

Lecture

JavaScript and Browser Manipulation

This content is protected and may not be shared, uploaded, or distributed.

Testing the Speed of Your JavaScript (2009)

CSCI 571 Home Page - Mozilla Firefox

File Edit View History Bookmarks Tools Help

Gmail - Inbox (1... Department Of ... CSCI 571 ... Learning JavaScript... JavaScript Obj...

 SunSpider JavaScript Benchmark Results

Run Again

<http://www2.webkit.org/perf/sunspider-0.9/sunspider-results.htm>
(You can bookmark this results URL for later comparison.)

To compare to another run, paste a saved result URL in the text field below and press enter:

RESULTS (means and 95% confidence intervals)

	Total:	5103.6ms +/- 15.7%
3d:	619.2ms +/- 11.3%	
cube:	226.0ms +/- 18.5%	
morph:	187.0ms +/- 6.4%	
raytrace:	206.2ms +/- 20.4%	
access:	768.6ms +/- 18.3%	
binary-trees:	84.4ms +/- 49.8%	
fannkuch:	360.8ms +/- 16.7%	
nbody:	213.6ms +/- 18.9%	
nsieve:	109.8ms +/- 32.3%	
bitops:	513.2ms +/- 13.1%	
3bit-bits-in-byte:	106.8ms +/- 8.8%	
bits-in-byte:	144.0ms +/- 11.8%	
bitwise-and:	102.6ms +/- 41.6%	
nsieve-bits:	159.8ms +/- 8.2%	
controlflow:	63.8ms +/- 31.8%	
recursive:	63.8ms +/- 31.8%	
crypto:	338.8ms +/- 23.4%	

Done 0.419s YSlow

Firefox, v 3.0
5103ms

SunSpider JavaScript Benchmark Results

File Edit View History Bookmarks Window Help

http://www2.webkit.org/perf/sunspider-0.9/sunspider-results.htm

 SunSpider JavaScript Benchmark Results

Run Again

<http://www2.webkit.org/perf/sunspider-0.9/sunspider-results.htm?%7B%23d-c>
(You can bookmark this results URL for later comparison.)

To compare to another run, paste a saved result URL in the text field below and press enter:

RESULTS (means and 95% confidence intervals)

	Total:	6202.6ms +/- 7.2%
3d:	786.8ms +/- 18.6%	
cube:	257.6ms +/- 4.5%	
morph:	258.0ms +/- 59.0%	
raytrace:	271.2ms +/- 2.5%	
access:	984.8ms +/- 13.4%	
binary-trees:	148.4ms +/- 59.1%	
fannkuch:	424.0ms +/- 2.2%	
nbody:	303.6ms +/- 26.0%	
nsieve:	108.8ms +/- 5.3%	
bitops:	641.8ms +/- 18.0%	
3bit-bits-in-byte:	115.2ms +/- 42.8%	
bits-in-byte:	166.8ms +/- 46.7%	
bitwise-and:	196.2ms +/- 0.7%	
nsieve-bits:	163.6ms +/- 14.9%	
controlflow:	162.6ms +/- 46.4%	
recursive:	162.6ms +/- 46.4%	
crypto:	409.8ms +/- 1.2%	
aes:	139.4ms +/- 0.5%	

Safari v 3.2
6202ms

SunSpider JavaScript Benchmark Results - Windows Internet Explorer

File Edit View Favorites Tools Help

http://www2.webkit.org/perf/sunspider-0.9/sunspider-results.htm?%7B%23d-c

 SunSpider JavaScript Benchmark Results

Run Again

<http://www2.webkit.org/perf/sunspider-0.9/sunspider-results.htm?%7B%23d-c>
(You can bookmark this results URL for later comparison.)

To compare to another run, paste a saved result URL in the text field below and press enter:

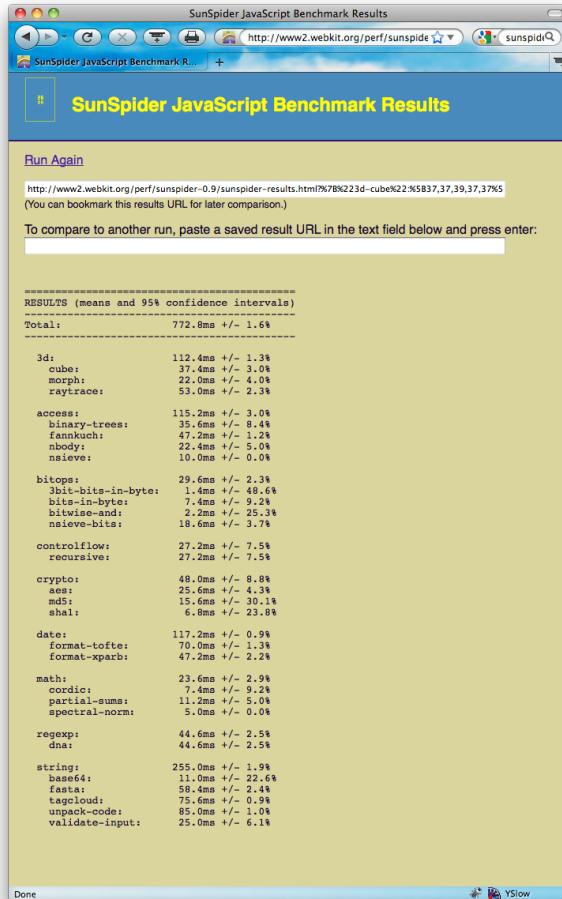
RESULTS (means and 95% confidence intervals)

	Total:	45928.2ms +/- 3.8%
3d:	2066.0ms +/- 1.5%	
cube:	556.4ms +/- 4.7%	
morph:	637.6ms +/- 1.4%	
raytrace:	872.0ms +/- 1.0%	
access:	2690.6ms +/- 6.7%	
binary-trees:	656.4ms +/- 3.6%	
fannkuch:	1018.6ms +/- 21.2%	
nbody:	515.6ms +/- 0.1%	
nsieve:	500.0ms +/- 17.6%	
bitops:	2175.0ms +/- 4.3%	
3bit-bits-in-byte:	565.6ms +/- 4.5%	
bits-in-byte:	506.4ms +/- 3.5%	
bitwise-and:	600.0ms +/- 8.8%	
nsieve-bits:	503.0ms +/- 1.7%	
controlflow:	681.2ms +/- 3.8%	
recursive:	681.2ms +/- 3.8%	
crypto:	1480.8ms +/- 6.3%	
aes:	534.2ms +/- 1.7%	
md5:	455.2ms +/- 0.5%	

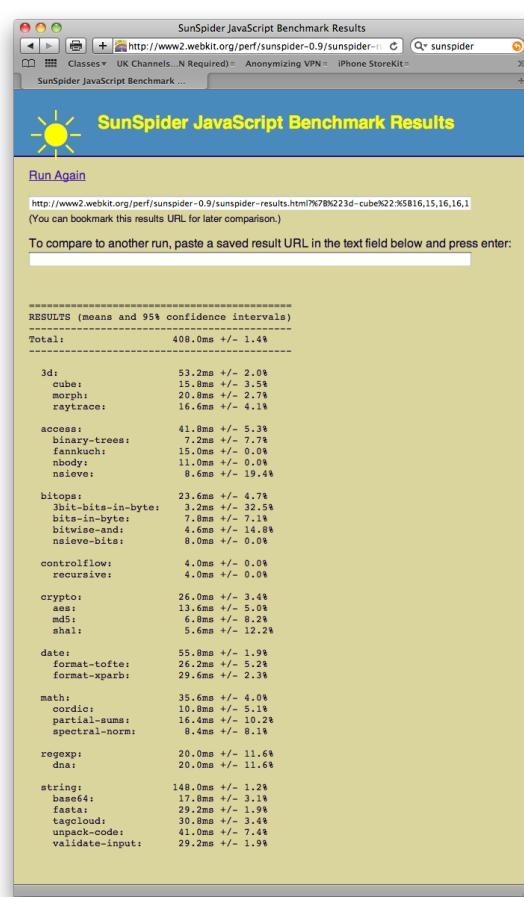
Done Internet 100%

I.E. 7.0
45,928ms

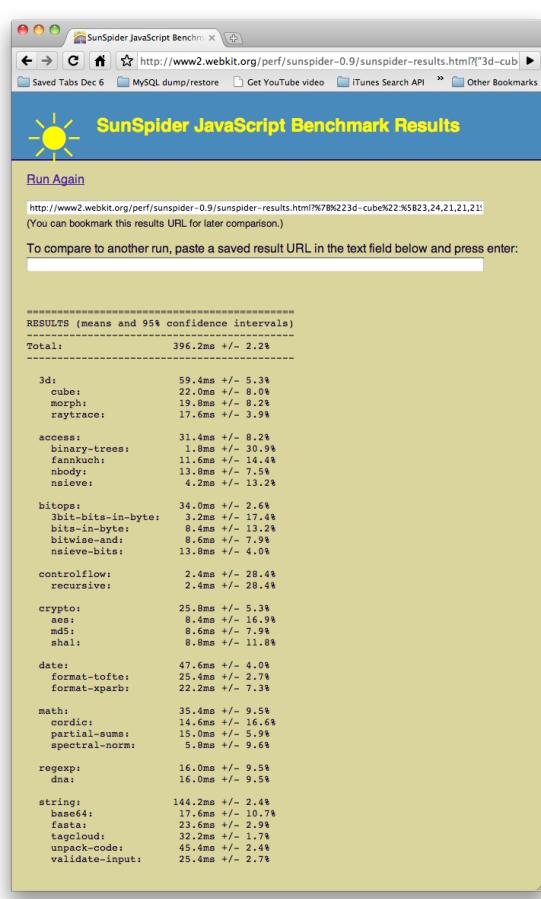
Testing the Speed of Your JavaScript (2010)



