

CS110 JavaScript Quick Reference

This is not the place to *learn* JavaScript, but it can be useful as a reference.

Table of Contents

- [Using JavaScript](#)
- [Datatypes](#)
- [Variables](#)
- [Arrays](#)
- [Operators](#)
- [Comparison Operators](#)
- [Expressions](#)
- [Comments](#)
- [Statements](#)
- [Conditionals](#)
- [Built-in Functions](#)
- [Objects and Methods](#)
- [User-Defined Functions](#)
- [Document Object Model](#)
- [Events and Event Handlers](#)

Using JavaScript

Put [JavaScript on your page](#) using the SCRIPT tag:

```
<script type="text/JavaScript">  
</script>
```

or read in a file of JavaScript code:

```
<script type="text/JavaScript"></script>
```

or use it in an event handler:

```
<tag onClick="JS code here" onMouseOver="JS code here">
```

Datatypes

A *datatype* is a representation for a kind of data.

- **Number**, something we can add, subtract,... calculate with. Two kinds: integer and floating point. Integers are like 1, -7. Floating point are like 3.14, -2.18.
- **String**, something to represent and display text. String *literals* have either single or double quotation marks around them, such as "hi" or 'bye'
- **Boolean**, something that represents true or false. The literal values are **true** and **false**

Variables

A variable is a symbolic name for an unspecified value in an expression or statement. A variable can be used in any situation where the data that it refers to can be used. A variable name can refer to any of the [data types](#), described above. Variables are given values by using

the [assignment operator](#) as described below.

When you are *declaring* a new variable (creating a new storage location), you use the **var** keyword in front of the variable.

```
var myVar = initialValue;
```

This tells JavaScript that you are creating a new variable.

Scope

The use of the **var** keyword determines the *scope* of a variable. Roughly, the word “scope” means the places or contexts where a variable name has a single meaning. In English, consider the word “president”. In many contexts, “the president” would mean “the President of the United States”, but in other contexts, it might mean the president of Wellesley College, the president of a student organization, or something else entirely.

If you use the **var** keyword inside a function definition, the scope of the variable you create is only inside that function, and the variable's value only lasts until the function returns. The variable is re-created afresh each time the function is invoked. Thus, the following two **tmp** variables are invisible to each other and don't interfere with each other, because the two “tmp” variables refer different storage locations.

```
function hypotenuse(a,b) {  
    var tmp = a*a+b*b;  
    return Math.sqrt(tmp);  
}  
  
function bill_with_tip(subtotal,tip_percentage) {  
    var tmp = Math.round(subtotal*tip_percentage);  
    return subtotal + tmp;  
}
```

On the other hand, the following two functions *share* the global **count** variable: the “count” variables all refer to the same storage location:

```
var bearCount = 0 ; // how many bears have we seen?  
  
function bearSighting() {  
    bearCount = bearCount + 1;  
    alert("Threatdown, a bear!");  
}  
  
function neverMind() {  
    bearCount = bearCount - 1;  
    alert("nevermind, it was just a squirrel.");  
}
```

JavaScript Reserved Words

The following lists identify variable names that should be avoided because they are already used by JavaScript.

break	continue	delete	else
false	for	function	if
in	new	null	return
this	true	typeof	var
void	while	with	

Java Keywords Reserved by JavaScript

The following lists keywords used by the Java programming language and should *also* be avoided in JavaScript.

abstract	boolean	byte	case
char	class	const	default
do	double	extends	final
finally	float	goto	implements
import	instanceof	int	interface
long	native	package	private
protected	public	short	static
super	switch	synchronized	throw
throws	transient	try	

Other Variable Names to Avoid

The following are various built-in names known to JavaScript and therefore should be avoided

alert	Anchor	Area	Array
assign	blur	Boolean	Button
Checkbox	clearTimeout	close	closed
confirm	Date	defaultStatus	Document
document	Element	escape	eval
FileUpload	focus	Form	Frame
frames	Function	getClass	Hidden
History	Image	isNaN	java
JavaArray	JavaClass	JavaScript	JavaScript
length	Link	Location	location
Math	MimeType	name	navigate
Navigator	netscape	Number	Object
onblur	onerror	onfocus	onload
onunload	open	opener	Option
Packages	parent	parseFloat	parseInt
Password	Plugin	prompt	prototype
Radio	ref	Reset	scroll
Select	self	setTimeout	status
String	Submit	sun	taint
Text	Textarea	top	toString
unescape	untaint	valueOf	Window
window			

(Source: JavaScript the Definitive Guide. David Flanagan, O'Reilly & Associates, Inc. 1997.)

