JavaScript

CSC 337, Fall 2013
The University of Arizona
William H. Mitchell
whm@cs

Review: The Big Picture

The big picture as we know it

Here are the discrete entities that we're currently working with:

- Browser
- HTTP Server
- PHP system
- MySQL database server

Let's review their responsibilities.

Note: Asking what does what is a great exam question!

The big picture, continued

Browser:

When a URL is entered in the address field it sends a GET request to the HTTP server on the specified host.

Renders the HTML sent in the HTTP server's response.

Generates a GET request for clicked hyperlinks and renders the HTML in the response.

Collects data entered in forms and generates GET/POST request when appropriate, and renders the HTML in the response.

Sends Cookie: header in requests based on site; cookies are set based on Set-Cookie: headers in responses.

The big picture, continued

HTTP Server (like XAMPP's or cgi.cs.arizona.edu's Apache):

It is a program running on a machine that is also called a server.

Listens for and accepts http/https connections from processes via the Internet.

Responds to requests like GET and POST.

If a request hits a .php file, asks PHP system to run the file, passing GET/POST data, cookies, etc., and returns PHP output as the response.

The big picture, continued

PHP system:

It is a program that is often on the same machine as the HTTP server.

Populates $_GET, $_POST, $_COOKIE, $_SESSION and more based on data in the request relayed by the HTTP server.

Executes PHP code, possibly reading and/or writing files on the server, possibly interacting with databases and other services.

Collects standard output and returns it to the HTTP server.

The big picture, continued

MySQL database server

A program that runs on possibly the same machine as the HTTP server and the PHP system.

Maintains some number of databases, each of which contain some number of tables with rows of data.

Listens for and accepts network connections from PHP programs.

Executes SQL statements sent via connections, makes changes in tables, and returns any resulting data.

The addition: JavaScript

HTML is not a programming language.

JavaScript is a general purpose programming language.

In addition to rendering HTML (with appearance controlled by CSS) browsers can execute code written in JavaScript.

JavaScript code can respond to events like clicks.

JavaScript code can access and modify all document elements and associated styling information.

What should run where?

Almost everything we've written in PHP thus far could have been written in JavaScript instead.

Two notable exceptions:

- JavaScript can't read/write files, so blog.txt is a problem.
- We can't secure Super KLKR with only JavaScript.

Speculate: What are the advantages and disadvantages of having code run in the browser (the "front-end") rather than on a distant server (the "back-end")?

A Little JavaScript History

Developed by Brendan Eich at Netscape in 1995. Briefly called LiveScript.

Netscape "...wanted a lightweight interpreted language that would complement Java by appealing to nonprofessional programmers, like Microsoft's Visual Basic."—Wikipedia

Was considered a hobbyist's language and not respected for a long time. (In fact, it's deep and interesting!)

The associated standard is ECMAScript (ECMA-262) but the situation is murky. "Edition 5" is widely supported but Edition 3 is perhaps closest to what many view as "JavaScript".

Oracle owns "JavaScript" trademark. Wikipedia says, "'JavaScript' is an ECMAScript variant managed by Mozilla."

Firefox's JavaScript engine is SpiderMonkey. Their latest version of JavaScript is called 1.8.5. Chrome's JavaScript engine is "V8".

JavaScript Resources

JavaScript: The Good Parts by Douglas Crockford

Maybe the best book that focuses on the language itself.

JavaScript: The Definitive Guide, 6th ed. by David Flannagan

1100 pages. Lots of details you'll find nowhere else.

JavaScript and jQuery: The Missing Manual, 2nd ed. by McFarland

Some good JavaScript basics but mostly focused on jQuery

Learning PHP, MySQL, JavaScript, and CSS, 2nd ed. by Nixon

Chapter 13 and following

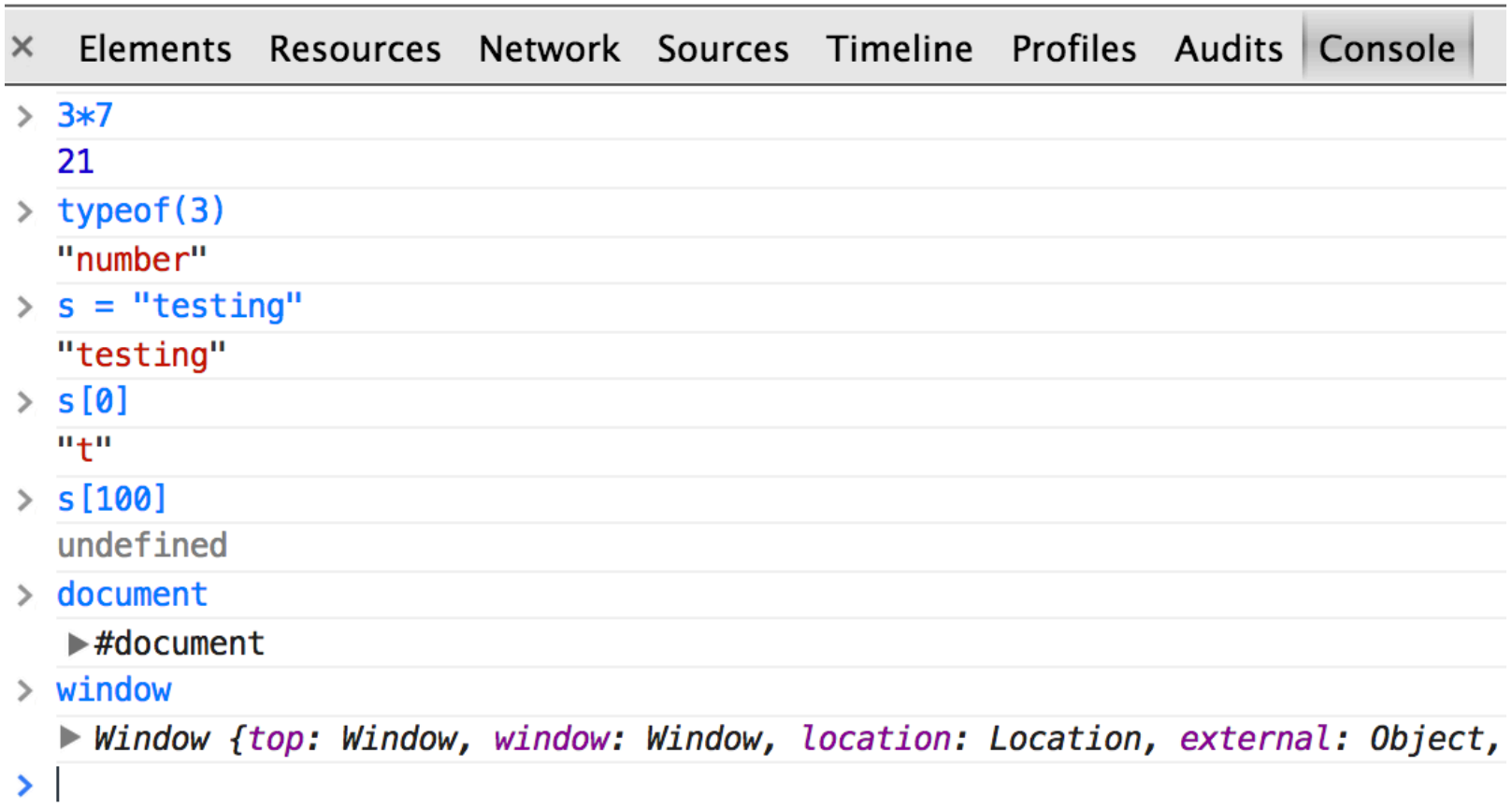
Perhaps the best on-line reference I've found:

developer.mozilla.org/en-US/docs/Web/JavaScript/Reference

JavaScript basics

Evaluating JavaScript expressions

The Chrome DevTools Console tab provides a JavaScript REPL (read-eval-print loop):



We'll use it to explore basic elements of the language.

Datatypes

JavaScript has four scalar datatypes:

- number
- string
- boolean
- undefined

JavaScript has an object type, too; arrays are a special kind of object.

Functions are values, too!

typeof(*expr*) is like PHP's gettype(*expr*) function

Numbers

`number` is the only numeric type. Internally, numbers are IEEE-754 floating-point values but they are shown as integers if the fractional part is zero. Arithmetic operators are conventional.

> **3 + 4**
7

> **3 * 4.1**
12.299999999999999

> **10 / 5**
2

> **5 / 10**
0.5

> **typeof(10/5)**
"number"

> **typeof(5/10)**
"number"

Variables

Variable names can be composed of any number of Unicode letters, digits, underscores, and $, but cannot start with a digit.

Like PHP, variables have no type themselves and need not be declared before use.

```
> x = 5
5

> typeof(x)
"number"

> x = "hello"
"hello"

> typeof(x)
"string"
```

Variables have global scope unless declared with a `var` declaration inside a function.

Strings

The string type represents strings of zero or more characters.

Strings can be enclosed in single or double quotes.

```
> s1 = "testing"
"testing"

> typeof(s1)
"string"

> s2 = "It's ok!"
"It's ok!"
```

A typical set of \ escapes is available.

Like PHP, there is no single-character type such as Java's char.

One place the string type is well-documented is here:
developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Strings, continued

Strings have a length *property*:

```
>  s = "testing"
"testing"

> s.length        // Note: property, not a method!
7
```

Individual characters can be accessed with a subscripting notation.

```
> s[0]
"t"

> s[6]
"g"
```

Assignment to a character produces no error but is ineffective:

```
> s[0] = "X"
"X"

> s
"testing"
```

Strings, continued

The + operator is overloaded, either concatenating strings or adding numbers, depending on the operand types.

```
> 3 + 4
7

> "3" + 4
"34"

> 1 + 2 + "3" + 4
"334"

> 1 + 2 + "3" + 4 * 5
"3320"
```

Unlike PHP, there is no interpolation of variables into strings; concatenation must be used to build strings that combine literals and values of variables: s = "x = " + x + ", y = " + y

Strings, continued

parseInt(s) returns the integer that the string s represents.
String(x) returns a string representation of its argument.

```
> s = "55"
"55"

> x = 20
20

> x + parseInt(s)
75

> String(x) + s
"2055"

> +s + +s              // Any ideas?
110
```

There's parseFloat(s), too.

Strings, continued

strings have a variety of Java-like methods. Here are some:

```
> s = "to be or not to be"
"to be or not to be"

> s.indexOf("or")
6

> u = s.toUpperCase()
"TO BE OR NOT TO BE"

> u.substr(6, 2)
"OR"

> r = u.replace("BE", "(b)")
"TO (b) OR NOT TO BE"
```

boolean values

The boolean type has literals `true` and `false`.

> **typeof(true)**
"boolean"

> **2 < 1**
false

Like PHP, certain non-boolean values are considered be false. All others are true. Here are some of the "falsy" values, demonstrated with the Boolean(x) function:

> **Boolean(0)**
false

> **Boolean("")**
false

> **Boolean("hello")**
true

Comparison operators

The comparison operators don't have the conversion-caused surprises that we saw in PHP.

> **20 < 3**
false

> **"20" < "3"**
true

> **2.1 == "02.1"** // What does this experiment tell you?
true

Like PHP, there are === and !== that compare on both type and value.

> **20 === "20"**
false

> **20 !== "20"**
true

Logical operators

Here are some logical operators in action. Challenge: explain this!

```
> true && false
false

> true && 0
0

> 1 || 2
1

> "" || "x"
"x"

> "0" && false
false

> 0 && false
0
```

There's !x, too.

Arrays

Arrays hold a sequence of any number of values of any type.
An array can be created with a square-bracket literal syntax:

```
> a = [10, "ten", 10.1, false]
[10, "ten", 10.1, false]

> a.length
4

> [a[0], a[2]]
[10, 10.1]

> words = "520-621-6613".split("-")
["520", "621", "6613"]

> words = "520-621-6613".split("").length
12
```

Sidebar: the value `undefined`

If we ask for a value that isn't there, JavaScript typically produces the value `undefined`. `undefined` is a literal, like `true`.