Firefox, v 3.6
772 ms

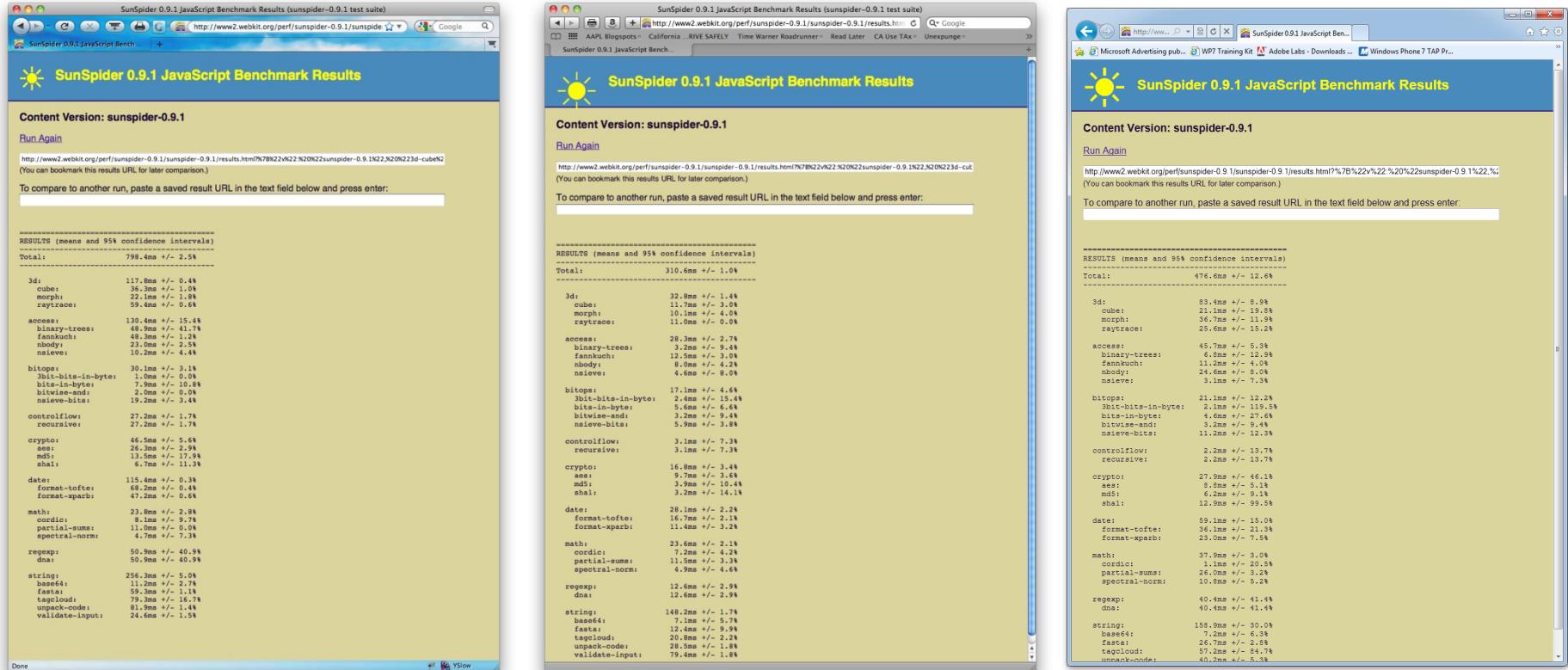


Safari v 4.0
408 ms

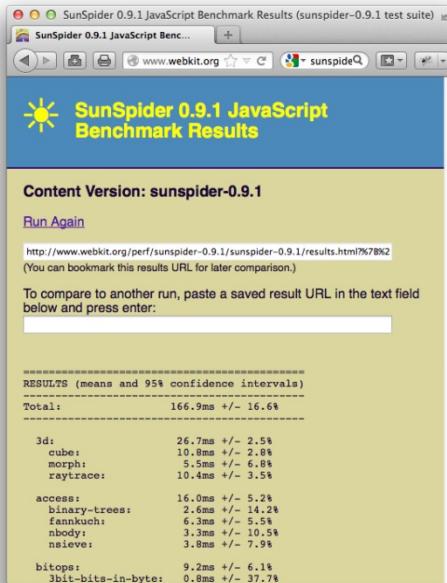


Chrome v 4.0
396 ms
Chrome v 14.0
214 ms

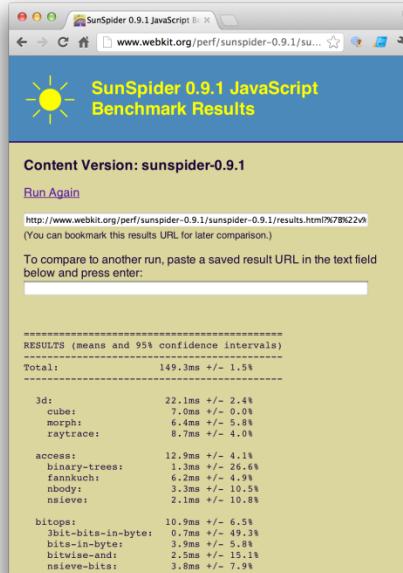
Testing the Speed of Your JavaScript (2011)



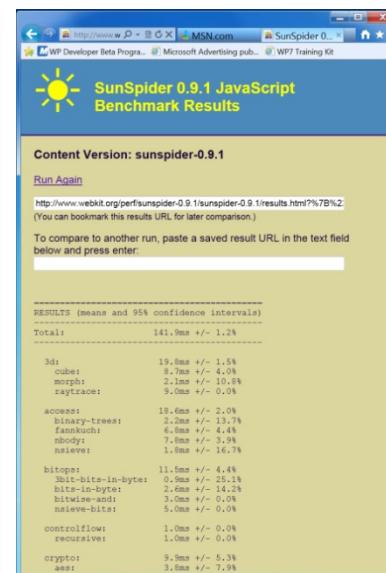
Testing the Speed of Your JavaScript (2012)



Firefox 16
166 ms



Chrome 21.0
149 ms



Internet Explorer 9.0
141 ms



Safari 6.0
138 ms

Summary
over the
years → 2009

2009

Firefox
5,103

Safari
6,202

I.E.
45,928

214

→ 2010

772

408

2011

187

198

314

2012

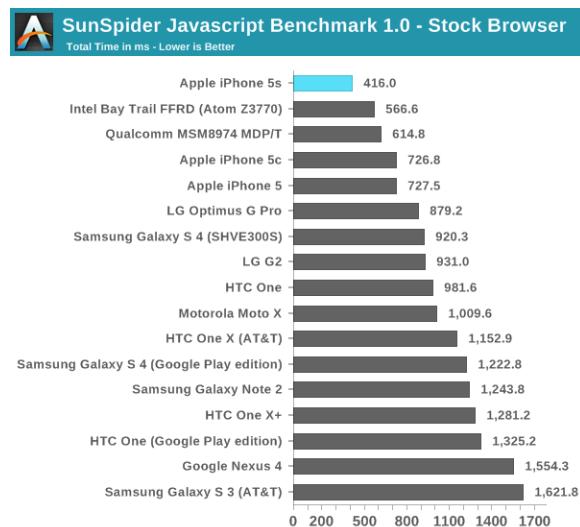
166

138

141

149

Mobile Speed of JavaScript (2012-2014)



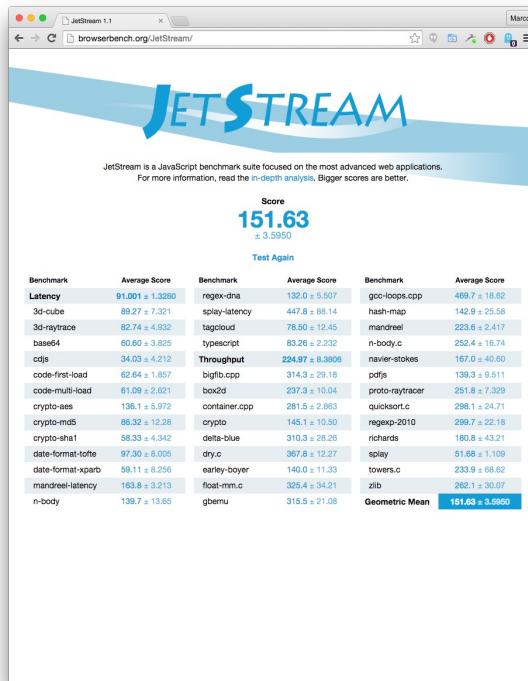
Safari, iPhone 5S
416ms

Safari, iPhone 6
352ms

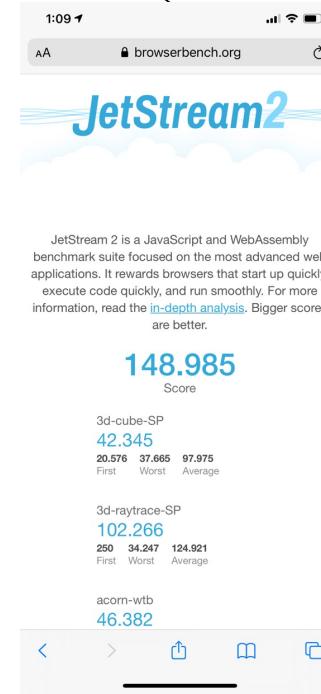
JetStream (2015-2018) – JetStream2 (2020-2022)



Safari 9, iPhone 6s
111.75 score
Safari 10, iPhone 7
164.79 score
Safari 11, iPhone X
216.74 score
Safari 12, iPhone XS
261.75



Safari 10, OS X
155.76 score
Safari 11, OS X
246-255.60
Chrome 45, OS X
151.62 score

