

JavaScript

Introduction

Simple programming language in the browser (Totally Client Side)

Basically no relation to Java – just a marketing name

Used to be incompatible versions

Later became a standard under ECMA

Works best in IE 5 or later, Netscape 6 or later, Mozilla 1.0 or later, including Firefox. Mozilla probably has the best JavaScript implementation.

We will not worry about weird JavaScript required by old browsers

JavaScript Niche

Make the client more responsive in a client/server system

A slightly “thicker” client, but better user experience

Advantages: more responsive, doesn't require a request/response roundtrip

e.g. on product detail page for a t-shirt, have a pop-up menu for color. Use JavaScript to swap the image to show different shirts without doing a whole form submit.

JavaScript can do things in the browser: adjust the HTML in the page, adjust the window, open new windows

Javascript cannot: use the local filesystem, do networking.

JavaScript Language Basics

<script> section in the HTML document – runs on document load

No type declarations required

Variables are essentially global by default. e.g. count below. (variables are actually properties of a global context)

Function definitions

strings – "hello" or 'hello', use + to concat

"var" – declare a local variable (as opposed to a global)

alert(str) – puts up an alert panel

JavaScript and Browser

document – the HTML document

document.name – refer to a named element in the document

document.images – array of images

document.forms – array of forms

There are also ways to access the window, cookies, etc.

Use Mozilla's JavaScript Console to see error messages.

JavaScript Example

```
<html>
<head>
<title>JS Demo</title>

<script language="JavaScript">

function hello(greeting) {
    var str = greeting + "!!!";
    alert(str);
}

count = 0;
function upCount() {
    count++;
    alert(count);
}

function noFear() {
    var fear = document.affirm.fear.value;
    if (!document.affirm.mockMode.checked) {
        alert("No " + fear + " to be seen around here!");
    }
    else {
        var mock = "Being afraid of " + fear + " is stupid!";
        window.status = mock
        document.affirm.mock.value = mock;
    }
}
</script>

</head>

<body>

<p>

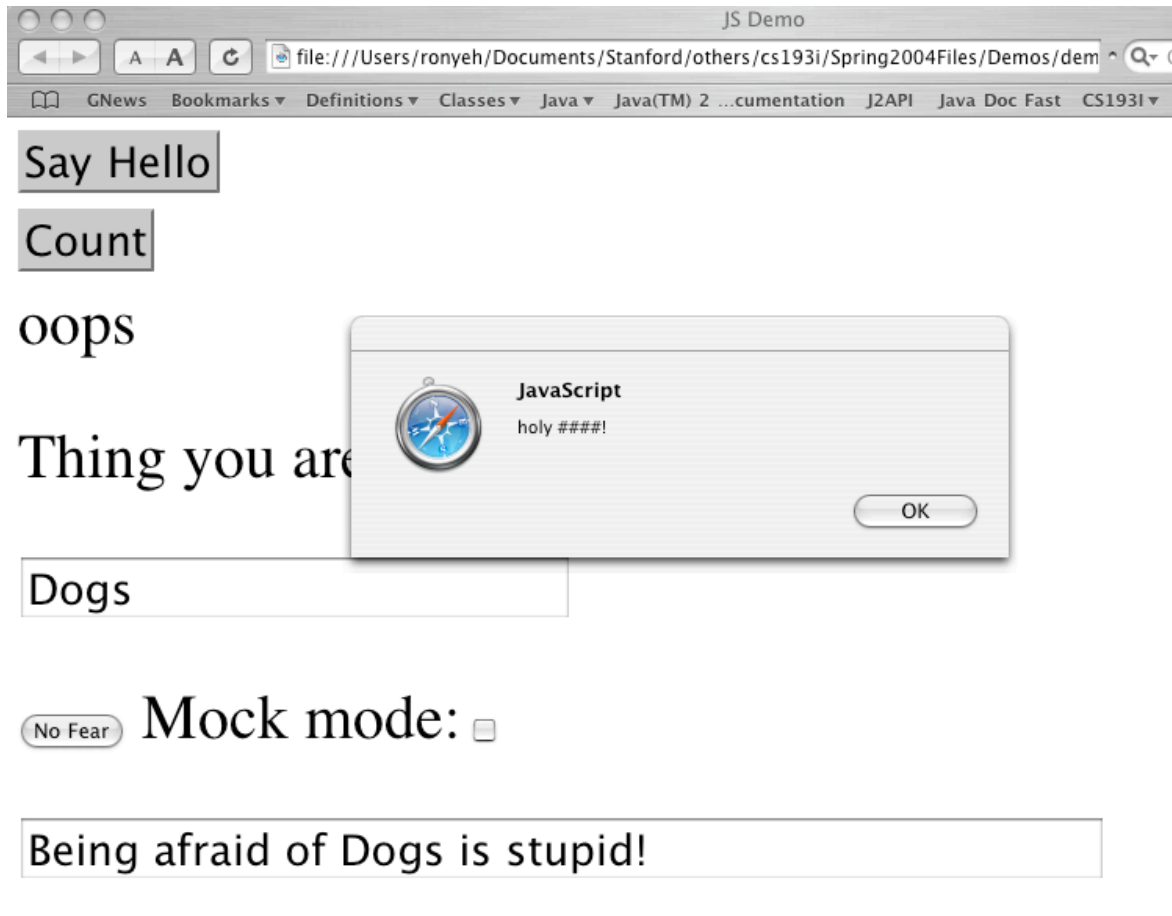
<button onClick="hello('hi there');" >Say Hello</button>

<br><button onClick="upCount();" >Count</button>

<br><a onClick="alert('holy ####!')">oops</a>

<p>
Thing you are afraid of...
<form name=affirm>
    <input type=text name=fear>
    <p><input type=button onClick="noFear();" value="No Fear">
    Mock mode:<input type=checkbox name=mockMode>
    <p><input type=text size=40 name=mock>
</form>

</body>
</html>
```