> **a = [1,2,3]; a[10]**
undefined

> **typeof(a[10])**
"undefined"

> **"abc"[3]**
undefined

> **typeof(x)**
"undefined"

> **a = undefined; typeof(a)**
"undefined"

Array methods

The array methods push(), pop(), shift(), and unshift() provide stack- and queue-like operations on arrays.

```
> a = [10, 20, 30]
> a.push(40, 50)          // Note use of multiple values
5

> a.unshift("one", "two", "three")
8

> a
["one", "two", "three", 10, 20, 30, 40, 50]

> a.shift()
"one"

> a.pop()
50

> a
["two", "three", 10, 20, 30, 40]
```

Array methods, continued

The sort() and reverse() methods do an in-place sort and reverse, respectively.

> **vals=[5, -3, "x", "", undefined, "five", true, false]**
[5, -3, "x", "", undefined, "five", true, false]

> **vals.sort()**
["", -3, 5, false, "five", true, "x", undefined]

> **vals**
["", -3, 5, false, "five", true, "x", undefined]

> **vals.reverse()**
[undefined, "x", true, "five", false, 5, -3, ""]

> **vals**
[undefined, "x", true, "five", false, 5, -3, ""]

Array methods, continued

slice() and join() are often handy. Can you explain splice()?

> **a=[10,20,30,40,50,60,70]**

> **a.slice(3,5)**
[40, 50]

> **a.join("<>")**
"10<>20<>30<>40<>50<>60<>70"

> **a.splice(2, 3, "x", "y")**
[30, 40, 50]

> **a**
[10, 20, "x", "y", 60, 70]

More on array methods:
developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/prototype#Methods

Assigning an array assigns a reference!

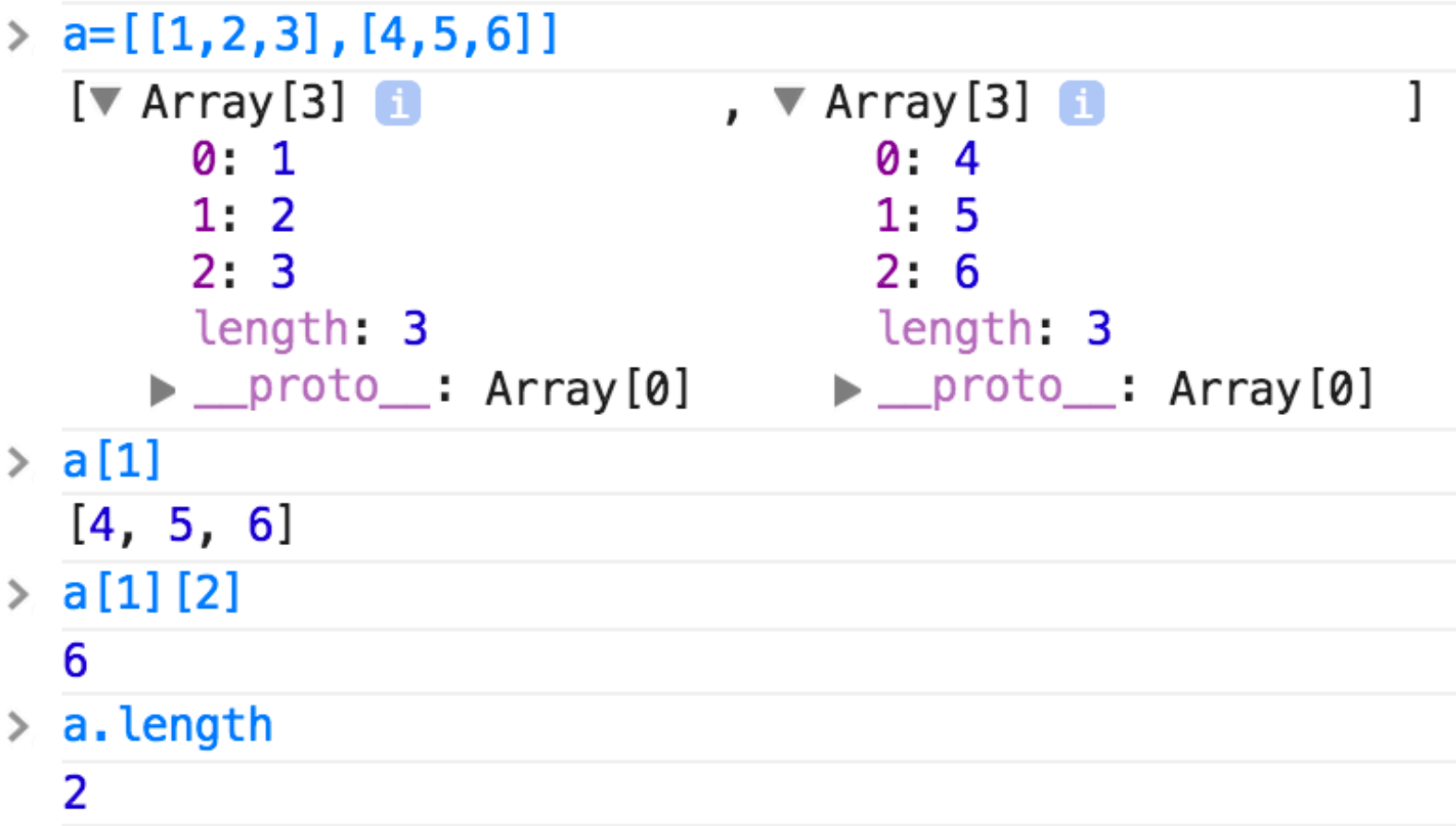
In PHP, assigning an array assigns a copy of the array.

In JavaScript, like Java and Python, **assigning an array assigns a reference to the array**.

```
>  a = [10, 20, 30]
[10, 20, 30]

> b = a
[10, 20, 30]

> b.push("testing")
4

> b
[10, 20, 30, "testing"]

> a
[10, 20, 30, "testing"]
```

Arrays of arrays

Arrays can contain arrays



Objects

A JavaScript object can be created like this:

> **o1 = {a: 10, b: 20}**
Object {a: 10, b: 20}

> **typeof(o1)**
"object"

We can say that the object o1 has two *properties*: a and b. Properties can be referenced with *dot notation* or with a string subscript:

> **o1.a**
10

> **o1["b"]**
20

Objects, continued

If a non-existent property is referenced, undefined is produced

```
> o1 = {a: 10, b:20}
Object {a: 10, b: 20}

> o1.x
undefined

> o1.c
undefined
```

Assigning to a non-existent property creates it:

```
> o1.c = 30
30

> o1["d"] = 40
40

> o1
Object {a: 10, b: 20, c: 30, d: 40}
```

Objects and arrays

Arrays can hold references to objects, and vice-versa.

> **line = [{x: 0, y: 0}, {x: 10, y: 20}]**
[> Object, > Object]

> **line[0]**
Object {x: 0, y: 0}

> **line[1].y**
20

> **o = { x: [12, 16, 20], y: ["blue", "gray"] }**
Object {x: Array[3], y: Array[2]}

> **specs = { sizes: [12, 16, 20], colors: ["blue", "gray"] }**
Object {sizes: Array[3], colors: Array[2]}

> **specs.colors[1]**
"gray"

The value `null`

The value `null` is used to represent the absence of object.
Imagine a simple linked list:

> **node1 = {value: 5, next: null }**
Object {value: 5, next: null}

> **node2 = {value: 3, next: node1 }**
Object {value: 3, next: Object}

> **node2.next.value**
5

> **node2.next.next**
null

The type of null is object:

> **typeof(null)**
"object"

Control structures

`while`, `if/else`, `for`, `break`, and `continue` look and behave like their counterparts in Java, C, and PHP.

Like PHP, any value can be used for the condition in `while`, `if`, and `for` statements.

Sidebar: The DOM

The Document Object Model (DOM) API provides a mechanism for JavaScript code to access an HTML page as a data structure.

Let's hit blog3.php and try these expressions with the Chrome DevTools Console:

```
typeof(document)
document.title
document["title"]
document.head
document.styleSheets
document.body
document.body.children
document.images
document.links
```

Documentation on the DOM API:

https://developer.mozilla.org/en-US/docs/DOM

Sidebar, continued

The DOM API lets us change elements, too.

Changes are immediately reflected on the screen.

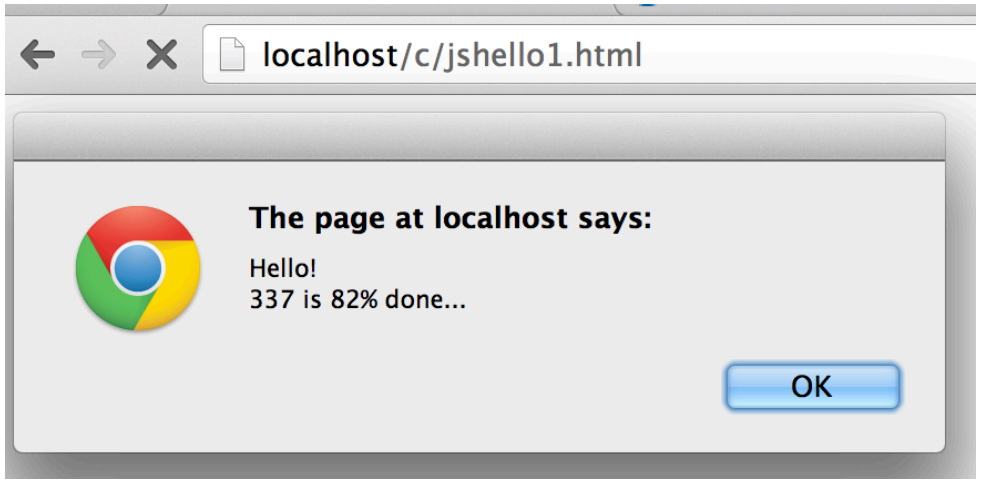
Let's try these expressions:

```
document.body.style.color = "green"
s = document.links[1].style
s.color = "orange"
s.fontSize = 50       (Why not s.font-size?)

links = document.links
for (i = 0; i < links.length; i++) links[i].style.fontSize = i
```

JavaScript on a page

One way to run JavaScript code on a page is to put it inside a `script` element. This is jshello1.html:

```
<!doctype html>
<title>Hello!</title>
<script>
    done = Math.round(36/44*100);
    alert("Hello!\n337 is " + done + "% done...");
</script>
```