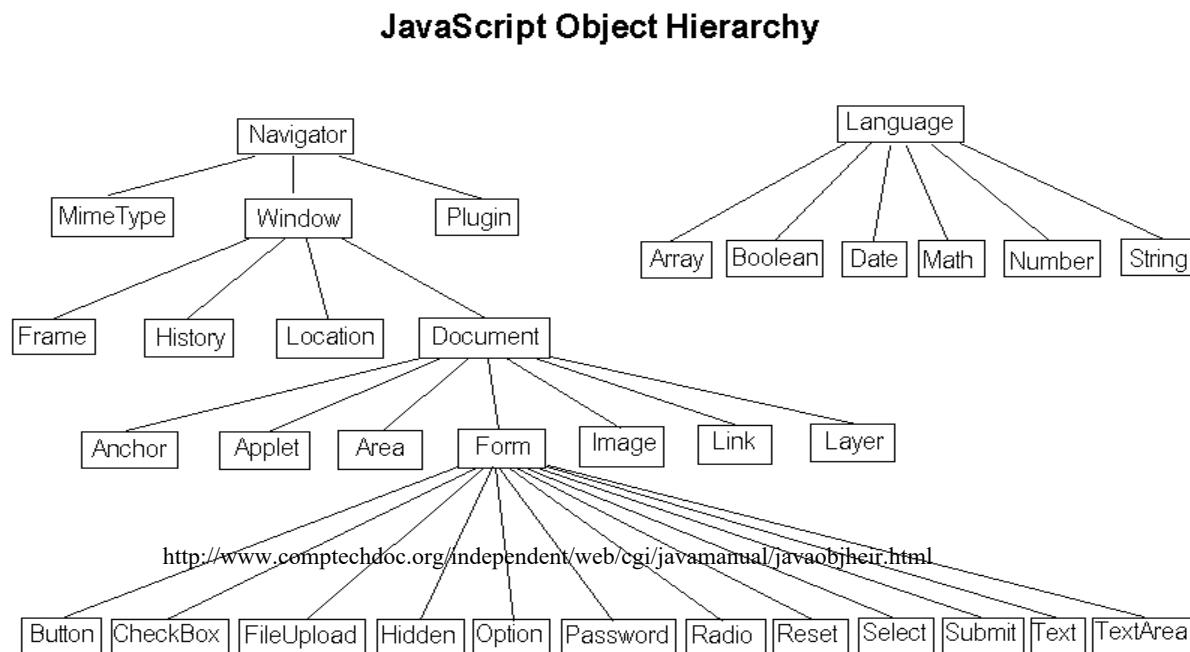


Safari, iPhone 11 Max Plus,
iOS 14, 148.98 score
Safari, iPhone 12 Max Plus,
iOS 14.6, 181.65 score
Safari, iPhone 13 Max Plus
iOS 15.5, 227.75 score
Safari, iPhone 14 Max Plus
iOS 16, 272.44 score
Firefox 81, Mac mini,
macOS 10.15, 90 score

browserbench.org

Javascript Object Hierarchy

JavaScript contains a set of built-in objects related to the browser. Each object includes a set of properties and methods



The Window Object

- The **window** object is the highest level built-in JavaScript object; it corresponds to the browser window
- Some properties of the Window object include:
 - `window.closed` – a boolean indicating if the window is closed or not
 - `window.history` – returns the history object
 - `window.location` – returns the location object
 - `window.navigator` – returns the navigator object
 - `Window.parent` – returns the parent window of the current window
 - `Window.self` – returns the current window
 - `Window.status` – sets the text in the statusbar window
- Some methods of the Window object include:
 - `open()` – opens a new window
 - `blur()` – removes focus from the current window
 - `close()` – closes the current window
 - `focus()` – sets the focus to the current window
 - `resizeBy()` – resizes the window by the specified pixels
- For a complete list see, e.g., http://www.w3schools.com/jsref/obj_window.asp

Examples Using the Window Object

- To create a new window that shows only the toolbar and status bar and is resizable

```
window.open("newURL", "New Window", "toolbar,status,resizable")
```

- The height and width defaults are the same as the browser
- The new window is positioned in the upper left-hand corner of the screen
- A call to window.open() returns a value of the new window's object; this should always be assigned to a variable, e.g.

```
newWindow = window.open("", "")  
if (newWindow != null) {  
    newWindow.document.write("<HTML><HEAD>  
    <TITLE>Hi</TITLE></HEAD><BODY>Hello  
    World</BODY></HTML>") }
```

- Try https://csci571.com/examples/js/js_22_2.html

Parameters of `windows.open()`

- The `open()` method includes three top level parameters:

```
newWin = open(url, name, [features, [replace]]);
```

- **newWin**: is a reference to the new window object
- **url**: a string URL value to be fetched by the new window (can be empty)
- **name**: this name can be used with the HTML target attribute (do not use spaces in the name)
- **features**: a string that specifies the desired browser features (syntax: feature=value)
- **replace**: allows new entries to be made to the browser history

More on the features argument of the windows.open method

- The ‘features’ string is a comma separated list of items (original list)
 - toolbar Back, Forward Buttons
 - location the URL input field
 - directories What's New, What's Cool, etc
 - status browser status line
 - menubar browser menu bar
 - scrollbars enables scrollbars when needed
 - resizable allows the window to be resizable
 - **width, height** window dimensions in pixels
- When the ‘features’ string is absent, the new browser window has all of the standard controls
- New, much larger set of recent ‘features’ available at:
<https://developer.mozilla.org/en-US/docs/Web/API/Window/open>
- Chrome does not support most of the ‘features’
- Support varies by browser. Width & Height supported by all browsers

Sample Code to open a window with attributes:

width=640, height=480, URL bar, browser toolbar, Menus, Directory buttons, Window status bar, window resizable, window scrollable

- **Here is the code**
- ```
window.open('http://www.usc.edu', 'Sam',
'location=yes,toolbar=yes,menubar=yes,directories=yes,status=yes,resizable=y
es,scrollbars=yes,height=480,width=640', false);
```
- **Three possible ways to invoke the code above:**
  1. Triggered at page load time
    - `<body onload="[open new window code];"> </body>`
  2. Triggered by clicking on a hyperlink
    - `<a href="javascript:[open new window code]; void 0;">click me to open
a new window!</a>`
  3. Triggered by clicking on a button
    - `<input type="button" value="click me to open a new
window" onclick="[open new window code];" />`

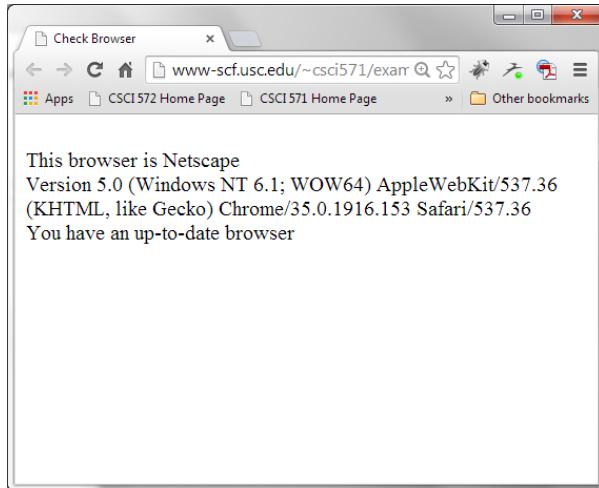
# Determining the Browser

- navigator is a built-in object with properties that describe the browser
  - **navigator.appName** is a string with the browser name
  - **navigator.appVersion** is a string with the version number
    - to determine the correct version, you may need to convert from string to number; parseFloat returns a number from a string, and ignores any part of the string after the number
  - **navigator.cookieEnabled** determines whether cookies are enabled
  - **navigator.language** returns the language of the browser
  - **navigator.userAgent** returns the user-agent header sent by the browser

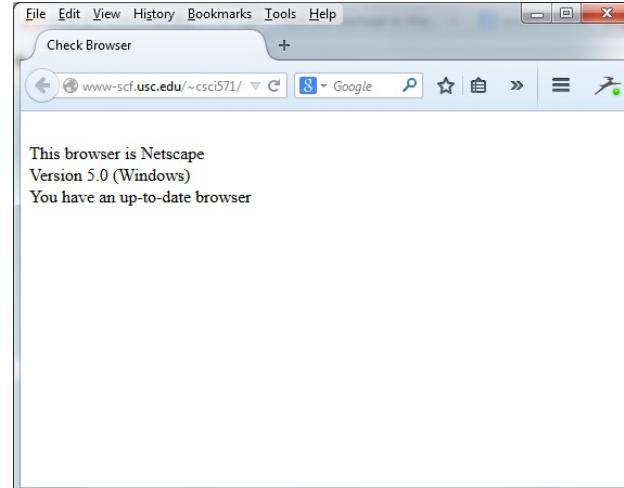
# Determining the Browser Version

```
<HTML>
<HEAD>
<TITLE>Check Browser</TITLE>
</HEAD>
<BODY>
<SCRIPT language=JavaScript>
document.write("
 This browser is " + navigator.appName);
document.write("
 Version " + navigator.appVersion);
if (parseFloat(navigator.appVersion) >= 7)
 { document.write("
 You have an up-to-date browser"); }
</SCRIPT>
</BODY>
</HTML>
```

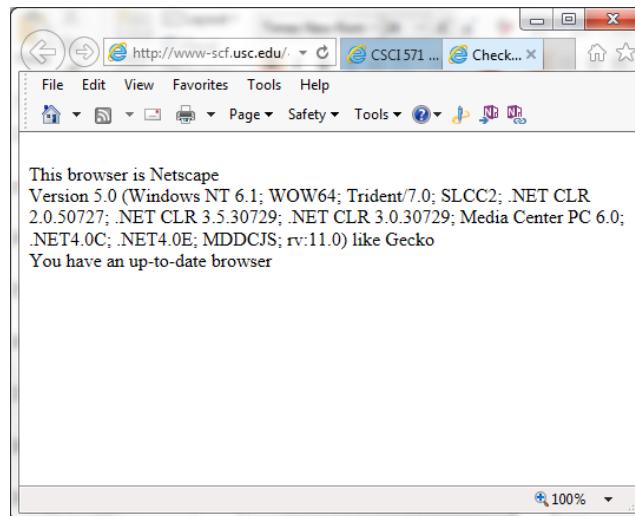
# Example: Browser Output



Chrome



Firefox



Internet Explorer

It is an historical anomaly that they all return Netscape; to determine the browser the appname value must be carefully analyzed, or use userAgent