Array

An [array](#) is a datatype where there are a numbered set of elements, each of the same datatype. For example, a bunch of numbers could be stored in an array, or a bunch of strings, or a bunch of anything. The elements are distinguished by their *index*: the number of the element. Indexes start at zero.

Goal	General Form	Examples
Create an empty array	var var = Array()	var myThings = Array()
Create an initialized array	var var = Array(elt, elt ...)	var myKids = Array("Ross", "Charlotte");
Create a literal array	var var = [elt, elt ...]	var myKids = ["Ross", "Charlotte"];

Get an array element	<code>var[index]</code>	<code>myKids[0]</code>
Set an array element	<code>var[index] = value</code>	<code>myKids[2]="Maggie"</code>
Find out how long an array is	<code>var.length</code>	<code>if (myKids.length == 2)</code>

Operators

See [JavaScript Operators](#)

Operator	Name	Data Types	Examples	Result
=	Assignment	Number String Boolean	<code>x = 3;</code> <code>y = 7.654;</code> <code>Str = "foo";</code> <code>flag = true;</code>	<code>x == 3</code> <code>y == 7.654</code> <code>Str == "foo"</code> <code>flag == true</code>
+	Addition or Concatenation	Number String	<code>x = 4 + 3;</code> <code>y = x + 7.432;</code> <code>Str = "foo" + "bar";</code>	<code>x == 7</code> <code>y == 14.432</code> <code>Str == "foobar"</code>
-	Subtraction	Number	<code>x = 7 - 2;</code> <code>y = x - 3.359;</code>	<code>x == 5</code> <code>y == 1.641</code>
*	Multiplication	Number	<code>x = 2 * 5;</code> <code>y = x * 9.773;</code>	<code>x == 10</code> <code>y == 97.73</code>
/	Division	Number	<code>x = 10 / 4;</code> <code>y = x / 8;</code>	<code>x == 2.5</code> <code>y == .3125</code>
%	Remainder ¹	Number	<code>x = 10 % 4;</code> <code>y = x % .75;</code>	<code>x == 2</code> <code>y == .5</code>
&&	AND	Boolean	<code>flag = true;</code> <code>D = true && flag;</code> <code>D = D && false;</code>	<code>flag == true</code> <code>D == true</code> <code>D == false</code>
	OR	Boolean	<code>flag = true;</code> <code>D = flag true;</code> <code>D = false D;</code>	<code>flag == true</code> <code>D == true</code> <code>D == false</code>
!	NOT	Boolean	<code>D = !true;</code> <code>D = !D;</code>	<code>D == false</code> <code>D == true</code>

1. The Remainder operator is often referred to as the Modulo operator.

Comparison Operator:

See [JavaScript Comparison and Logical Operators](#)

Operator	Name	Data Types	Examples
==	Equality	Number String Boolean	<code>3 == 3</code> <code>"test" == "test"</code> <code>false == false</code>

!=	Inequality	Number String Boolean	12 != 39 "foo" != "bar" false != true
<	Less Than	Number String ¹	2 < 3.1415 "a" < "b"
<=	Less Than or Equal	Number String ¹	17 <= 17 "ABC" <= "ABD"
>	Greater Than	Number String ¹	12 > -4 "foo" > "bar"
>=	Greater Than or Equal	Number String ¹	3.99 >= 3.98 "fool" >= "foo"
1. Strings follow standard dictionary ordering, i.e., "A" < "B" < ... < "Z" < "a" < "b" ... < "z" "a" < "aa" < ... < "ab" < "aba" < ... < "b"			

Expressions

An *expression* is some syntax that yields a value. For example, 3+4 is an expression that yields the value 7. You can think of it as a long-winded notation for 7. Expressions can be put almost anyplace in a JavaScript program where a value can be put: the right hand side of an assignment statement, the argument of an `alert()`, `document.write()`, or other function or method, the condition of an `if` statement, and others.