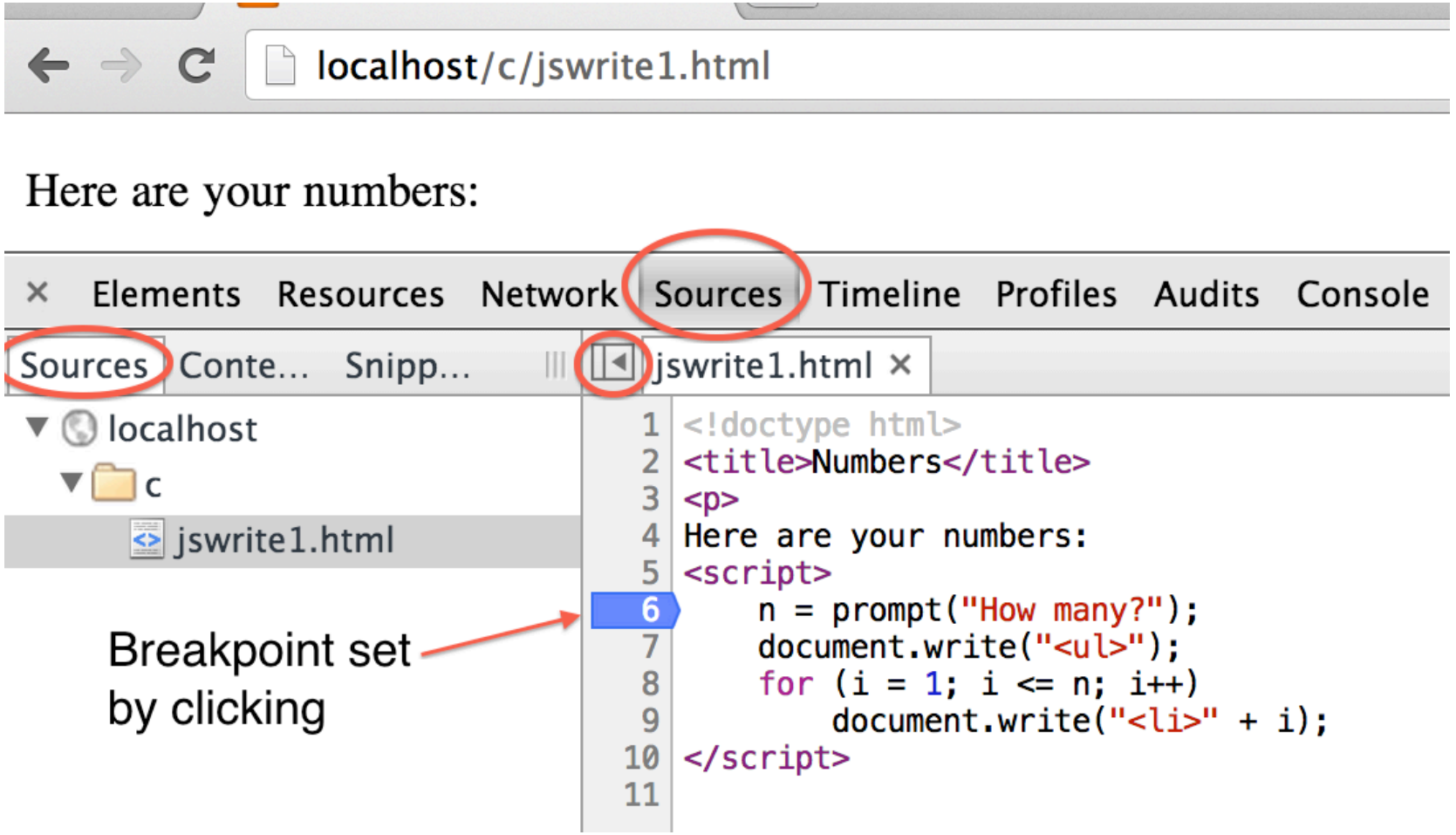
JavaScript on a page, continued

developer.mozilla.org/en-US/docs/Web/API/Document shows that Document instances have a write() method. jswrite1.html uses it. Let's run it!

```
<!doctype html>
<title>Numbers</title>
<p>
Here are your numbers:
<script>
    n = prompt("How many?");
    document.write("<ul>");
    for (i = 1; i <= n; i++)
        document.write("<li>" + i);
</script>
```

Debugging JavaScript

DevTools has a JavaScript debugger:



Refresh the page to run. Use F10, F11, and F8 to step over, step into, and "go".

Errors in JavaScript code

If there are errors in the JavaScript code on a page, a typical result is that nothing happens.

Practice:

Load jserror1.html and find/fix the error.

Repeat for jserror2.html.

One tactic when puzzled: Comment out everything in the script block with /* ... */ and reinstate lines gradually.

Common puzzler: Missing/bogus </script> closing tag.

JavaScript on a page, continued

Claim:

When the browser is parsing a page of HTML, any script elements are run when they are encountered.

Challenge:

Prove it!

~~Functions~~

[skip to slide 53!]

Function basics

Here's a simple JavaScript function definition: (jsfunc1.html)

```
function m_to_n(n, m)
{
    var result = [];
    for (var i = n; i <= m; i++)
        result.push(i);

    return result;
}
```

Notes:

(1) Uses `function` keyword, like PHP.

(2) `var` keyword is used to make `result` and `i` local variables, not globals.

(3) Extra arguments are ignored. Missing arguments have `undefined` as value.

Variable-length argument lists

Inside a function, `arguments` is an array that contains the arguments the function was called with.

This function uses `arguments` to simply write out (to the document!) the arguments the function was called with:

```
function writeargs()
{
   document.write(arguments.length + " arguments:<br>");
   for (var i = 0; i < arguments.length; i++)
      document.write(arguments[i] + "<br>");
}
```

It's in jsfunc2.html. Let's run it.

Problem: Write a function that sums its arguments:
sum(10,20,30) // returns 60 (in jsfuncs2.html)

Functions are values in JavaScript!

In JavaScript, functions are a kind of value, just like string, number, and object.

> **typeof(writeargs)**
"function"

> **typeof(sum)**
"function"

> **f = sum**
function sum() { ....}

> **f(10, 20, 30);**
60

> **f === sum**
true

> **f === writeargs**
false

Functions are values, continued

Since functions are values, we can do things like put them in an array and reference them:

> **a = [writeargs, sum, m_to_n]**

> **a[1](10,20,30)**
60

> **a[2](5,10)**
[5, 6, 7, 8, 9, 10]

> **a[0](1,2,3)**
undefined

> **a.sort()[0](5,10)** (Which did we run?)
[5, 6, 7, 8, 9, 10]

Mystery function

What does this function do? How might we call it?

```
function m(f, a)
{
    var result = [];
    for (var i = 0; i < a.length; i++)
        result.push(f(a[i]));

    return result;
}
```

Hint: Look at the operations being performed on each of the two arguments. What's implied?

Mystery function, continued

The function is widely known as "map". It calls the function `f` with each value in the array `a`, producing an array of the results.

```
function map(f, a)                          // jsmystery.html
{
    var result = [];
    for (var i = 0; i < a.length; i++)
        result.push(f(a[i]));
    return result;
}
```

> **function negate(x) { return -x; }**
> **function zero(x) { return 0; }**

> **map(negate, m_to_n(1, 5));**
[-1, -2, -3, -4, -5]

> **map(zero, m_to_n(1, 5));**
[0, 0, 0, 0, 0]

Functional programming

A function like map can be called a *higher-order function* because it is a function that has a function as an argument.

map is an example of *functional programming*.

JavaScript facilitates functional programming.

Does Java facilitate functional programming?

Does Python facilitate functional programming?

Anonymous functions

Here are two ways to make the **variable** `negate` reference a function that negates its operand:

```
function negate(x) { return -x; }

negate = function (x) { return -x; }
```

The second creates an *anonymous function* and assigns it to negate, a variable.

A common idiom in JavaScript is to use an anonymous function as an argument to a higher-order function:

```
> vals = map(function (x) { return x * 2 - 1; }, [8, 3, -3])
[15, 5, -7]

> map(function (x) { return "Z" + x }, vals)
["Z15", "Z5", "Z-7"]
```

Note: Slides 44-51 are hereby discarded in favor of the following section, but you might still find slides 46 and 49-51 to be interesting.

Functions (v2)

Function basics

Here's a simple JavaScript function definition: (jsfunc1.html)

```
function m_to_n(m, n)
{
    var result = [];
    for (var i = m; i <= n; i++)
        result.push(i);

    return result;
}
```

Notes:

(1) Uses `function` keyword, like PHP.

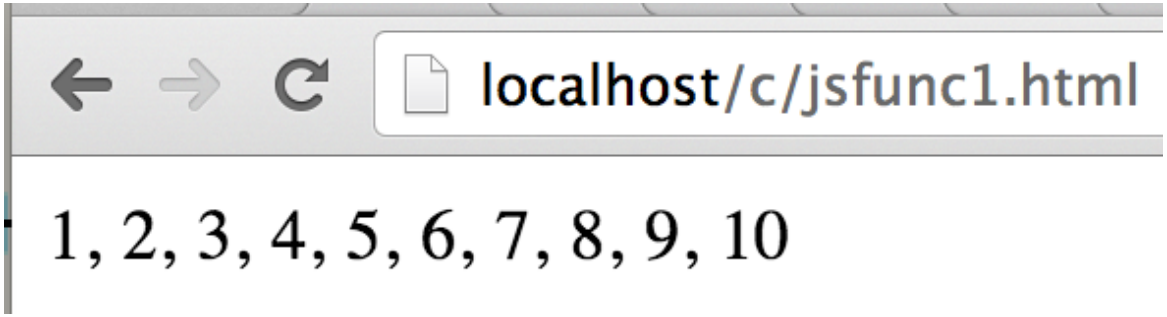
(2) `var` keyword is used to make `result` and `i` local variables, not globals.

(3) Extra arguments are ignored. Missing arguments have `undefined` as value.

Function basics, continued

Usage on a page: (jsfunc1.html)

```
<!doctype html>
<title>Function</title>
<script>
function m_to_n(m, n)
{
    var result = [];
    for (var i = m; i <= n; i++)
        result.push(i);

    return result;
}
document.write(m_to_n(1,10).join(", "));
</script>
```



Functions are values

A JavaScript function definition creates a variable that references a value of type `function`.

That value can be assigned to another variable. That variable can then be used to the call the function.

```
> typeof(m_to_n)
"function"

> seq = m_to_n
[...shows code...]

> seq(1,3)
[1, 2, 3]

> m_to_n === seq
true
```

Anonymous functions

Here's another way to define a function:

```
> double = function (x) { return x + x; }
```

Note that there is no identifier after "function"! This is an *anonymous function*. (Another name: *lambda expression*.)

```
> typeof(double)
"function"

> double(5)
10
```

Explain the following:

```
> (function (x) { return x*7; })(5)
35
```

Properties can hold function values

Functions can be assigned to properties of an object:

```
> ops = {
    negate: function (x) { return -x; },
    double: function (x) { return x+x;}
    }
Object {negate: function, double: function}

> ops.negate(5)
-5

> ops.double(5)
10
```

Array "methods" like sort and push are really just functions assigned to properties!

Functions can be added to objects

We can add functions to objects: (jsaddfunc1.html)

```
> f = function () {
    var r = 0;
    for (var i = 0; i < this.length; i++)
        r += this[i];
    return r; }
> a = [3,4,5]
[3, 4, 5]
> a.sum = f
> a.sum()
12
```

Inside the function, this === a, the object it was invoked on.

If we do Array.prototype.sum = f, every array will have sum!

The jQuery library

The problem with the DOM

It'd be fine to work with directly with the DOM if all browser implementations conformed to a standard.

Sadly, they don't.

jQuery is a JavaScript library that provides an interface to the DOM (and more!) that works with all browsers.

The jQuery library

Written by John Resig. Released in 2006.

Free, and open source.

Widely used. Also embraced by many vendors, including Microsoft and Adobe.

Motto: "write less. do more."

Two major versions:

- 1.X Supports almost everything, back to IE 6.
- 2.X Drops support for IE 6-8, to be smaller and faster.

Site: jquery.com

Using jQuery

jquery.com/download has two flavors:

development/uncompressed (267Kb)

`jquery-1.10.2.js`

Readable source code

production/compressed (a.k.a, "minified") (91Kb)

`jquery-1.10.2.min.js`

Not readable, but small for faster downloading:

```
(function(e,t){var n,r,i=typeof t,o=e.location,
a=e.document,s=a.documentElement,l=e.jQuery
,u=e.$,c={},p=[],f="1.10.2",d=p.concat,h=...
```

Using jQuery, continued

The src attribute of the script element can be used to load a JavaScript source file. By convention, it's in <head>.

```
<head>
    <script src="js/jquery.js"></script>
    ...
```

The closing tag must be present! (If wrong, nothing happens!)

The src attribute specifies a URL. The above is a relative URL that specifies jquery.js is in a subdirectory "js".

Note: This is a general mechanism with jQuery used as a specific example.

Loading jQuery from a CDN

A *content delivery network* (CDN) is a decentralized system of servers that deliver commonly requested static content with high performance.

Common practice is to load jQuery from a CDN. Here are three CDN-based possibilities:

```
<script src="http://ajax.googleapis.com/ajax/libs/jquery/
1.10.2/jquery.min.js"></script>

<script src="http://code.jquery.com/
jquery-1.10.2.min.js"></script>

<script src="http://ajax.aspnetcdn.com/ajax/jQuery/
jquery-1.10.2.min.js"></script>
```

The jQuery function (a.k.a. $)

When we load the jQuery source with a script element we end up with a jQuery(...) function:

> **typeof(jQuery)**
"function"

> **jQuery**
function (e,t){return new x.fn.init(e,t,r)} jquery.js:61

For convenience, the jQuery function is also assigned to the identifier $ (the dollar sign!)

> **typeof($)**
"function"

> **$ === jQuery**
true

Selection with jQuery

A fundamental capability of jQuery is selection of elements via a CSS selector specified as a string.