# Avoiding Errors

- Using newer functionalities of JavaScript, not fully supported by all browsers, will need an extra check to avoid run-time errors.
- One can refer to any name in a conditional and if it is undefined the conditional returns false

```
 newItem = something;
```
- Can also be used to test for new JavaScript APIs

produces an error if something is undefined

```
if (something) { newItem = something; }
```

does not produce an error, but only changes newItem if something exists

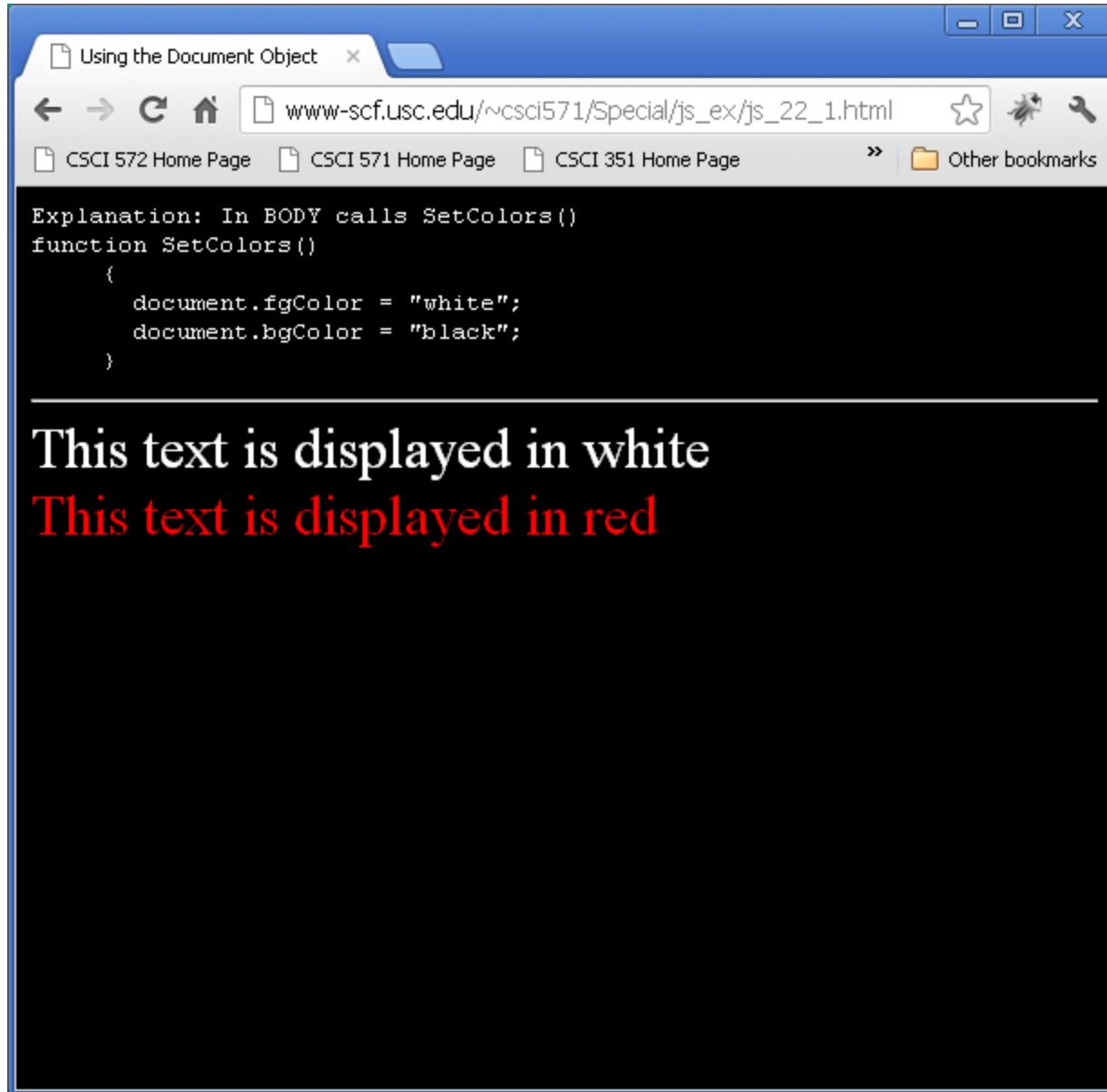
# The Document Object

- Each HTML document loaded into a browser window becomes a Document object.
- The Document object provides access to all HTML elements in a page, from within a script
- Some properties of the document object include:
  - `Document.anchors` - returns a collection of all anchors in the document
  - `Document.applets` - returns a collection of all applets in the document
  - `Document.body` - returns the body element of the document
  - `Document.cookie` - returns all name/value pairs of cookies in the document
  - `Document.forms` - returns a collection of all forms in the document
  - `Document.images` - returns a collection of all the images in the document
  - `Document.lastModified` - returns the date/time the document was last modified
- Some methods of the document object include:
  - `Document.close()` - closes the output stream previously opened
  - `Document.open()` - opens an output stream to collect the output from `document.write`
  - `Document.write()` - writes HTML expressions or JavaScript to a document

# Setting Document Object Attributes

```
<HTML><HEAD><TITLE>Using the Document Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function SetColors() {
 document.fgColor = "white";
 document.bgColor = "black"; }
</SCRIPT>
</HEAD>
<BODY><SCRIPT LANGUAGE="JavaScript">
SetColors()
</SCRIPT>
This text is displayed in white

This text is displayed in
 red
</BODY></HTML>
```



# The Location Object

- A reference to the URL of the current document
- Properties: hash, host, hostname, top, status, defaultStatus, window
- No Methods and no Event Handlers
- Example

`http://www.site.com:8080/ads/newitems.html#widget1`

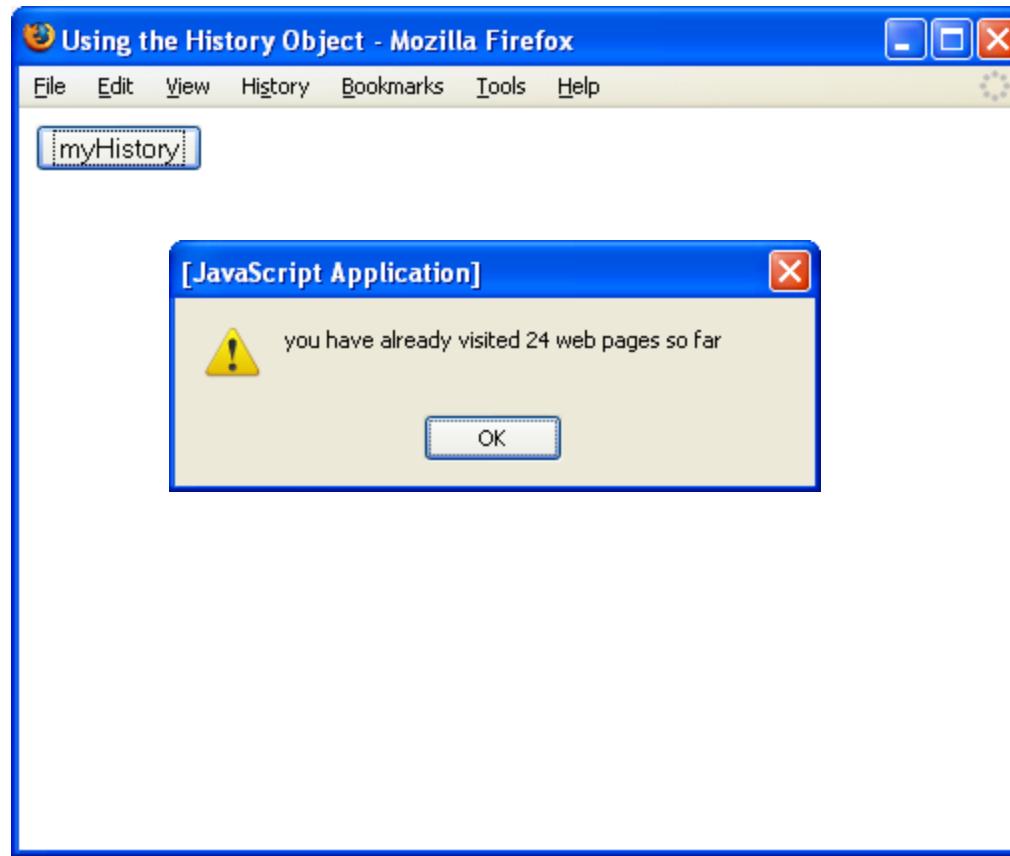
<b>Property</b>	<b>Value</b>
protocol	“http”
hostname	“www.site.com”
port	“8080”
host	“www.site.com:8080”
pathname	“/ads/newitems.html”
hash	“#widget1”
href	<code>http://www.big.com:8080/ads/newitems.html#widget1</code>

# History Object Example

- The history object contains the URLs visited by the user within a browser window
- The history object is part of the window object

```
<HTML><HEAD><TITLE>Using the History Object</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function showHistNumber() {
 var histNumb = window.history.length;
 alert("you have already visited " + histNumb + " web pages
so far");
}
</SCRIPT></HEAD>
<BODY><FORM>
<INPUT TYPE="button" NAME="history" VALUE="myHistory"
onClick="showHistNumber()">
</FORM></BODY></HTML>
```

# Browser Output



# Image Object

- Each image in an HTML document has an associated JavaScript object
  - the properties of the object include
    - **width** and **height** in pixels
    - **src**, URL of the image file
      - Assigning to the src changes the image
    - **complete**, true after an image finishes loading
    - **alt**, text string returned if image is unavailable
- To manipulate an image in JavaScript you can refer to it EITHER by its NAME attribute or by a built-in array of images that is automatically created by JavaScript, e.g.
  - **document.images.name**
  - **document.images[index]** where index is either a number or a string containing the name of the image
- An image object can also be created in JavaScript

```
var myImage = new Image();
```

  - this can be used to load images into a document that are not visible in the displayed document
  - pre-fetching to make images appear immediately; coming up is an example

# Example Working with Radio Buttons

```
<HTML><HEAD><TITLE>Beatle Picker</TITLE>
<SCRIPT LANGUAGE="javaScript">
function processData(form) {
for(var j = 0 ; j < form.Beatles.length ; ++j) if(form.Beatles[j].checked) {
 var RadioValue = form.Beatles[j].value; break;}
alert("You selected " + form.name + " form and Beatle " + "\n" + RadioValue
 + " and Beatle song " + document.forms[0].song.value);}
</SCRIPT></HEAD><BODY><FORM NAME="Abbey Road">
Choose your favorite Beatle:

<INPUT TYPE=radio NAME="Beatles" VALUE="Lennon" CHECKED="true">John

<INPUT TYPE=radio NAME="Beatles" VALUE="McCartney">Paul

<INPUT TYPE=radio NAME="Beatles" VALUE="Harrison">George

<INPUT TYPE=radio NAME="Beatles" VALUE="Starr">Ringo

Enter the name of your favorite Beatles song:

<INPUT TYPE="text" NAME="song"><P>
<INPUT TYPE="button" NAME="process" VALUE="process Request...">
onClick="processData(this.form)">
</FORM></BODY></HTML>
```

See samples under "JavaScript Objects at  
<http://csci571.com/examples.html#js>

# Beatle Picking Form

The screenshot shows a web browser window titled "Beatle Picker". The URL in the address bar is "www-scf.usc.edu/~csci571/Special/js\_ex/js\_22\_6.html". The page content is as follows:

**Forms Processing Example: Beatle Picker**

Choose your favorite Beatle:

John  
 Paul  
 George  
 Ringo

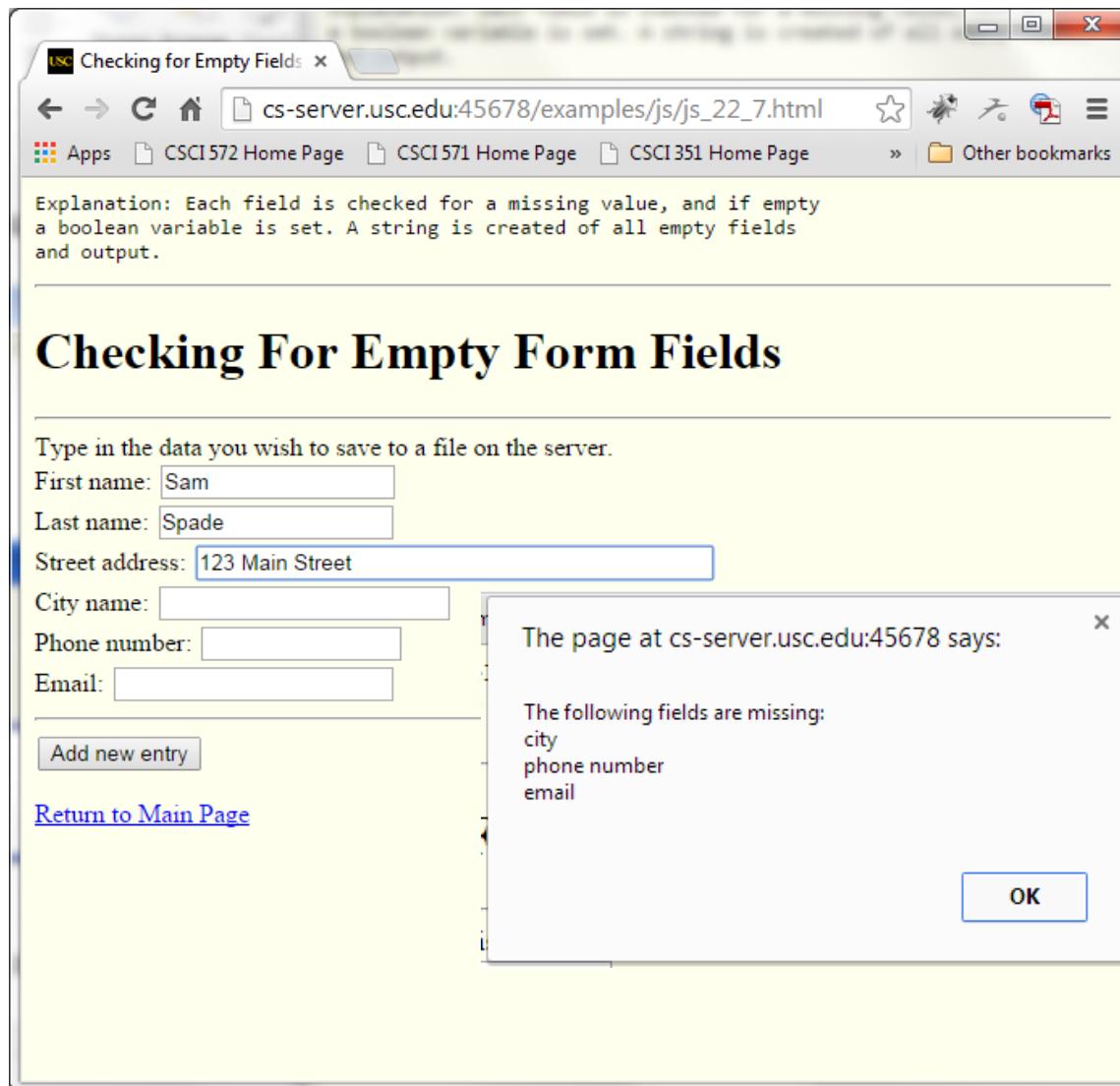
Enter the name of your favorite Beatles song:

let it be

[Return to Main Page](#)

A modal dialog box is displayed in the center of the page, titled "The page at www-scf.usc.edu says:". The message in the dialog is: "You selected Lennon and Beatle song let it be". There is an "OK" button at the bottom right of the dialog.

# Checking for Empty Form Fields



# Form Elements Checking (cont'd)

```
<html><head><title>Checking for Empty Fields</title><script language=JavaScript>
 function checkEmpty() { /* This function checks all of the fields of the form and
notifies the client
 which if any, form fields are empty. It returns a 1 if all the fields
are full, and a 0 otherwise. */
var firstname_filled = lastname_filled = streetaddress_filled = city_filled =
phonenumbers_filled = youremail_filled = false;
var blank="";
 if (document.myform.firstname.value != blank) firstname_filled=true;
 if (document.myform.lastname.value != blank) lastname_filled=true;
 if (document.myform.streetaddress.value != blank) streetaddress_filled=true;
 if (document.myform.city.value != blank) city_filled=true;
 if (document.myform.phonenumber.value != blank) phonenumbers_filled=true;
 if (document.myform.youremail.value != blank) youremail_filled=true;
if ((firstname_filled) && (lastname_filled) && (streetaddress_filled) && (city_filled) &&
(phonenumbers_filled) && (youremail_filled)) { alert("No missing fields");
return(true); }
 else /* check which fields are missing */
var alertstring="The following fields are missing:\n";
 if (!firstname_filled) alertstring=alertstring + "first name\n";
 if (!lastname_filled) alertstring=alertstring + "last name\n";
 if (!streetaddress_filled) alertstring=alertstring + "street address\n";
 if (!city_filled) alertstring=alertstring + "city\n"; if (!phonenumbers_filled)
alertstring=alertstring + "phone number\n";
if (!youremail_filled) alertstring=alertstring + "email\n";
 alert(alertstring); return(false); } }</script></head>
```

# Form Elements Checking (cont'd)

```
<BODY bgcolor="#ffffee">
<pre>Explanation: Each field is checked for a missing value, and if empty a
boolean variable is set. A string is created of all empty fields and output.
</pre><hr>
<H1>Checking For Empty Form Fields</H1><HR>
Type in the data you wish to save to a file on the server
<FORM NAME="myform">
 First name: <INPUT type="text" name="firstname" size=16 >

 Last name: <INPUT type="text" name="lastname" size=16 >

 Street address: <INPUT type="text" name="streetaddress" size=41 >

 City name: <INPUT type="text" name="city" size=21 >

 Phone number: <INPUT type='text' name='phonenumbers' size=13 >

 Email: <INPUT type='text' name='youremail' size =20>
 <hr>
 <INPUT type='button' value='Add new entry'
onClick="checkEmpty()" >

</FORM>
</BODY>
</HTML>
```

Note: **HTML5** provides the **REQUIRED** attribute for many Elements

Use libraries to support older browsers. See:

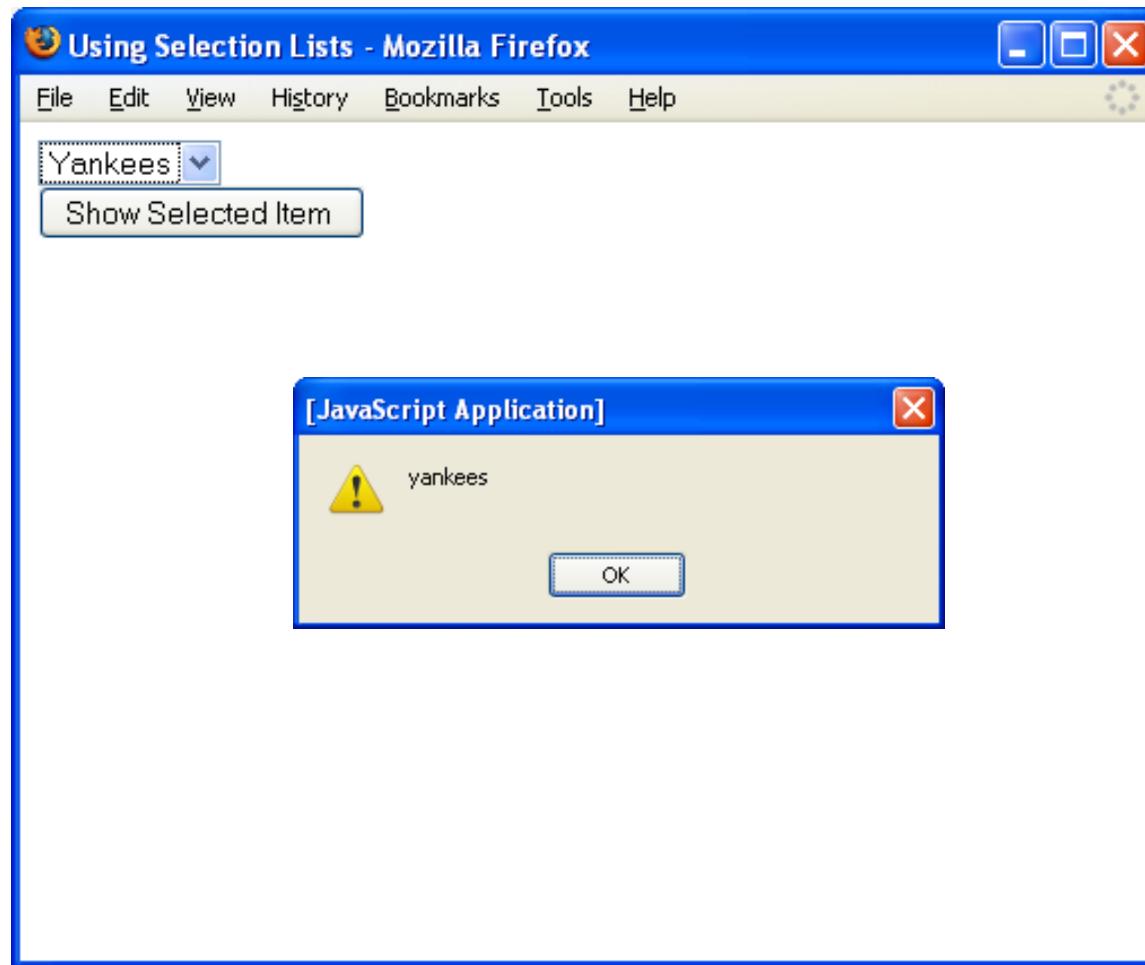
<http://www.useragentman.com/blog/2010/07/27/creating-cross-browser-html5-forms-now-using-modernizr-webforms2-and-html5widgets-2/>

