remove from end of array
- `unshift` and `shift` add and remove from front of array
- `shift` and `pop` return the element that is removed

Strings and arrays: `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"
```

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

Special values: `undefined` and `null`

```
var ned;
var benson = 9;

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

- `undefined` : has not been declared
- `null` : has been declared but not assigned a value

The `typeof` function

```
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"`

Timers: `setTimeout`, `clearTimeout`

```
function delayedMessage() {
    var myTimer = setTimeout("alert('Booyah!');", 5000);
}

<h2 onclick="delayedMessage();">Click me now!</h2>
```

Click me now!

- `setTimeout` executes a piece of code once after a given number of milliseconds
- the function returns an object representing the timer
- to cancel the timer, call `clearTimeout` and pass the timer object
  - `clearTimeout(myTimer);  // cancel self-destruct sequence!`

setInterval, clearInterval

```
function repeatedMessage() {
    var myTimer = setInterval("alert('Rudy!');", 1000);
}

<h2 onclick="repeatedMessage();">Click me now!</h2>
```

Click me now!

- `setInterval` executes a piece of code *repeatedly*, every given number of milliseconds
- the function returns an object representing the timer
- to cancel the timer, call `clearInterval` and pass the timer object
    - `clearInterval(myTimer);  // please make it stop!`

Common bug: local variable in timer

```
function delayed(text) {
    var myTimer = setTimeout("alert(text);", 1000);
}

<h2 onclick="delayed('hello');">Click me now!</h2>
```

Click me now!

- `setTimeout` and `setInterval` will execute later, after your function is done running
- any local variables in your function will be gone by this time
- therefore, make any variables needed by the timer code global

Practice problem: Stop the timer

- Modify the preceding Rudy timer example (HTML, JS) so that it has another heading that can be clicked to stop the annoying timer.

----- More Arrays and Objects -----

More Arrays and Objects

The `arguments` array

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

example("how", "are", "you");
```

- 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 take varying numbers of parameters

Arrays as maps

```
var map = new Array();
map[42] = "the answer";
map[3.14] = "pi";
map["champ"] = "suns";
```

- 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 a hash table data structure

Array map example

```
...
```

The "for each" loop

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

- loops over every index of the array, or every property name of the object