Example, with jquery1.html loaded:

```
> pars = $('p')          // Equivalent: jQuery('p')
[ <p>…</p>, <p>…</p>, <p>…</p> ]
```

In English: Create an array of all the paragraph elements in the current document and assign it to pars.

pars looks like an array but it has many jQuery functions, too!

```
> typeof(pars.hide)
"function"

> typeof(pars.append)     // Dozens more, too!
"function"
```

Selector testbed from McFarland's book

The downloadable files for *JavaScript and jQuery: The Missing Manual*, 2nd ed. by McFarland include a testbed for various aspects of jQuery.

Here's the testbed for selectors:

http://cgi.cs.arizona.edu/classes/cs337/fall13/jqmm/testbed/selectors.html

Let's try it! (I use a jqtb keyword to hit it.)

Exploring blog3.php with jQuery selectors

A <script> element for jQuery has been added to blog3.php.

Here's the URL:
http://cgi.cs.arizona.edu/classes/cs337/fall13/blog3.php

Problem: Use jQuery to see how many there are of each of these:

- divs
- Images
- Letters that hop (class=hop)
- Blog entries (class=entry)
- Blog entries with a tag

Bottom line: All(?) CSS selectors (and some jQuery extensions) can be used to select elements.

Implicit looping

Recall that the jQuery function ($) adds jQuery functions to the array it returns:

> **typeof($('p').hide)** // http://api.jquery.com/hide
"function"

If we call that function it is applied to **all** of the selected elements.

This hides all paragraphs:

> **$('p').hide()**

This causes all paragraphs to fade in over a period of three seconds:

> **$('p').fadeIn(3000)**

Implicit looping, continued

Many of the jQuery functions that are added to arrays of selections typically return that array:

> **pars = $('p')**
> [Stuff to do , Second paragraph... , … , …]
>
> **hidden = pars.hide()**
> [Stuff to do , ...]
>
> **pars === hidden**
> true

What can we do with that returned array?

p.s. Does jQuery use CSS visibility:hidden to implement hide()?

Chaining of jQuery calls

If a function returns an array of selections, we can call another function on that result.

What does the following do?

```
$('p').hide().fadeIn(3000).fadeOut(3000)
```

This chaining is a jQuery idiom.

jQuery's css() function

jQuery's css() function can be used to set CSS properties.

```
> $("li").css("color", "red")

> $("p").css("border", "1px green dashed")
```

An object can be used to set multiple properties at once:

```
> $("p:even").css(
    {"font-family": "monospace", color: "green"})
```

css(), continued

If the css() function is called with only a property name, it produces the value of that property for the first element:

> **$("p").css("font-size")**
"16.363636016845703px"

> **$("h1").css("font-weight")**
"bold"

If an array of property names is specified, the result is an object:

> **$('p:even').css(["color","font-family"])**
Object {color: "rgb(0, 0, 0)", font-family: "Times"}

Speculate: What does the jQuery attr() function do?

Events

JavaScript programs are typically *event-driven*.

Events are things like loading a page, mouse clicks, mouse movement, pressing keys, and resizing the window.

We can use jQuery to specify a function to handle an event for an element.

Events, continued

We can use jQuery to specify a function to run when an event occurs. Example:

```
$('p').click(f);
```

The above says, "Whenever any paragraph element is clicked, run the function named f."

Here's the function f. We can say it is an *event handler*.

```
function f()
{
    $(this).append(" Clicked!");
}
```

Let's run jqevent1.html.

Events, continued

Let's take a closer look at the event handler:

```
function f()
{
    $(this).append(" Clicked!");
}
```

When an event handler is invoked the special variable `this` is set to reference the *target* of the event.

We use $(this) to wrap that ordinary JavaScript element in an array that has the jQuery functions attached.

We then use the jQuery append() function to append text to the text of the element.

Events, continued

The testbed for *JavaScript and jQuery: The Missing Manual* has an events page:

http://cgi.cs.arizona.edu/classes/cs337/fall13/jqmm/testbed/events.html

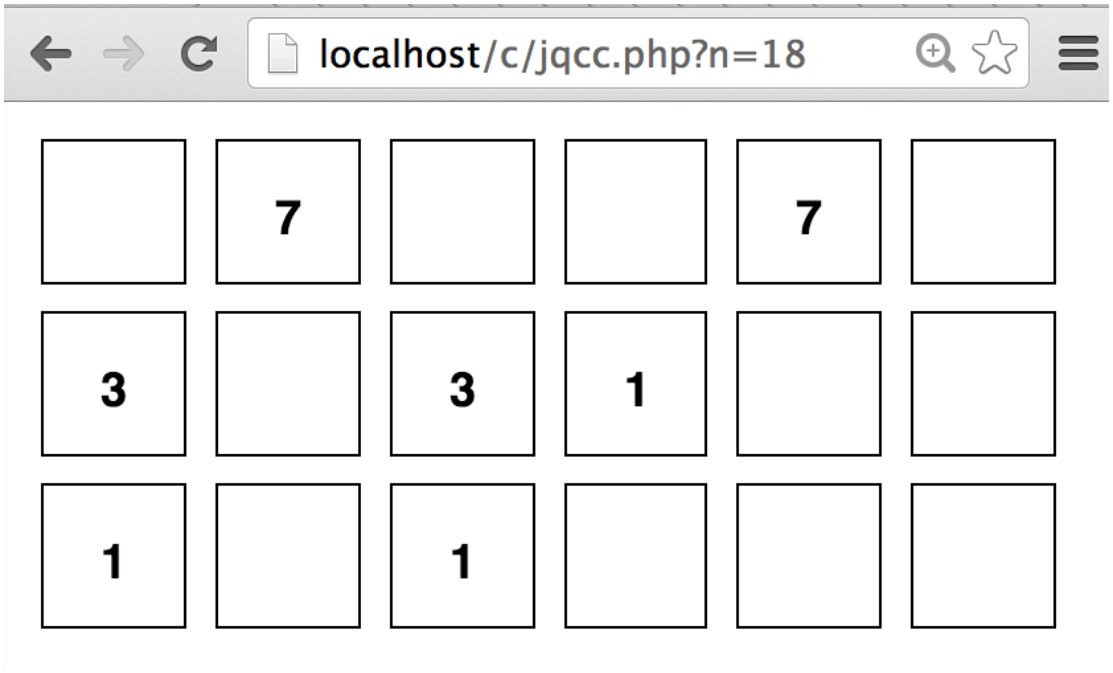
Let's try it and observe:

- mouse{up,down} and clicks
- mouse{over,out,move}
- key events, with modifier keys
- focus and blur on the input field

Also try testbed/content_functions.html

Problem: Click counter

Create a web app that displays N cells that hold a count of the number of times they've been clicked.



How can functionality be divided between HTML, CSS, PHP and JavaScript? Do we really need JavaScript?

Click counter solution

jqcc.php

```
<!doctype html>
<title>Click Counter</title>
<script src="jquery.js"></script>
<style>
div { height: 1em; width: 1em; padding: 1em; margin: 5px;
     border: solid 1px; display: inline-block;
     text-align: center; font: bold 1em sans-serif
     }
</style>
<?php
for ($i = 0; $i < $_GET["n"]; $i++)
   echo "<div> </div>";
?>
<script>
$("div").click(
   function () {
      var target = $(this);
      var count = +(target.text()) + 1;
      target.text(count);
      }
   );
</script>
```

jQuery's ready() function

An issue we haven't considered is when "top level" jQuery code should be run.

Native JavaScript provides a "load" event that "fires" when a page and all resources, like images, have been fully loaded, but jQuery code can typically be executed as soon as the DOM hierarchy is fully constructed.

jQuery's ready() function is used to specify code to run as soon as the DOM is ready.

ready(), continued

jqcc2.php uses the ready() function. Note that its argument is document:

```
<title>Click Counter</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/
1.10.2/jquery.min.js"></script>
<script>
$(document).ready(function () {
   $("div").click(
      function () {
         var target = $(this);
         var count = +(target.text()) + 1;
         target.text(count);
         });
   });
</script>
[<style> and <?php code as before...]
```

This is the typical way to use jQuery, wrapping the top-level code in a function called by $(document).ready().

Ajax

"Ajax" stands for Asynchronous JavaScript and XML.

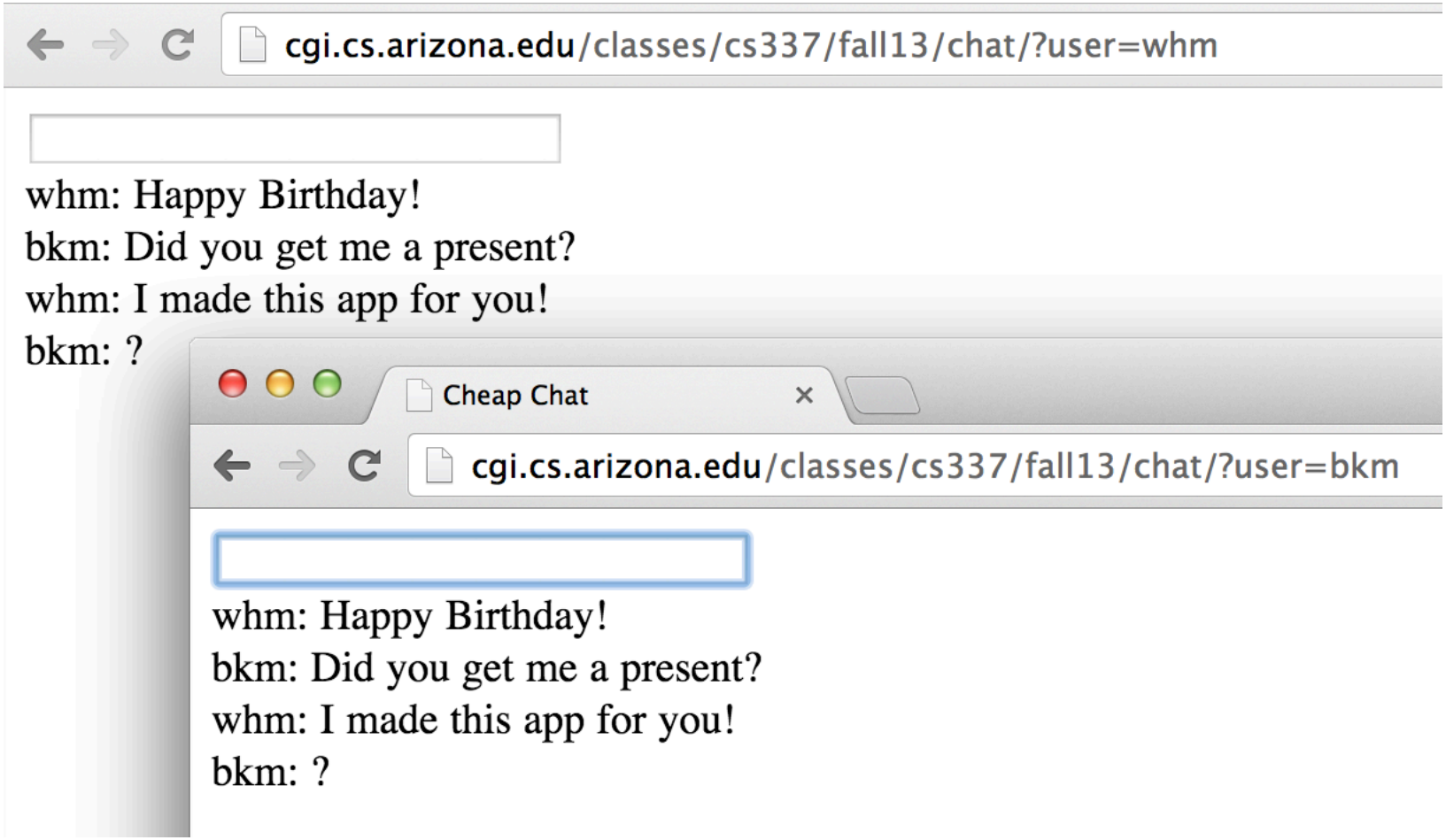
The basic idea of Ajax is to make a GET or POST request, but instead of reloading the entire page, the data in the response is used to change only some element(s) of the page.

Instead of the whole page being rendered again, only part of it is rendered again.

With Ajax a web app can approximate the user experience a desktop app provides.

With native JavaScript, XMLHttpRequest is used to make an Ajax call, but we'll use jQuery functions instead.