# Example Selecting Using Selection Lists

```
<HTML><HEAD><TITLE>Using Selection Lists</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function getSelectValue(selectObject) {
 return
 selectObject.options[selectObject.selectedIndex].value
}
</SCRIPT></HEAD>
<BODY>
<FORM NAME="myform">
<SELECT NAME="teams">
<OPTION value="dodgers">Dodgers
<OPTION value="yankees">Yankees
<OPTION value="angels">Angels
</SELECT>

<INPUT TYPE="button" value="Show Selected Item "
onClick="alert(getSelectValue(this.form.teams))">
</FORM>
</BODY></HTML>
```

# Browser Output

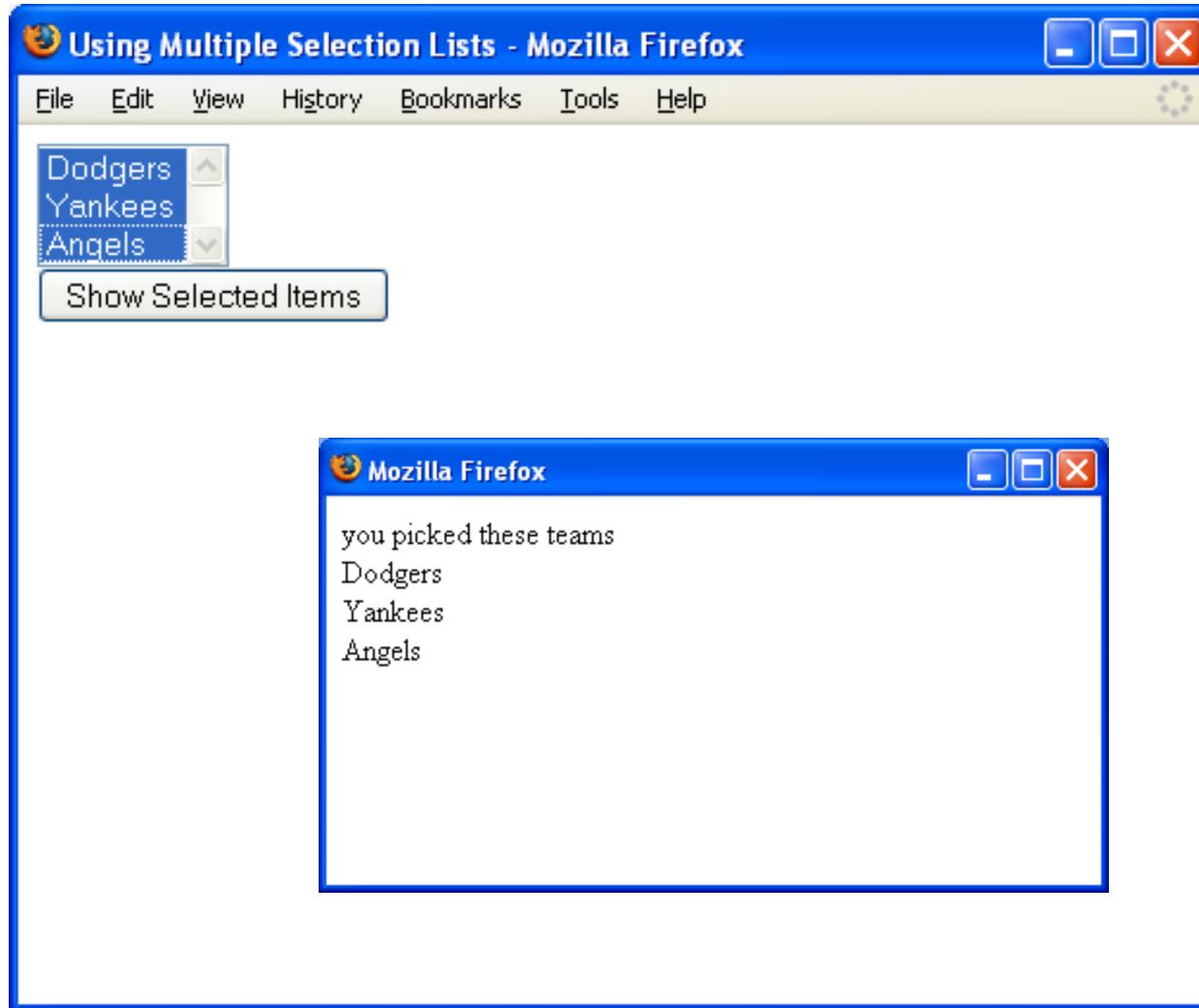


# Example Selection Using Multi-selection Lists

```
<HTML><HEAD><TITLE>Using Multiple Selection Lists</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function displaySelectionValues(objectName) { var ans = ""
for (var i = 0; i < objectName.length; i++) {
 if (objectName.options[i].selected) {
 ans += objectName.options[i].text + "
" }
myWin = window.open("", "Selections",
"height=200,width=400")
myWin.document.write("you picked these teams
")
myWin.document.write(ans) }
</SCRIPT></HEAD><BODY>
<FORM NAME="myform">
<SELECT NAME="teams" size=3 multiple>
<OPTION value="dodgers">Dodgers<OPTION value="yankees">Yankees
<OPTION value="angels">Angels </SELECT>

<INPUT TYPE="button" value="Show Selected Items"
onClick="displaySelectionValues(this.form.teams)">
</FORM></BODY></HTML>
```

# Browser Output



# JavaScript: addEventListener

- The `addEventListener()` method attaches an event handler to the specified element.
  - it does not overwrite existing event handlers.
  - You can add many event handlers to one element.
  - You can add event listeners to any DOM object not only HTML elements. i.e the window object

- **Example: Add an event listener that fires when a user clicks a button:**

```
document.getElementById("myBtn").addEventListener("click", displayDate);
```

- **Example: Use the alert function to print "Hello World!" when the user clicks on an element:**

```
element.addEventListener("click", function(){ alert("Hello World!"); });
```

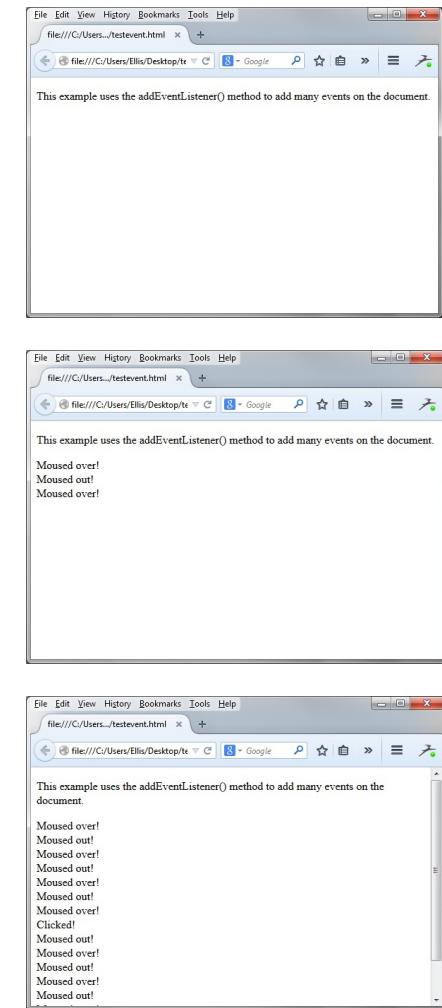
- **Example: on a click event change the associated text to Hello World; note the reference to an external function**

```
document.addEventListener("click", myFunction);
function myFunction() {
 document.getElementById("demo").innerHTML = "Hello World";
}
```

# Example: Adding 3 Events to a Document

```
<!DOCTYPE html><html><body>
<p>This example uses the addEventListener() method to add many events on the document.</p>
<p id="demo"></p>
<script>
document.addEventListener("mouseover", myFunction);
document.addEventListener("click", mySecondFunction);
document.addEventListener("mouseout", myThirdFunction);
function myFunction() {
 document.getElementById("demo").innerHTML += "Moused over!
"}
function mySecondFunction() {
 document.getElementById("demo").innerHTML += "Clicked!
"}
function myThirdFunction() {
 document.getElementById("demo").innerHTML += "Moused out!
"}
</script></body></html>
```

See: <http://csci571.com/examples/js/Add3Events.html>



# **JavaScript Language Issues**

# Outline

- Functions as first-class objects
- Functions as methods
- Functions as constructors
- Functions have unlimited arguments
- By-value vs. By-reference
- Data Types
- Strings
- Expressions and Operator Anomalies
- Regular Expressions

# Functions

- Functions are a type of object

```
function twice(x) { return x * x }
```

creates an object whose variable name is twice

- Therefore, functions can be treated as an object

- assigned to new variables

- stored in arrays

- assigned to properties of objects

- passed as arguments to functions

- Example of function names

```
var twice_again = twice;
```

```
var sum = twice(5) + twice_again(10);
```

# Methods of Functions

- One can associate a function with an object
- If `func()` is a function, and `obj` is an object,

`obj.method = func;` assigns the function as an object method

- To invoke the method, do

`obj.method()`

- The `this` keyword refers to object `obj`
- Example

```
function compute_area() { return this.length *
this.width}

var rect = new Object();

rect.length = 10; rect.width = 20;

rect.area = compute_area; //passing the method

var area = rect.area(); //invoke the method
```

## Special Purpose Functions - Constructors

- Used with new to initialize fields of an object
- Example

```
function Rectangle(l, w)
{ this.length = l; this.width = w;
this.area = compute_area; }

...

var rect1 = new Rectangle(150, 100);

var rect1_area = rect1.area;
```

- **this** refers to the newly created object
- Rectangle creates properties length, width, and method area

# **By-Value versus By-Reference**

## **By-Value**

## **By-Reference**

### **copy (as in variable assignment)**

value is copied      reference to value is copied

### **pass (as in an actual parameter to a formal parameter)**

a new copy of the value      reference to the value is passed  
is passed to the function      to the function

### **compare (as in a conditional clause)**

two different values      two references are compared  
are compared

# Handling Data Types

	Copied by	Passed by	Compared by
numbers	value	value	value
boolean	value	value	value
string	immutable	immutable	value
object	reference	reference	reference
array	reference	reference	reference
function	reference	reference	reference

- Changes to object properties are visible outside the function
- See “JavaScript: the Definitive Guide”:  
[http://docstore.mik.ua/orelly/webprog/javascript/ch11\\_02.htm](http://docstore.mik.ua/orelly/webprog/javascript/ch11_02.htm)