JavaScript Resources

JavaScript URLs on the course page

JavaScript the Definitive Guide 4th ed, by David Flanagan (O'Reilly)

The "Rhino" book

Important to get the latest edition, since the language (and browsers) have evolved a lot

JavaScript and Beyond

JavaScript has all sorts of features we're not going to worry about: objects, exceptions, regular expressions

But we're not messing with that

It is, at its heart, a simple language, intended for non-programmers, so a little goes a long way

Undefined

The "undefined" value is like undef in Perl

The value of a variable that has not been given a value

Use == to test for undefined

```
if (a == undefined) { ...
```

Strings

Either " or ' work as delimiters

Use + to concat strings (converts ints to string form automatically)

s.toLowerCase() // returns lowercase form

== does a "deep" string compare (unlike Java, which does a reference/pointer compare)

s.indexOf(target) // returns the index of the target, or -1

Arrays

a = new Array(); // new empty array

var b = new Array(); // as above, but "b" is a local variable

Add three elements

a.push(1);

a.push(2);

a.push("hello");

a[0] – access element 0

a[100] = "foo"; // makes array bigger

a.length // current length of array

Other obvious built-in functions: pop(), shift(), unshift(), sort(), join(), reverse()

c = [1, 2, "hello"]; // [...] literal defines an array on the fly

array.toString() // returns a string form, but without the []: 1,2,hello

Arrays - Objects

Arrays and objects are actually the same thing – the a[1] syntax is just an alternate way of saying a.1.

So to access a.foo may be written as a["foo"] or a[x] where x is variable containing the string "foo". Notice that the JSP 2.0 Expression Language adopted this syntax.

For Loop – Array

Syntax to loop an index over an array..

```
for (int i in array) {
```

```
    // i iterates 0..len-1
```

```
    // use array[i] to access the actual elements
```

```
}
```

I like the Perl foreach a little better, since it does the [i] for you

Form/Field/ImgAccess

Suppose we have a tag with a name=foo that contains a field with name=bar

Can refer to the field as document.foo.bar

Can refer to the value of the field as document.foo.bar.value

For checkbox, field.checked is true if the checkbox is checked

Names also work for images , and the src may be accessed as document.imgname.src

document.forms, document.images – these are arrays of the elements, numbered by the order they occur in the document.

Id access

Names do not work for all HTML elements

However, any element may have an "id" and we can use that to get a pointer to that element on the JavaScript side.

HTML – the <div> tag is a way of identifying a section in the HTML

```
<div id="foo"></div>
```

JavaScript

```
var node = document.getElementById("foo");  
if (node != null) { ...
```

Node HTML

Given a pointer to a node, can manipulate the HTML

This is the same Document Object Model (DOM) tree-of-nodes that is used in XML

DOM way to add text after a node...

```
node.childNodes[0].data = text;
```

There are lots of other DOM functions that can be used to edit the DOM

innerHTML way – this is much easier, although it does not have quite the official support as the DOM way. However, it works on all modern browsers. The text can include its own tags.

```
node.innerHTML = text;
```

onclick="foo():"

Calls a function when a link or button or some text is clicked...