Cheap Chat: A bare-bones chat web app with Ajax



Everybody: Go hit it! Don't forget user=... Keep it clean!

How can we build it?

Cheap Chat overview

Use MySQL table to hold messages:

```
create table message (
   id        bigint not null auto_increment primary key,
   sender varchar(20) not null,
   text     varchar(160) not null );
```

Insert row when user hits ENTER in text input field.

Check once a second for new messages and append any that are returned to a div:

Cheap Chat HTML

```
<!doctype html>                    <!-- from chat/index.php -->
<title>Cheap Chat</title>
<script src="http://ajax.googleapis.com/ajax/libs/jquery/
1.10.2/jquery.min.js"></script>
<script>
   $(document).ready(
      function ()
      {
         ...top-level code...
      });
</script>
<input id=input size=40 autofocus>
<div id=messages></div>
```

Ajax with jQuery

With jQuery, we use $.get (or $.post) instead of XMLHttpRequest. Example:

```
$.get("backend.php",
    { newerthan: latest_msg_id },
    new_messages);
```

If latest_msg_id is 12, a GET is performed on this URL:

backend.php?newerthan=12

Execution immediately continues with the statement following the $.get(...) call.

Analogy: Instead of making a phone call, we dropped a letter in the mail and went on about our business.
(This is the asynchronous part—the "A" in "Ajax".)

Ajax with jQuery, continued

Here's the backend.php code that gets executed:

```
$stmt = $conn->prepare(
    "select id, sender, text from message
        where id > :newerthan order by id");
    ...bindParams, execute, check result...

 $rows = $stmt->fetchAll(PDO::FETCH_ASSOC);

echo json_encode($rows);  // JavaScript Object Notation
```

Here's a string that GET backend.php?newerthan=12 might produce:

```
'[{"id":"13","sender":"whm","text":"Hello!"},
{"id":"14","sender":"whm","text":"Testing!"}]'
```

The 89-character JSON string represents an array with two messages. It is the response to 'GET backend.php?newerthan=12'

Ajax with jQuery, continued

Recall::

```
$.get("backend.php", { newerthan: latest_msg_id }, new_messages);
```

Here is new_messages, which is called asychronously with the GET response. Note that we access the messages as JavaScript objects.

```
function new_messages (data)
{
   var msgs = JSON.parse(data); // turn JSON string into JS array

   if (latest_msg_id == 0) { // discard first batch, which is old msgs
      latest_msg_id = msgs[msgs.length-1].id;
      return;
      }

   for (var i = 0; i < msgs.length; i++) {
      var msg = msgs[i];
      $("#messages").append(
            msg.sender + ": " + msg.text + "<br>");
      latest_msg_id = msg.id;
      }
}
```

Ajax with jQuery, continued

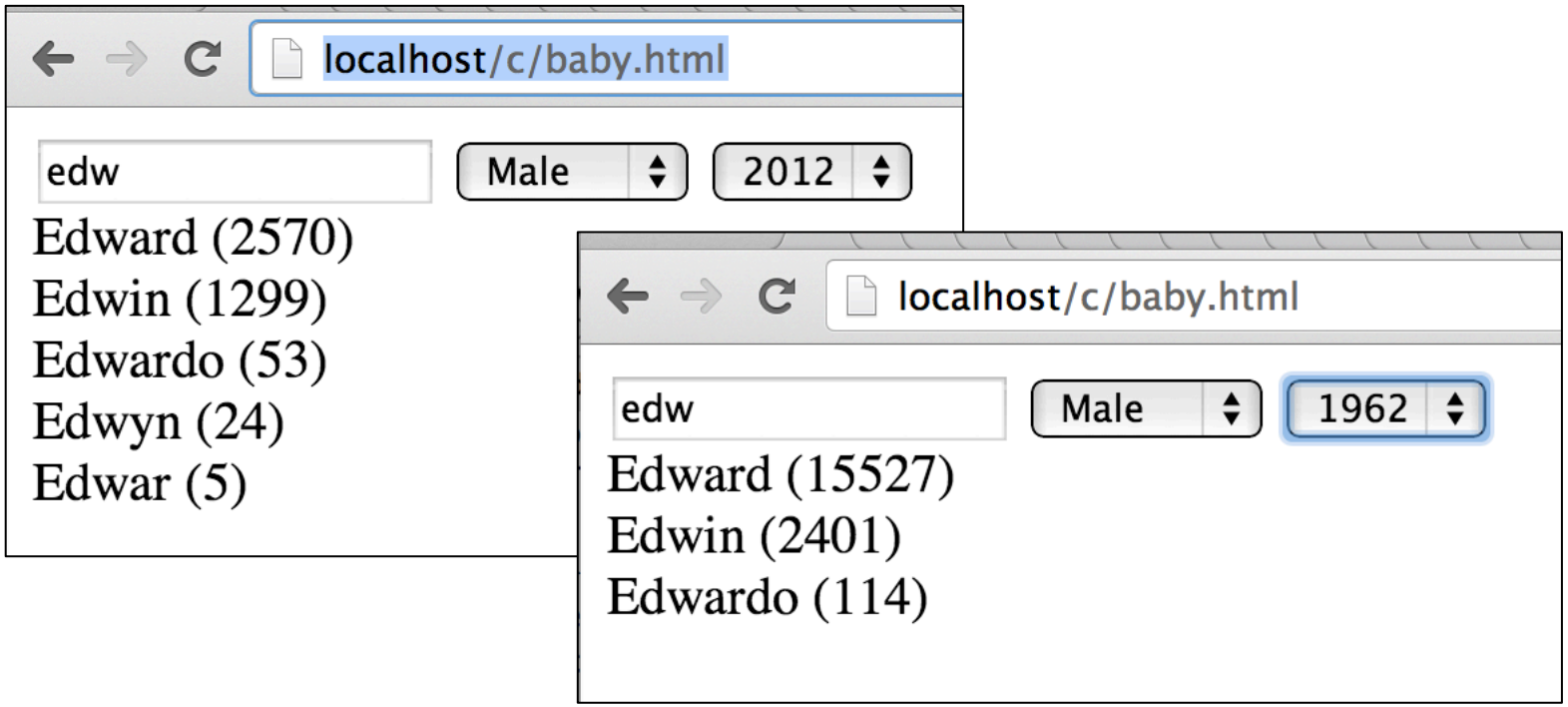
The code below goes in "...top-level code..." on slide 86.

```
var latest_msg_id = 0
setInterval(get_new, 1000);        // Call get_new every 1000ms

$("#input").change(function () {  // Called when user hits ENTER
                                  //   in input field. (Has id=input)

   var line = $("#input").val();      // Get input field's current value
   sender = "<?=$_GET["user"]?>"
   $.get("backend.php", { sender: sender, text: line});
   get_new();

   $("#input").val("");               // Clear input field
   })

function get_new() {
   $.get("backend.php",
       { newerthan: latest_msg_id }, new_messages);
   }
```

Another Ajax app: Birth name counts

The U.S. Social Security Administration makes available yearly counts of first names on birth certificates back to 1885.

baby.html uses Ajax to show names matching a prefix as it is typed:



Birth name counts: baby.html

```
<script>
$(document).ready(function () {
      var name = $("#name"); var gender = $("#gender"); var year = $("#year");

      for (var y = 2012; y >= 1885; y--)
         year.append("<option>" + y);

      name.keyup(getnames); gender.change(getnames); year.change(getnames);

      function getnames() {
         $.get("nameswith.php",
            {name: name.val(), gender: gender.val(), year: year.val()}, shownames);
         }

      function shownames(data) { $("#list").html(data); }
});
</script>
<input id=name type=text autofocus>
<select id=gender name=gender>
      <option value="m">Male<option value="f">Female
</select>
<select id=year name=year></select>
<div id=list></div>
```

Birth name counts backend: nameswith.php

Example GET: nameswith.php?name=edw&gender=m&year=2012

```
//
// Instead of writing PHP to search we pass the buck to grep!
//
// popen(..., "r") lets us reads the output of a shell command.
//
// Sample grep invocation:  grep -i '^edw.*,m,' yob2012.txt
//
$f = popen("grep -i '^{$_GET['name']}.*,{$_GET['gender']},' " .
                    "yob/yob{$_GET['year']}.txt", "r");
//
// Output HTML, broken into lines with <br> elements.
//
while ($line = fgets($f)) {
   $parts = explode(",", trim($line));
   echo "{$parts[0]} ({$parts[2]})<br>";
   }
```

Benched: draw and colony from assignment 9

Two possibilities for assignment 9 were:

"draw"—A collaborative drawing app

Run it with a URL color parameter:

http://cgi.cs.arizona.edu/classes/cs337/fall13/a9/draw.php?color=rgb(23,100,200)

Draw by holding down shift and moving the mouse.

"colony"—A simple game

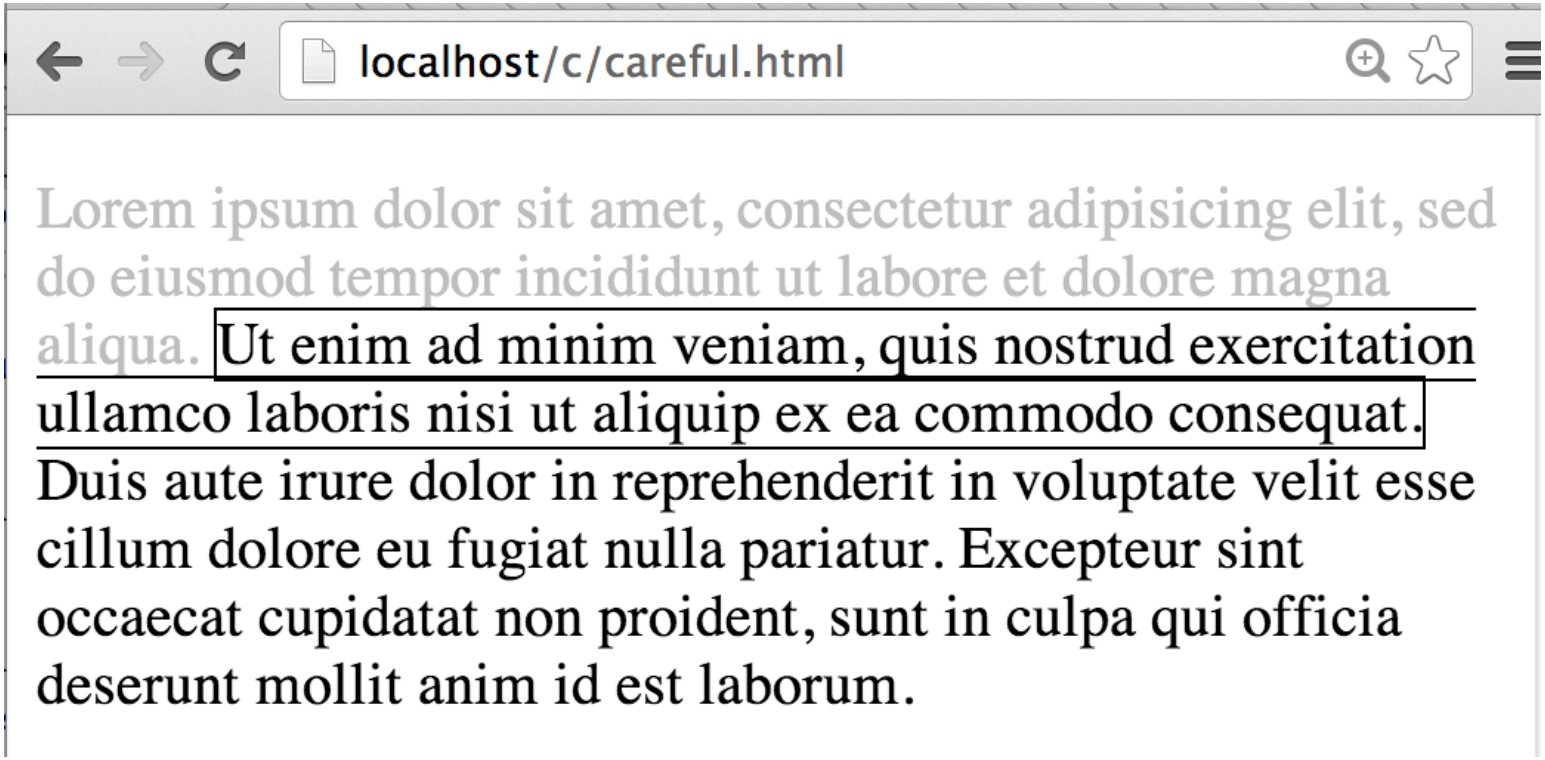
http://cgi.cs.arizona.edu/classes/cs337/fall13/a9/colony.php

Draw by holding down shift and moving the mouse, and wipe out the red colonies.