# JavaScript Strings

- Strings are **immutable**, once created they can never be changed
- You can **search** a string and **extract** substrings, but you **cannot modify** a string. And **cannot delete it!**
- “immutable” means that once you instantiate the object, you **can't change its properties.**
- So, when calling methods on a string, JavaScript will return the modified string, but it won't change the initial string
- Now this doesn't mean that you can't assign a new string object to the str variable. You just can't change the current object that str references.
- One generally builds a string through concatenation
- JavaScript runtimes include a **garbage collector**
- See: <https://javascript.info/garbage-collection>

```
var newWebPage = ""

newWebPage += "<HTML><HEAD>"

newWebPage += "<TITLE>A Sample Page</TITLE></HEAD>"

newWebPage += "<BODY>My Home Page</BODY>"

newWebPage += "</HTML>"
```

# Properties of Strings

- Strings have a length property

```
"Lincoln".length // result = 7
```

```
"Four score".length //result = 10
```

```
"One\nTwo".length // result = 7
```

```
"".length // result = 0
```

- Some String methods

```
string.toLowerCase(); string.toUpperCase()
```

```
string.indexOf(searchString [, startIndex]) //returns
index value of char within string where searchString
begins
```

```
string.charAt(index) //returns the one char at
position index
```

```
string.substring(indexA, indexB) //returns characters
of string between indexA and indexB
```

# Expressions and Operators

- Most of the C/C++ operators (lowest to highest order)

comma	,
assignment	= += -= *= /= %= <<= >>= >>>= &= ^=  =
conditional	? :
logical or	
logical and	&&
bitwise or	
bitwise xor	^
bitwise and	&
equality	== !=
comparison	< <= > >=
bitwise shift	<< >> >>>
addition/subtraction	+ -
multiply/divide	* / %
negation/increment	! ~ - ++ --
call, data structure()	[ ] .

# **Operator Anomalies**

- Implicit conversions

"a" \* "b" is illegal

"3" \* "5" is legal, result is 15, not "15"

- Plus used for concatenation

"a" + "b" is legal "ab"

"a" + 1 is legal "a1"

- Right associativity makes sense

i = j = k = m; is the same as i = (j = (k = m));

d = e ? f : (g ? h : (i ? j : k));

# Equality Operator

- **`== (!=)` returns true or false**
- **Recall**
  - numbers, strings, Booleans are compared by value
  - objects, arrays and functions are compared by reference
- **Implication**
  - to test if two objects have identical values, or to test if two arrays have identical values, check the fields element-by-element
- Some examples
  - "1" == 1 evaluates to true
  - true == 1 evaluates to true
  - false == 0 evaluates to true
- The identity (==>) operator behaves identically to the equality (==) operator **except** no type conversion is done, and the types must be the same to be considered equal, e.g., if `x = 5`, then
  - `x == 8` is false
  - `x == 5` is true
  - `x ==> "5"` is false
  - `x ==> 5` is true

# **Regular Expressions in JavaScript**

- Introduced in Javascript 1.2
  - [Netscape >=4 & IE >=5.5]
- Modeled after Perl's regular expressions
- There is a regular expression object, with properties and methods
- The constructor function for a regular expression is RegExp
- A regular expression object is created using the new operator
- See  
[http://www.w3schools.com/jsref/jsref\\_obj\\_regex.asp](http://www.w3schools.com/jsref/jsref_obj_regex.asp)

# Regular Expressions in JavaScript

- Constructing a regular expression
  - `var re = /ab+c/` initializes `re` to an expr
  - `re = new RegExp("ab+c")` compiles the expr
- **test** and **exec** are methods of object `RegExp`
  - `exec` executes a search for a match in a string; it returns an array
  - `test` tests for a match and returns true/false
  - Both `exec` and `test` set properties during execution
- **match** and **search** are methods of `String`
  - `match` executes a search for a match in a string and returns an array or null
  - `search` tests for a match in a string and returns the index or -1

# Regular Expression Examples

- Uses exec method to locate a string

```
<script language=javascript1.2>
myRe = /d(b+)d/g; //a d, 1 or more b, and a d
myArray = myRe.exec("cdbbdbsbz");
</script>
```

- An alternate approach

```
<script language=javascript1.2>
myArray = /d(b+)d/g.exec("cdbbdbsbz");
</script>
```

- An alternate approach

```
<script language=javascript1.2>
myRe = new RegExp ("d(b+)d", g:);
myArray = myRe.exec("cdbbdbsbz");
</script>
```

## Parenthesized Substring Matches

- Example that uses the replace method to switch the words

```
<script language=javascript1.2>
re = /(\w+) \s (\w+)/ ;
str = "John Smith";
newstr = str.replace(re, "$2, $1");
document.write(newstr)
</script>
```

This prints "Smith, John"

# Regular Expression Flags

- Regular expressions have two flags that permit global and case insensitive searching
- **g** stands for **global** match; **i** stands for **ignore case**
- Syntax

```
re = /pattern/[g|i|gi]
re = new RegExp("pattern", ['g'|'i'|'gi'])
```

- Example that creates a regular expression that looks for one or more characters followed by a space, throughout the string

```
<script language=javascript1.2>
re = /\w+\s/g;
str = "fee fi fo fum";
myArray = str.match(re);
document.write(myArray);
</script>
```

- Output is ["fee ", "fi ", "fo "]

## Example

- User enters a **phone number**, click Test, and script checks validity

```
<script language=javascript1.2>
re = /\(?(\d{3})\)?([-\/\.\.])\d{3}\1\d{4}/
function testInfo() {
RegExp.input=document.myform.Phone.value;
OK = re.exec(RegExp.input)
if (!OK)
 window.alert(RegExp.input + "isn't a phone
number with area code")
else
 window.alert("Thanks, your phone number is" +
OK[0]) }
</script>
```

Enter your phone number with area code and then click Test

```
<form name=myform><input name=Phone>
<input type=button name=PhoneButton value="Test"
onClick=testInfo(this);>
</form></ html>
```

# Using Regular Expressions for Password Validation

```
<script type="text/javascript">
function checkForm(form) {
if(form.username.value == "") {
 alert("Error: Username cannot be blank!");
 form.username.focus(); return false; }
re = /^[w+$/;
if(!re.test(form.username.value)) {
 alert("Error: Username must contain only letters, numbers and underscores!"); form.username.focus(); return false; }
if(form.pwd1.value != "" && form.pwd1.value == form.pwd2.value) {
 if(form.pwd1.value.length < 6) { alert("Error: Password must contain at least six characters!");
 form.pwd1.focus(); return false; }
 if(form.pwd1.value == form.username.value) { alert("Error: Password must be different from Username!");
 form.pwd1.focus(); return false; }
re = /[0-9]/;
if(!re.test(form.pwd1.value)) { alert("Error: password must contain at least one number (0-9)!"); form.pwd1.focus(); return false; }
re = /[a-z]/;
if(!re.test(form.pwd1.value)) { alert("Error: password must contain at least one lowercase letter (a-z)!");
 form.pwd1.focus(); return false; }
re = /[A-Z]/;
if(!re.test(form.pwd1.value)) { alert("Error: password must contain at least one uppercase letter (A-Z)!");
 form.pwd1.focus(); return false; }
} else { alert("Error: Please check that you've entered and confirmed your password!"); form.pwd1.focus(); return false; }
alert("You entered a valid password: " + form.pwd1.value); return true; } </script>
```



# Simplifying the Regular Expression

- The code on the previous slide is fine in that it checks everything that we wanted to check but uses a lot of code to test each requirement individually and present different error messages.
- Simpler way to accomplish it, with a single regular expression
- See: <http://www.the-art-of-web.com/javascript/validate-password/>
- Guide here: [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular\\_Expressions](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Regular_Expressions)
- Consider the following:

```
<script type="text/javascript">
// at least one number, one lowercase, one uppercase letter
// at least six characters
var re = /(?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{6,}/;
var validPassword = re.test(input);
</script>
```

the thing that is new is the **notation ?=**

?=.\*\d means at least one character that is a number

and

?=.\*[a-z] means at least one character that is a lower-case letter

and

?=.\*[A-Z] means at least one character that is an upper-case letter

and

.{6,} means at least 6 characters

# Implementation

```
<script type="text/javascript">
function checkPassword(str) {
 var re = /^(?=.*\d)(?=.*[a-z])(?=.*[A-Z])\w{6,}$/; return re.test(str); }
function checkForm(form) {
 if(form.username.value == "") { alert("Error: Username cannot be blank!");
 form.username.focus(); return false; }
 re = /^[\w+$/;
 if(!re.test(form.username.value)) {
 alert("Error: Username must contain only letters, numbers and underscores!");
 form.username.focus(); return false; }
 if(form.pwd1.value != "" && form.pwd1.value == form.pwd2.value) {
 if(!checkPassword(form.pwd1.value)) {
 alert("The password you have entered is not valid!");
 form.pwd1.focus(); return false; }
 } else { alert("Error: Please check that you've entered and confirmed your password!");
 form.pwd1.focus(); return false; }
 return true; }
</script>
<form method="POST" action="form-handler.php" onsubmit="return checkForm(this);">
 <p>Username: <input type="text" name="username"></p>
 <p>Password: <input type="password" name="pwd1"></p>
 <p>Confirm Password: <input type="password" name="pwd2"></p>
 <p><input type="submit"></p> </form>
```

# PassPhrases (Preview)

3. A passphrase is like a password but longer and more secure. It is an encryption key that you memorize.
- **Problem:** how do you come up with easy-to-memorize but very secure passphrases? Just thinking of one is incredibly hard, especially if your adversary really is capable of one trillion guesses per second.
- Using an entirely random sequence of characters it might be very secure, but it's also agonizing to memorize.
- **Solution:** use Diceware:  
<https://en.wikipedia.org/wiki/Diceware>
- Shakespeare quotes are not good passphrases because they lack something called entropy (randomness)
- Diceware is based on a word list which contains 7,776 English words — 37 pages of it printed:  
<http://world.std.com/~reinhold/dicewarewordlist.pdf>
- Roll a dice 5 times to select one word at a time for your passphrase
- If you are worried about the NSA, come up with a 7-word passphrase: there is a one in 1,719,070,799,748,422,591,028,658,176 chance that an attacker will pick your passphrase each try.
- At one trillion guesses per second — per Edward Snowden's January 2013 warning — it would take an average of 27 million years to guess this passphrase.
- This one does it: "bolt vat frisky fob land hazy rigid"
- **Recommendation:** use a password database (KeePassX) locked up with a master "diceware" passphrase:  
<https://www.keepassx.org>
- See: <https://theintercept.com/2015/03/26/passphrases-can-memorize-attackers-cant-guess/>