onclick="callJSFunction();" – works in ...

```
<input type=submit onclick="foo();">
```

```
<button onclick="foo();">Button</button>
```

```
<a href=bar.html onclick="foo(); return false;">foo</a> – the return false prevents the jump to the next page
```

onsubmit

```
<form onsubmit="foo(); return false;">
```

Runs on form submit – submit button or return-key

The "return false;" prevents the actual submit back to the server (some older browsers do the submit anyway).

text – onchange, onkeypress

For text field and textareas, detect changes to the text

onchange is most widely supported

onkeypress, onkeydown, onkeyup are less standard, but allow you to detect individual keypresses. Onkeyup is handy, since it runs **after** the char has been added to the field

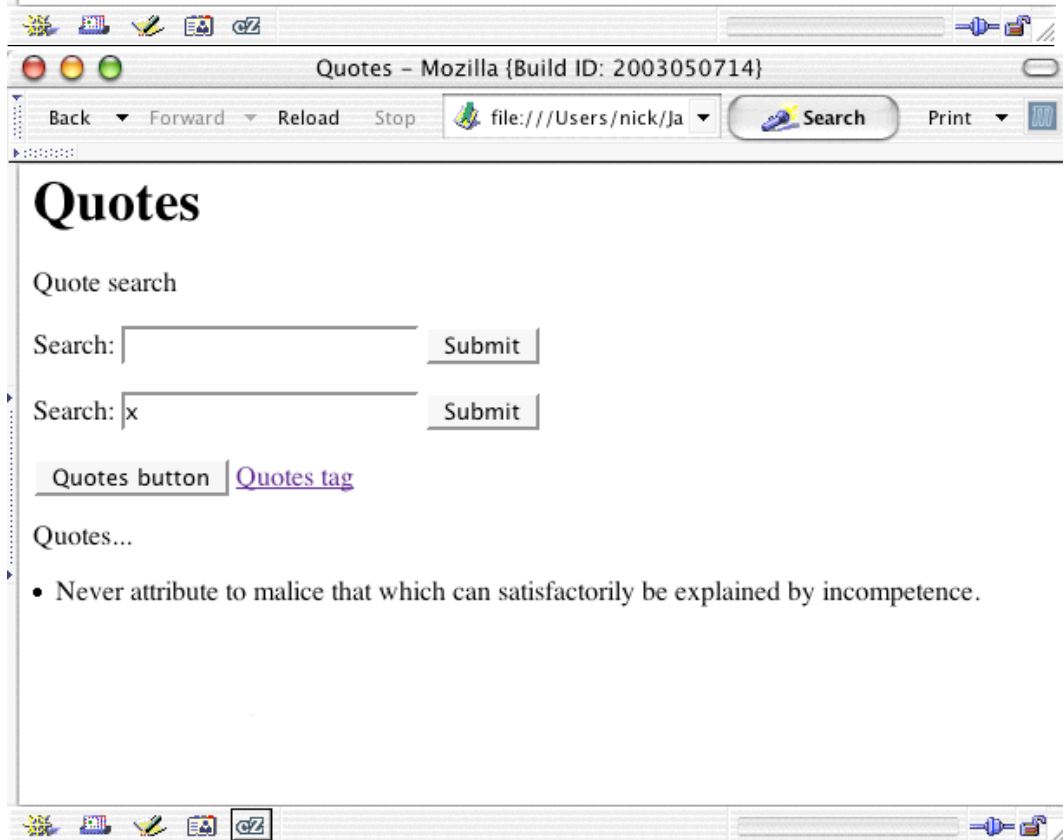
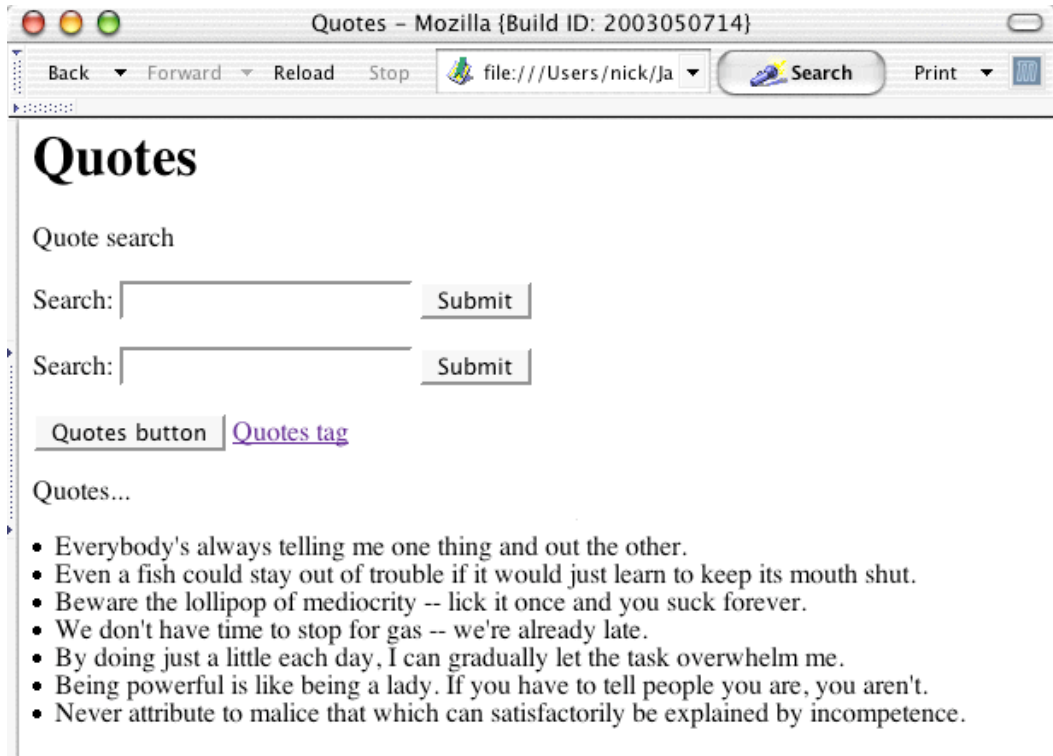
body – onload

```
<body onload="foo();">
```