Unfinished!

Example: Careful reading

Idea: Present paragraphs of text. When a sentence is first clicked, a box is placed around it. A second click grays it.



How can we approach it?

Careful reading, continued

```
$(document).ready(function () {

   $("p").each(wrap_sentences);   // call wrap_sentences on each paragraph
   $(".initial").click(cycleSentence);  // add handler for click to each sentence

   //
   // Wrap each sentence in the paragraph with <span class=initial>...</span>
   //
   function wrap_sentences(index, par) {  // index is position--not used
      par = $(par);  // Make one-element jQuery array from native JS <p>
      var sentences = par.text().split(".");  // Simple minded!
      par.text("");
      for (var i = 0; i < sentences.length-1; i++) {
         nsen = $("<span class=initial>" + sentences[i] + ". </span> ");
         par.append(nsen);
         }
      }
   ...
```

Careful reading, continued

Classes are used both to represent state and to style:

```
<style>
 .initial { }
 .boxed { border: 1px solid; }
 .faded { border: none; color: silver; }
</style>
```

cycleSentence is called when a sentence is clicked:

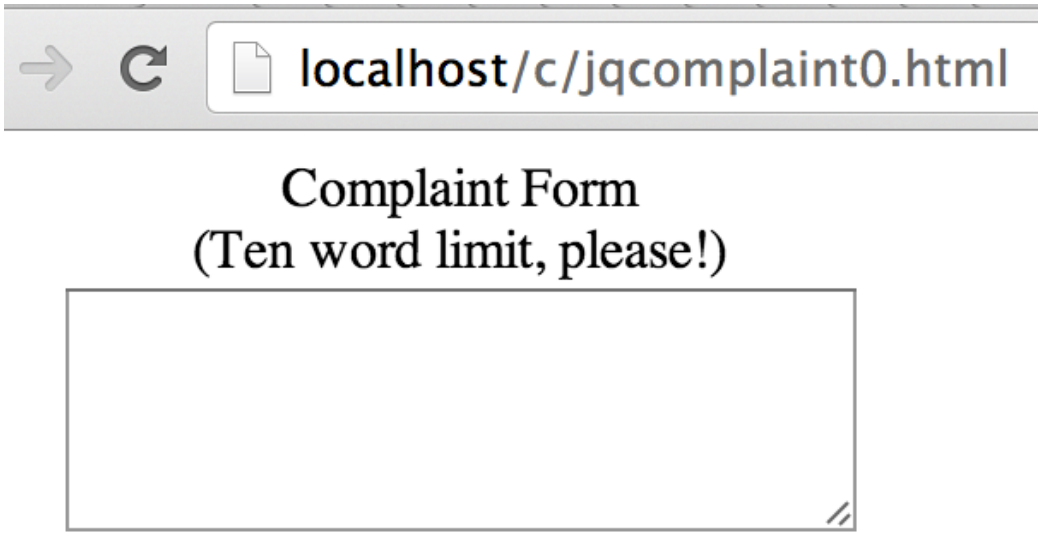
```
function cycleSentence() {
    sen = $(this);
    if (sen.hasClass("initial"))
       sen.removeClass("initial").addClass("boxed");
    else
       sen.removeClass("boxed").addClass("faded");
 }
```

Alternative: jQuery's data() method can be used to associate arbitrary data with elements.

Example: Complaint form

Problem: Limit user input in a complaint form to ten "words". jqcomplaint0.html is the starting point:

```
<script>
$(document).ready(function() {
   $("#text").keydown(function() { /* TODO */ })
});
</script>
<style>
   #complaint {
      width: 20em;
      text-align: center
      }
</style>
<div id=complaint>
Complaint Form<br>
(Ten word limit, please!)<br>
<textarea id=text rows=5 cols=30 autofocus></textarea>
```



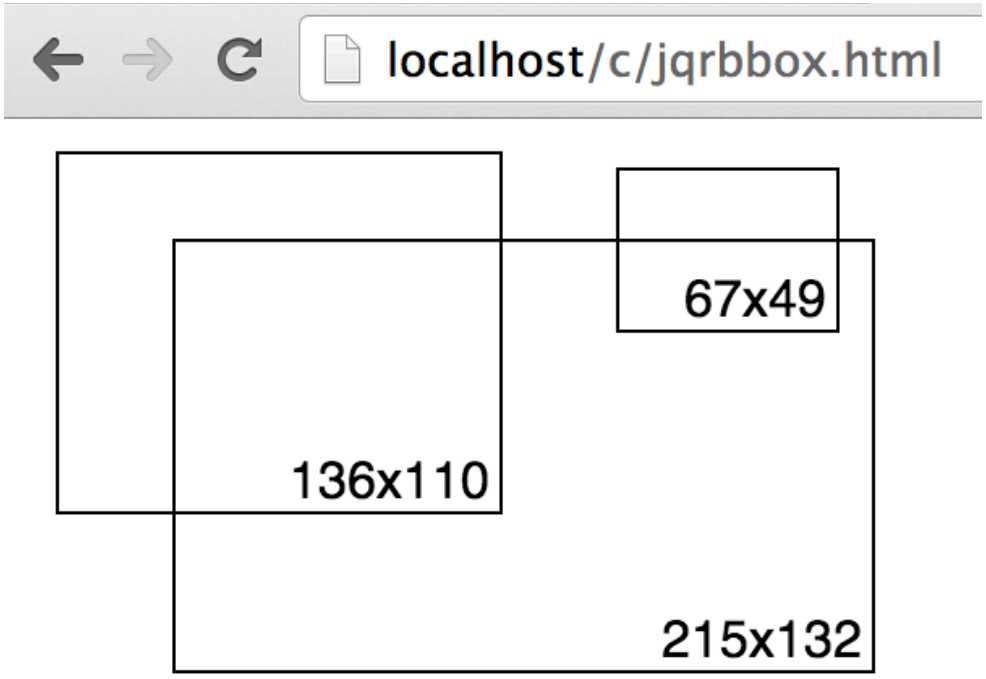
Complaint form solution

```
<script>
$(document).ready(function() {                  // jqcomplaint.html
   $("#text").keydown(function() {
      var words = $(this).val().split(" ").length
      if (words > 10 && event.keyCode != 8) {
         $("#limit").addClass("red");
         return false;  // Or, event.preventDefault();
         }
      else {
         $("#limit").removeClass("red");
         }
      $("#count").text(words + " words");
      })
     // no jQuery .paste() so use .bind(...) instead to handle and prevent pasting
     $("#text").bind("paste", null, function() { alert("Sorry, no pasting!"); return false; });
});
</script>
<style>
   #complaint { width: 20em; text-align: center }
   .red { color: red }
</style>
<div id=complaint>
Complaint Form<br>
<span id=limit>(Ten word limit, please!)</span><br>
<textarea id=text rows=5 cols=30 autofocus></textarea><br>
<span id=count></span>
```

Go back to Ajax, slide 83

Rubber-banding box with low level events

jqrbbox.html uses mousedown, mousemove, and mouseup events to draw boxes with a "rubber-banding" effect.



Try it, then discuss events and how to approach it.

Rubber-banding, continued

Per-event activity:

mousedown:
Save current position of mouse as upper-left corner.

Set flag indicating that drawing is in progress.

mousemove:
Get rid of previous div drawn by mousemove, if any.

Create a div with established upper-left (by mousedown) and lower- right at current position of mouse.

mouseup:
Clear drawing in progress flag.

Switch div border from dotted to solid.

Simplification: Assume "sweep" is always upper-left to lower-right.

Rubber-banding, continued

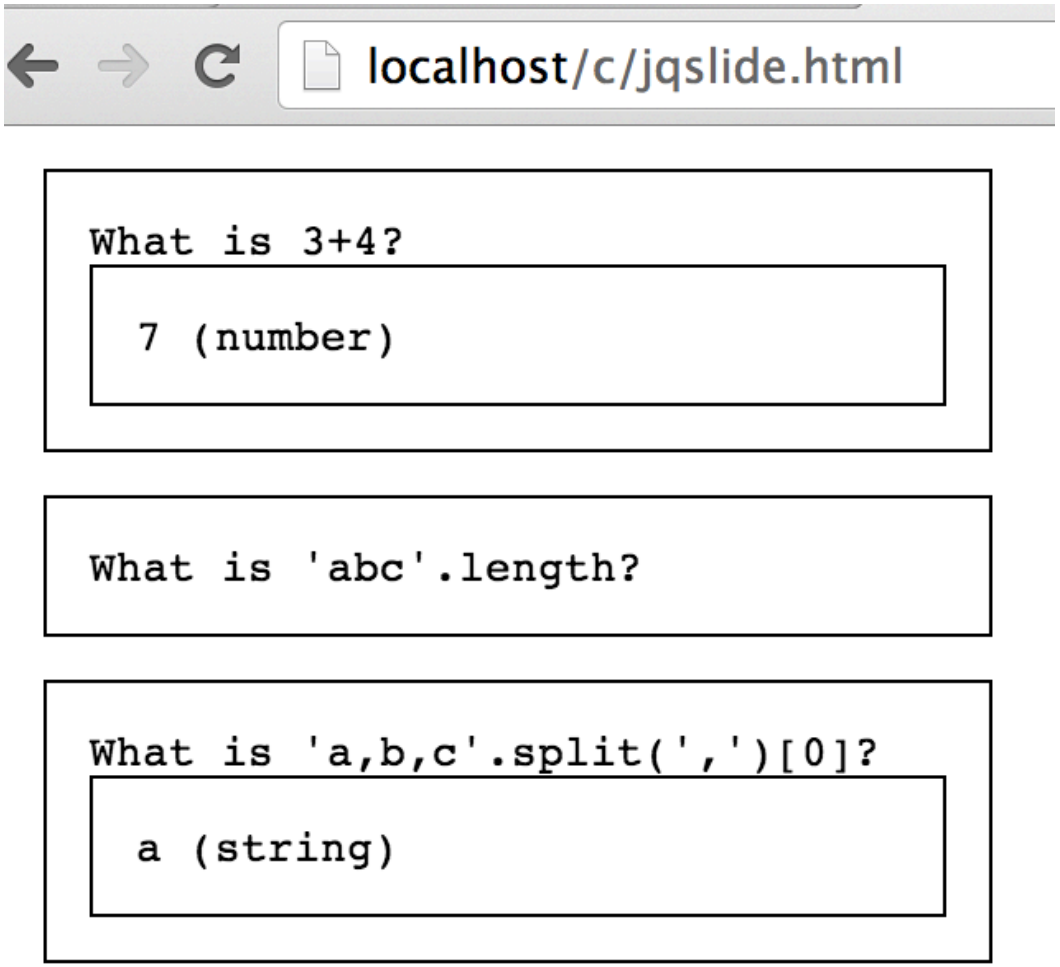
```
var box_in_progress = false;
var box_num = 1;

$('#outer').mousedown(function () {
      startX = event.clientX;
      startY = event.clientY;
      box_in_progress = true;
      });

$('#outer').mousemove(function () {
      if (!box_in_progress)
         return;
       $("#box" + box_num).remove();
       $("#outer").append(div(startX, startY,
                                          event.clientX, event.clientY));
       event.preventDefault(); // suppress text selection w/ drag
      }
   )
```

Rubber-banding, continued

```
$('#outer').mouseup(function() {
    box_in_progress = false;
    var box = $("#box" + box_num);
    box.removeClass("in_progress").addClass("positioned");
    box_num += 1;
    })

  // draw div with (x0, y0) and (x1, y1) as upper-left and lower-right, resp.
  function div(x0, y0, x1, y1) {
    var width = x1 - x0;
    var height = y1 - y0;
    return "<div class=in_progress id=box" + box_num + " style=" +
      prop("left", x0) + prop("top", y0) +
      prop("width", width) + prop("height", height) + ">" +
        "<span class=wh_feedback>" +
        width + "x" + height +
        "</span></div>";
    }

  function prop(prop, px) { return prop + ":" + px + "px;" }
```

slideDown()

jqslide.html uses slideDown() to cause answers to slide into view as questions are moused over.



slideDown(), continued

```
var questions =
   ["3+4", "'abc'.length", "'a,b,c'.split(',')[0]", "4/3", "1||2", "true||2"];

for (var i = 0; i < questions.length; i++) {
    $('body').append("<div class=question>What is " +
        questions[i] + "?" + answer(questions[i]) + "</div>");
    //
    // Note the two-argument form of $(...), to look for
    // elements with class=answer in "this", the question div that
   // was clicked.
    //
    $('.question').mouseover(
        function () { $('.answer', this).slideDown(); })
    }

function answer(expr) {
    var result = eval(expr);  // evaluates JavaScript expression!
    return "<div class=answer>" + result + " (" +
          typeof(result) + ")</div>";
    }
```

slide{Down,Up} with hover()

jQuery's hover() function has a two argument form, which specifies to functions to execute when hovering begins and ends, respectively.