## PassPhrases (cont'd)

3. In August 2017, the original author of **NIST 2003 password guidelines**, Bill Burr, regrets making the error of recommending passwords with:

- ▶ Uppercase and lowercase.
- ▶ Letters and Numbers.
- ▶ Special characters.
- ▶ A minimum of 8 characters
- ▶ See:  
<https://www.wsj.com/articles/the-man-who-wrote-those-password-rules-has-a-new-tip-n3v-r-m1-d-1502124118>
- ▶ The NIST 2017 “Digital Identity Guidelines” now recommends “diceware” passphrases:  
<https://pages.nist.gov/800-63-3/>
- ▶ A comic poster is available here:  
<https://xkcd.com/936/>
- ▶ A passphrase generator is available here:  
<https://www.rempe.us/diceware/#eff>
- ▶ A password checker is available here:  
<http://csci571.com/zxcvbn/demo/index.html>

# JavaScript: `use strict`

- The idea of “strict” mode:
  - in normal JavaScript, mistyping a variable name creates a new global variable. In strict mode, this will throw an error, making it impossible to accidentally create a global variable.
  - In normal JavaScript, a developer will not receive any error feedback assigning values to non-writable properties.
  - In **strict mode**, any assignment to a non-writable property, a getter-only property, a non-existing property, a non-existing variable, or a non-existing object, will throw an error.
- Strict mode is declared by adding "`use strict`"; to the beginning of a JavaScript file, or a JavaScript function.
- Declared at the beginning of a JavaScript file, it has global scope (all code will execute in strict mode).
- Declared inside a function, it has local scope (only the code inside the function is in strict mode).

# Common JavaScript Mistakes

## 1. **Undefined** may not be **null**

- In JavaScript something that has not been assigned to is not null, but undefined is different from null when using `!=` but not when using the weaker `==` because JavaScript does some implicit casting in the later case
- For details see <http://weblogs.asp.net/bleroy/archive/2005/02/15/373815.aspx>

## 2. You cannot **overload** a function

- If you try to define two different functions with the same name but with different arguments, assuming the proper function will be called when the number of arguments match (this is overloading), then your assumption is incorrect. JavaScript will simply the latest-defined version of the function and call it;
- If a parameter is omitted, it is undefined

## 3. **Undeclared** variables are **global**

- If a variable is NOT declared using `var`, then it is global. Two variables of the same name, both undeclared will create conflicts that are hard to debug

# JavaScript Libraries

- See

<https://medium.com/javarevisited/10-of-the-most-popular-javascript-frameworks-libraries-for-web-development-in-2019-a2c8cea68094>

- Each library listed includes a link to library home page



## 10 of the Most Popular JavaScript Frameworks & Libraries for Web Development in 2022

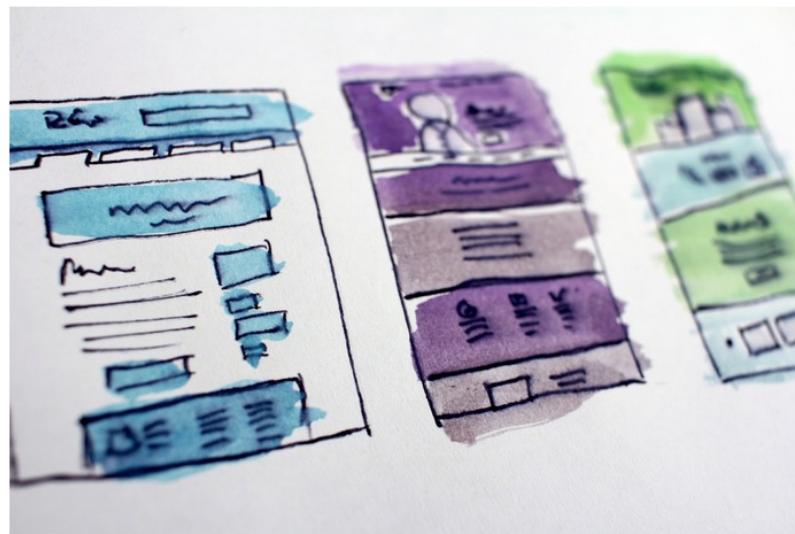
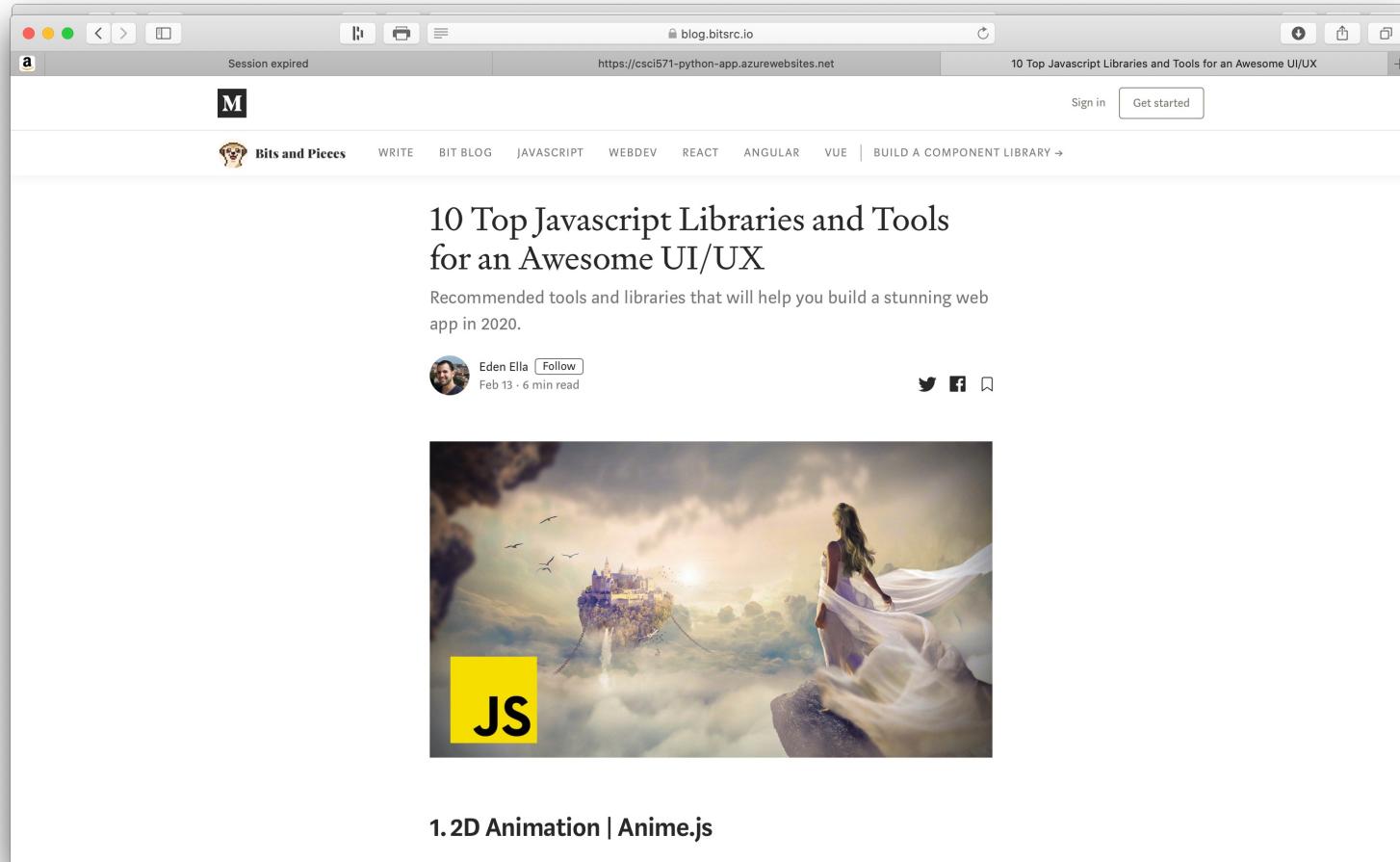


Photo by [Hal Gatewood](#) on [Unsplash](#)

# JavaScript UI/UX Libraries

- These library are good to build stunning looking UIs
- See <https://blog.bitsrc.io/10-top-javascript-libraries-and-tools-for-an-awesome-ui-ux-828a314752cc>



# How do you judge a JavaScript programmer by only 5 questions?

- **1. Can you explain the difference between “call” and “apply” to me?**

The answer to this question is a bit of a factoid, so that someone can answer it doesn't give you any information, but if they cannot, it gives a truckload. Almost all JavaScript programmer that has written a library or two (which most curious ones will, after programming it a few years) will know this.

Addendum: Several people are calling #1 into question. I must be very clear here that I stand firm on this one. If you haven't used apply, you are most likely missing out on the most powerful and overlooked aspects of the language. It's also an indicator that you haven't tried your hand at building a library yet, because when building libraries, apply and call are very commonly used.

- **2. Can you explain “map” to me?**

Map is a an extremely useful functional programming concept that any computer science person will know. If someone doesn't know this, it's a sign that they lack an understanding of computer science and/or lack an understanding of the language. In addition, the explanation itself will give you a sense of how much the person knows about the language just in the way they talk. If the person does well on this question, ask about reduce as a follow up. If you do not know what map is, it means that you have done zero functional programming and you're missing out. Severely.

- **3. Can you explain “bind” to me?**

This is a really great question, because it delves into the concept of this. You can basically drill the interviewee for quite some time on this, as it is a very large subject. You'll get a good sense of a programmer by having this discussion.

- **4. Can you explain how “closures” work to me?**

This is a great question to ask programmers that claim to be experienced in general, but not with JavaScript. Closures are a general programming concept that is extraordinarily important in JavaScript. If they understand closures well, they will learn JavaScript pretty quickly.

- **5. Can you please tell me a story about a JavaScript performance problem that you've encountered and how you approached solving it?**

This will tell you a lot about how much programming a person has actually done, in their own words. A big one to keep an eye out for is that they should be praising the Google Developer tools, and not rely too much on theoretical time complexity.