Only works in the <body tag ... runs when the entire document has been loaded

Use to call some function that you want to run after everything is loaded

Quotes Example



Quotes Code

```
<html>
<head>
<title>Quotes</title>

<script language="JavaScript">

// We allocate a global array and fill it with the quote data.

lines = new Array();

lines.push("Everybody's always telling me one thing and out the other.");
lines.push("Even a fish could stay out of trouble if it would just learn to
keep its mouth shut.");
lines.push("Beware the lollipop of mediocrity -- lick it once and you suck
forever.");
lines.push("We don't have time to stop for gas -- we're already late.");
lines.push("By doing just a little each day, I can gradually let the task
overwhelm me.");
lines.push("Being powerful is like being a lady. If you have to tell people
you are, you aren't.");
lines.push("Never attribute to malice that which can satisfactorily be
explained by incompetence.");

// Search for an element with the given id
// and set its innerHTML to the given text.
function setText(id, text) {
    var node = document.getElementById(id);
    if (node != null) {
        node.innerHTML = text;
        //node.childNodes[0].data = text; // alternative for some simple tags
    }
}

// Given the name of a form, access the target field
// within that form and using its target text, generate
// the quote list and put it into the result tag.
function setQuotes(form_name) {
    // cute: use [] instead of . to access the form by name
    var target = document[form_name].target.value;
    //alert(target);
    var contents = "";
    target = target.toLowerCase();
    for (var i in lines) {
        if (lines[i].toLowerCase().indexOf(target)!=-1) {
            contents = contents + "<li>" + lines[i] + "\n";
        }
    }
    setText("result", contents);
}

</script>
</head>
```

```

<!-- once the whole body is loaded, run setQuotes once -->
<body onload="setQuotes('form1');" >

<h1>Quotes</h1>

<p>Quote search

<!--
  For this form, we call setQuotes('form1') when the form is submitted.
  The return false inhibits the actual form submit.
-->
<form name=form1 onsubmit="setQuotes('form1'); return false;" >
Search: <input type=text name=target >
<input type=submit value=Submit>
</form>

<!--
  For this first form, we setQuotes() on every keystroke (onKeyUp).
  We also use the ordinary submit in case they paste something in without
  touching a key.
-->
<form name=form2 onsubmit="setQuotes('form2'); return false;" >

Search: <input type=text name=target onKeyUp="setQuotes('form2');" >
<input type=submit value=Submit>

</form>

<p>
<!-- can use button outside of a form to trigger things -->
<button onclick="setQuotes('form1');" >Quotes button</button>

<!-- can use an a tag, but need the return false so it doesn't
  jump to the href -->
<a href onclick="setQuotes('form1'); return false;" >Quotes tag</a>

<p>Quotes...

<!-- the div we refer to from the script to store the output -->
<div id=result></div>

</body>

```