We can use it to cause answers to show when one is hovering over a question:

```
$('.question').hover(
    function () { $('.answer', this).slideDown(); },
    function () { $('.answer', this).slideUp(); }
    )
```

Try it with jqslide2.html

Animation

Effects like slides and fades are done by making a series of small changes to properties with numeric values.

jQuery's animate() function lets us specifies animations of property values over a period of time.

Simple example:

```
$("div").animate({width: "400px"}, 2000)
```

That call will cause the width of all divs to be gradually changed over a period of 2000 milliseconds, either increasing or decreasing each width to 400px.

Try jqanim1.html, then try some chained animations.

Animation, continued

Here's the core of jqanim2.html. How does it behave?

```
var initial = $("div").css(["width","height","opacity"]);

$("div").click(function () {
    $(this).animate( {
        width: ($(this).width() * 1.3) + "px",
        height: ($(this).height() * 1.2) + "px",
        opacity: ($(this).css("opacity") * .9)
        }, 100 )
        return false; // stop the click from reaching body
    });

$("body").click(function () {
    $("div").animate(initial, 1000)
    })
```

Animation, continued

This is jqanim3.html. Can you predict its behavior?

```
<style>
   html { background: black }
   div { width: 150px; height: 75px; float:left;
         margin: 10px; border: solid 1px; background: white}
</style>
<script>
$(document).ready(function() {
   for (i = 1; i <= 100; i++)
      $('body').append("<div></div>");

   $('div').mouseover(function () {
      d = $(this)
      d.animate({width: d.height() + 'px',
                 height: d.width() + 'px'}, 1000)
      });
   });
```

Try jqanim4.html, too.

Animation, continued

.animate() has an optional third argument: a function to call when the animation is complete. jqanim5.html uses it:

```
$("div").click(function () {
      cycle(this, ["One","Two","Three","Four","Five"],10, 60)
      })

   function cycle(e, strings, start, end) {
      if (strings.length == 0) return

      $(e).html("<span style=font-size:" + start + "px>" +
        strings.shift() + "</span>");
      $("span", $(e)).animate({fontSize: end + "px"}, 1000,
        function () { cycle(e, strings, start, end) })
      }
```

Animation, continued

Harrison created hsanim1.html. Here is its core:

```
$(function(){
    $('div').hover(function() {
        var pwidth = $('body').innerWidth(),
            pheight = $('body').innerHeight();
        var top = Math.random()*pheight,
            left = Math.random()*pwidth;
        $(this).animate({
            top: top + "px",
            left: left + "px"
        }, 100);
        $(this).css('background', RandColor());
    });
});
var RandColor = function(){
    var r = Math.floor((Math.random()*255)+1),
        g = Math.floor((Math.random()*255)+1),
        b = Math.floor((Math.random()*255)+1),
        rgb = 'rgb('+r+','+g+','+b+')';
    return rgb;
}
```

Run it, then answer this: Why does the circle make two movements?

"easing" in animations

Consider an animation that takes a value x from 0 to 100 over a period of 1000ms. What should x be at 100ms, 300ms, 900ms?

jQuery uses "easing functions" that specify how an animated value should change over time.

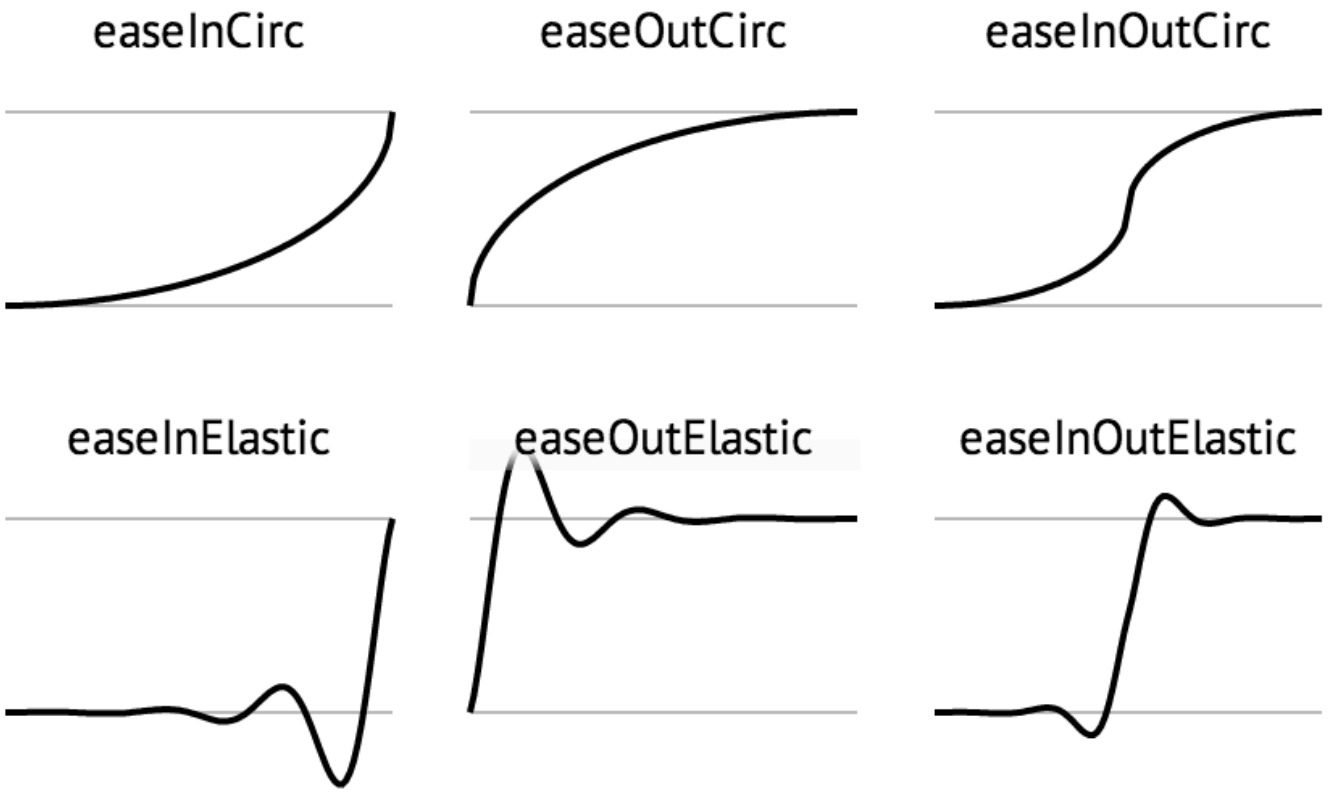
With "linear" easing, the rate of change is constant. For the question above, x would be 10, 30, 90 respectively.

jQuery's default easing is "swing".

jqanim6.html compares "linear" and "swing" easing functions.

easing functions, continued

easings.net has animations of a number of easing functions. Here's a static shot of some:



easing functions, continued

jqanim7.html allows comparison of linear easing to the easing functions in the jQuery Easing Plugin, which can be found at gsgd.co.uk/sandbox/jquery/easing.

jQuery UI

jQuery UI provides a collection of user interface components. It also supports additional effects and provides facilities such as drag and drop.

It's built on top of jQuery.

It's at jqueryui.com.

It's about 30k lines of JavaScript; 2500 lines of CSS in the base "theme". (Based on quick counts—don't quote me!)

Components are "themeable".

jQuery UI configuration

The short story:

```
<script src="http://code.jquery.com/jquery-1.10.2.js"></script>
<script src="http://code.jquery.com/ui/1.10.3/jquery-ui.js">
</script>
<link rel="stylesheet" href="http://code.jquery.com/ui/1.10.3/
themes/smoothness/jquery-ui.css">
```

The long story:

http://jqueryui.com/download/ (The Download Builder)

jQuery UI "Interactions"

jQuery UI supports several high-level mouse-based interactions:

- Dragging
- Dropping
- Resizing
- Selecting
- Sorting

jQuery UI uses the term "widget" to refer to both visible components, like menus, and the encapsulation of code that implements interactions.

There's a Draggable Widget, a Droppable Widget, a Resizable Widget, etc. (api.jqueryui.com/category/interactions)

The Draggable Widget

To allow an HTML element be dragged about with the mouse, we simply call draggable() on a jQuery selection!

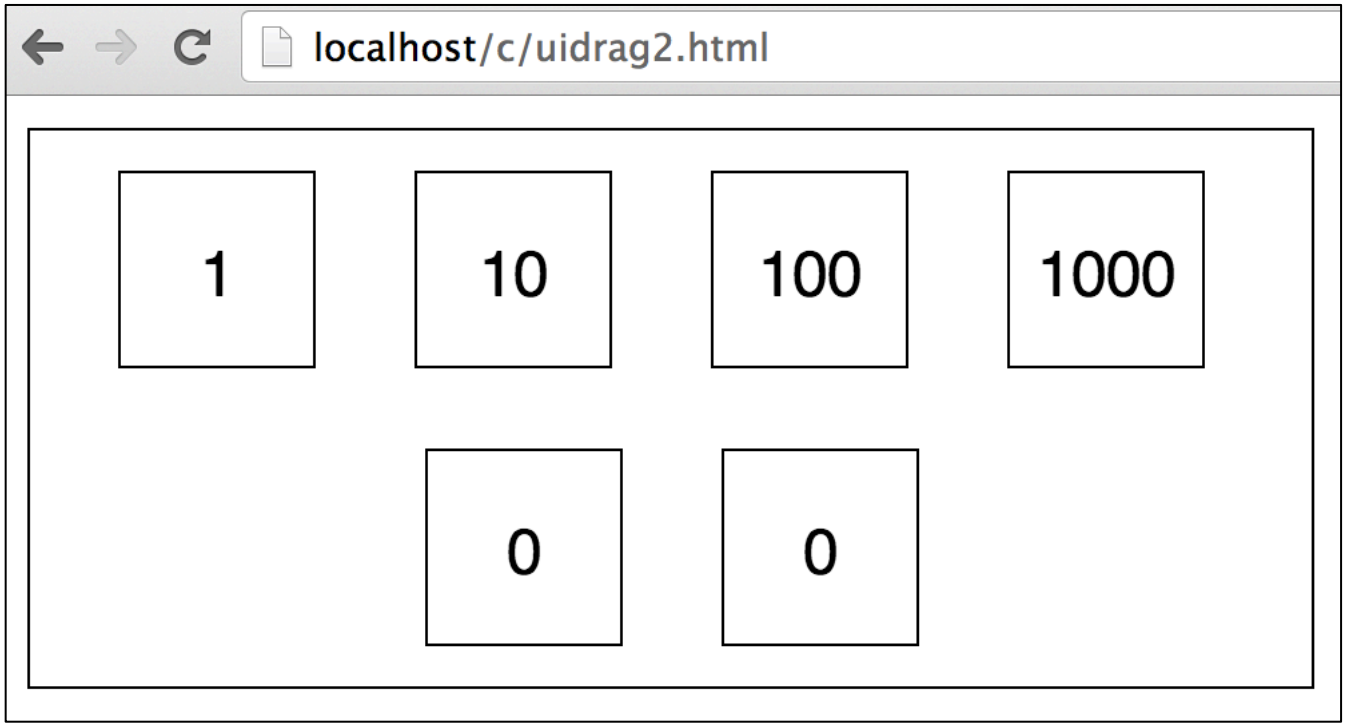
```
<style>
.disc { width: 150px; height: 150px; border-radius: 75px;}
</style>

<script>
$(document).ready(function () {
  $('.disc').draggable();
});
</script>

<div id="content" style="height: 400px;">
  <div class=disc style=background:rgba(100,0,0,.6)></div>
  ... two more...
```

Let's run uidrag1.html and observe changes via the Elements tab.

Drag and drop

We can apply the Droppable widget to an element to allow Draggables to be dropped on it.

uidrag2.html allows numbers to be dragged/dropped onto accumulators that hold a sum:



Drag and drop, continued

Styling and elements:

```
<style>
   body { font-family: sans-serif }
   #content { width: 20em; border: 1px solid; text-align: center }
   .number { display: inline-block; min-width: 3em; height: 2em;
                  border: solid 1px; text-align: center;
                   padding-top: 1em; margin: 10px }
</style>

<div id="content">
   <div class=number>1</div>
   <div class=number>10</div>
   <div class=number>100</div>
   <div class=number>1000</div>
   <br>
   <div class="accumulator number">0</div>
   <div class="accumulator number">0</div>
</div>
```

Note that there are no CSS rules for the accumulator class. We use it to mark cells that we can drop numbers onto and get accumulation.

Drag and drop, continued

The core code:

```
// We use Draggable's "helper" option to indicate that a copy
// of the element is to be dragged, instead of the element itself.

$('.number').draggable( { helper: "clone" } );

// We use Droppable's "drop" option to specify a handler to be
// called when an element is dropped.

$('.accumulator').droppable( { drop: handleDropEvent } );

// The drop handler gets references to the element being
// dragged and the element it's being dropped on and replaces
// the target's text with the sum of the contained values.

function handleDropEvent( event, ui ) {
    var draggable = ui.draggable;
    var target = $(event.target);
    target.text(+target.text() + +draggable.text());
}
```

Drag and drop, continued

Minor improvement: instead of cloning the element being dragged we can specify a function that will return an element to be dragged.

```
$('.number').draggable({ helper: myHelper });
                // instead of { helper: "clone" }

function myHelper( event ) {
   return "<div id=helper>" + $(event.target).text() +
             "</div>"; }
```

Styling:

```
#helper { width: 3em; height: 2em; padding-top: 1em;
          border-radius: 1.5em; text-align: center;
          background: rgba(70,70,70,.5); font-weight: bold}
```

uidrag2a.html uses it.

jQuery UI Components

jQuery UI includes a number of user interface components:

- Accordion
- Button
- Datepicker
- Dialog
- Menu
- Progressbar
- Slider
- Spinner
- Tabs
- Tooltip

Components are another type of jQuery UI widget.

We'll look at a sampling. jqueryui.com has demos for all.

Datepicker

The HTML5 input element allows type=date but few browsers support it. With jQuery UI's Datepicker, we can have calendar-based input in all browsers.

We can create a Datepicker by calling .datepicker() on an input element.

Datepicker, continued

uidate1.html:

```
var bday = $("#birthday");
    bday.datepicker({    // argument is object specifying options
    changeYear: true,    // add drop-down to select year
    showWeek: true,      // show week number, too
    beforeShowDay: function (date) {   // only allow even days
       return [date.getDate() % 2 == 0]; }
    })

// When the date changes use the getDate method to append
// it to a logging area.
bday.change(function () {
   $("#log").append(bday.datepicker("getDate") + "<br>");
   });
...
Birthday: <input id=birthday type=text><div id=log></div>
```

Tabs

A section of tabbed content can be created by calling .tabs() on a div with a structure like this:

```
<div id=sections>
   <ul>
        <li><a href=#intro>Introduction</a>
        <li><a href=#story>Story</a>
        <li><a href=#concl>Conclusion</a>
   </ul>
   <div id=intro>Lorem ipsum dolor ...</div>
   <div id=story>Aenean porttitor ultrices ...</div>
   <div id=concl>Cras convallis ligula ...</div>
</div>
```

Try uitabs1.html. Click it to create the tabs.

Tabs, continued

Here's the JavaScript for uitabs1.html:

```
$("body").click(makeTabs);

function makeTabs() {
   $("#sections").tabs( {
      activate: function (event, ui) {
         var href = $("a", ui.newTab).attr("href");
         $("#log").append("activated " + href + "<br>");
         }
      });
   }
```

The options object passed to .tabs() specifies a handler for the Tabs "activate" event.

What other events do Tab widgets have available?

Dialogs

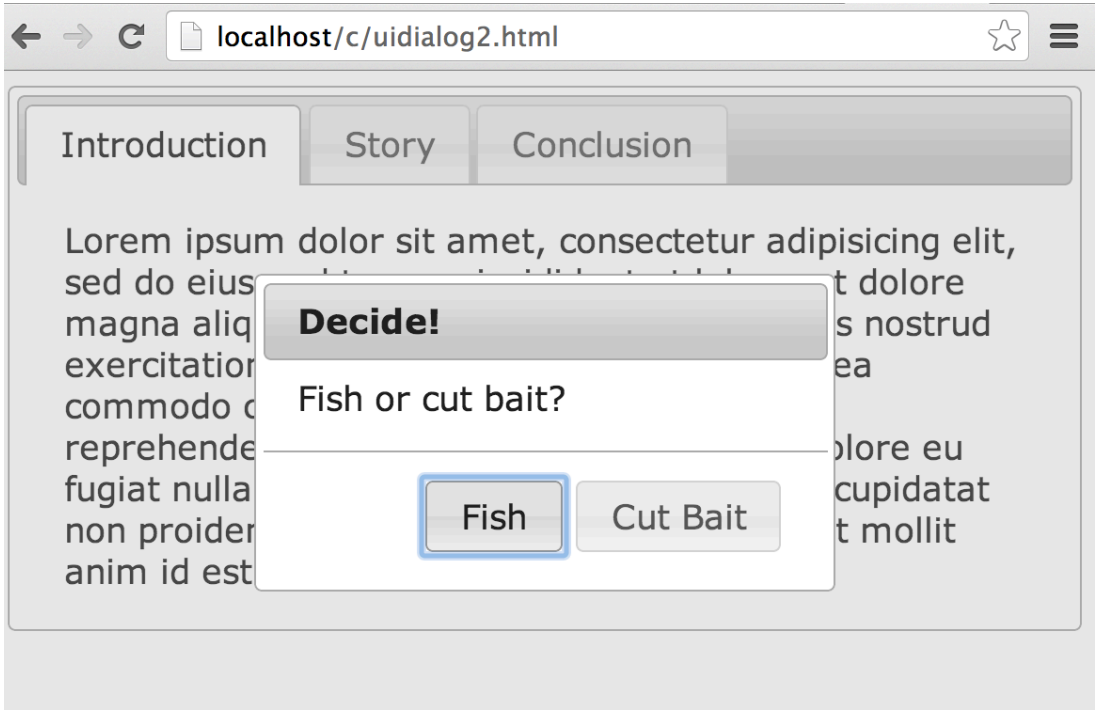
Applying .dialog() to an element turns it into a pop-up dialog. The code below creates a dialog immediately and then creates a second when the first is closed: (uidialog1.html)

```
<script>
$(document).ready(function() {
   $("#hello").dialog(
        { close: function () { $("#thanks").dialog() } })
});
</script>
<div id=hello title=Greetings>Hello, world!</div>
<div id=thanks style=display:none title=Thanks!>
Thanks for closing him.</div>
```

Note that display:none is used to hide the second dialog until we're ready for it. (Alternative: create it!)

Modal dialogs

A "modal" dialog is appropriate when we don't want to let the user do anything until they perform some action.



Note that we've hidden the 'X' icon and that an overlay dims the other elements and prevents interaction with them. We also prevent ESC from closing the dialog!

Modal dialogs, continued

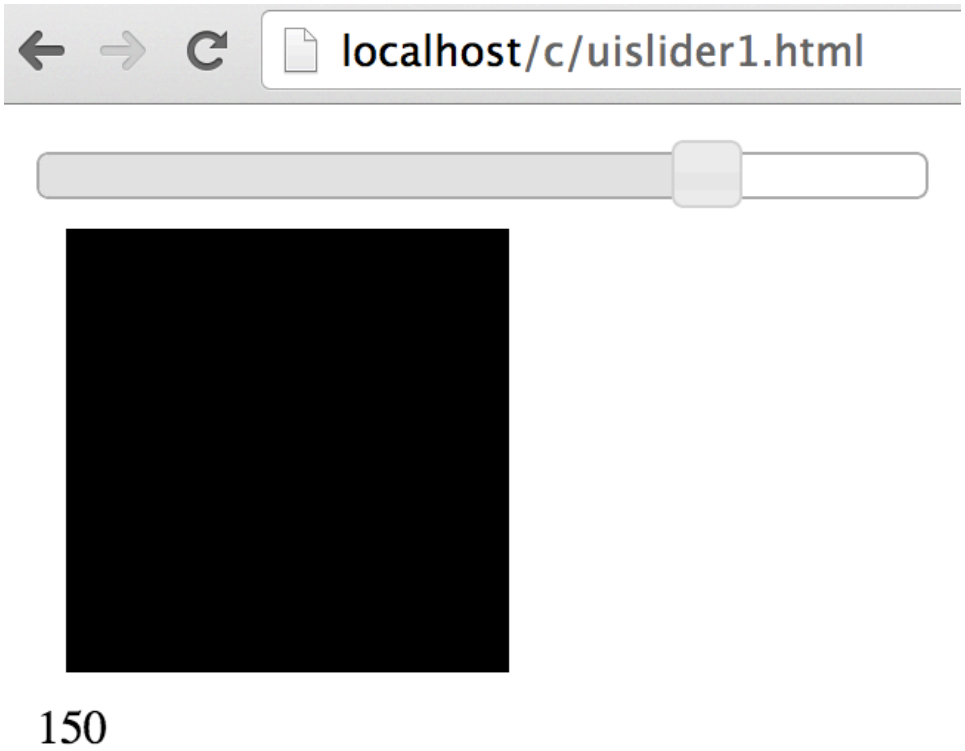
Here's the code from uidialog2.html:

```
.no-close .ui-dialog-titlebar-close {  display: none }
...
$("#decision").dialog(
   { closeOnEscape: false,
     dialogClass: "no-close",
     modal: true,
     buttons: {
       "Fish": function () { $(this).dialog("close"); fish(); },
       "Cut Bait":
             function () { $(this).dialog("close"); cut_bait(); }
       }
   })
...
<div id=decision title="Decide!">Fish or cut bait?</div>
```

Oddly, there's no option to suppress the 'X' (close) icon! The documentation recommends the maneuver with the no-close class.

Sliders

A slider provides a way for a user to adjust value within a range. uislider1.html displays a slider that sizes a div:



The "thumb" can be adjusted with mouse or keyboard.

Sliders, continued

Here's the code to create the slider:

```
$(document).ready(function() {
    var size = 50;            // initial size
    resizeBox(size);          // makes box match initial size
    $("#size").slider({
      orientation: "horizontal",
      range: "min",           // show fill to left of thumb
      max: 200,               // max value
      value: size,            // initial value for slider
      slide: refresh,         // called when user slides
      change: refresh         // called if value set by code
    });
```

Markup:

```
<div id="size"></div>     <!-- becomes the slider -->
<div id=box></div>        <!-- the box -->
<div id=log></div>        <!-- shows size, like "150" -->
```

Sliders, continued

Recall:

```
resizeBox(size);          // makes box match initial size
$("#size").slider({
    ...
    slide: refresh,       // called when user slides
    change: refresh       // called if value set by code
    });
```

Here are the refresh() and resizeBox() functions:

```
function refresh() {
    resizeBox($("#size").slider("value"));
    $("#log").text($("#size").slider("value"));
    }

function resizeBox(size) {
    size = size + "px"
    $("#box").css({width: size, height: size});
    }
```