One thing to keep in mind is *precedence* of operators: 2+3*4 means 2+(3*4), not (2+3)*4, because the multiplication operator, *, has precedence over the addition operator, +. In order of decreasing precedence, the operators from above are:

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

See [JavaScript Operator Precedence and Associativity Chart](#) for more than you ever want to know.

Comments

JavaScript has two kinds of [comments](#) — both end-of-line comments and block comments:

```
/* the following code is pretty confusing, so
   let me explain how it works. */

var w = 3; // I have no idea what this variable means
```

Statements

A [JavaScript statement](#) *does* something. That is, it's a syntactically meaningful part of a program that changes the state of the program or environment, such as reading something in, printing something out, changing the value of a variable, or the like.

In JavaScript, as in many programming languages, most statements end with a semi-colon. Thus, the following are equivalent:

```
x = 3 + parseInt(prompt("enter an integer"));
x = 3 +
  parseInt(
    prompt("enter an integer"));
```

Conditionals

JavaScript has two basic forms of the **if** or [conditional](#) statement.

```

if( condition ) {
  then-block
}

if( condition ) {
  then-block
} else {
  else-block
}

```

Note the syntax: parentheses around the condition, whether or not the condition contains parentheses, braces around the blocks of code (these are technically optional, but we prefer to teach you fewer syntaxes), and no semicolon (though you will see semi-colons inside the brace-enclosed blocks).

Here are two examples:

```

if( tipPercentage > 20 ) {
  alert("Thank you, you're very kind!");
}

if( tipPercentage > 20 ) {
  alert("Thank you, you're very kind!");
} else {
  alert("I hope everything was okay."); // you tightwad
}

```

The **if-else** construct can be chained together to yield what we call a “cascaded if”:

```

if( tipPercentage < 10 ) {
  alert("Sorry if your meal wasn't satisfactory"); // you cheap jerk
} else if( tipPercentage < 15 ) {
  alert("Please come again"); // and bring more money
} else {
  alert("Glad to be of service"); // really
}

```

Built-In Functions

Function	Examples	Result
<code>parseFloat</code>	<pre>x = parseFloat("12.8"); y = parseFloat("ABC");</pre>	<pre>x == 12.8 y == NaN</pre>
<code>parseInt</code>	<pre>x = parseInt("12.8"); y = parseInt("ABC");</pre>	<pre>x == 12 y == NaN</pre>
alert	<pre>alert("duck!"); alert("the variable X contains "+X);</pre> <p>The browser will pop up a window containing the given text. The user has to confirm before the browser will proceed, so this is intrusive and annoying, and therefore best used for debugging, as in the second example.</p>	
prompt	<pre>theName = prompt("Enter Your Name:", "Default Value");</pre> <p>The browser will display a dialog box, and prompt() will return the text that the user types in. In this example, that</p>	

	value will be assigned to theName .
write	<pre>document.write("Howdy"); document.write(2+3);</pre> <p>the document now contains Howdy and 5</p>

Objects and Methods

JavaScript is an [object-oriented programming](#) (OOP) language. *Objects* are a way of organizing the functionality of a large collection of software. Informally: an *object* represents some data, and allows you to access and manipulate that data using *methods*, which are functions that just work on that object.

Here are some of the objects, properties and methods we have learned:

- [String](#) objects. Some properties and methods, assuming **var str="hi there";**
 - **str.length** returns the length of the string, in this case, 8 (the space character counts, too).
 - **str.indexOf("e")** returns the location of the argument within the string, or -1 if it's not found. Here, it returns 4 (the location is zero-based).
- [Date](#) objects. Some properties and methods, assuming **var now=new Date();**
 - **now.getMinutes()** returns the minutes: 0-59
 - **now.getHours()** returns the hour of the day (24-hour clock): 0-23
 - **now.getDate()** returns the (one-based) day of the month: 1-31
 - **now.getDay()** returns the (zero-based) day of the week: 0-6
 - **now.getMonth()** returns the (zero-based) month number: 0-11
 - **now.getFullYear()** returns the year as a four-digit number.
 - **now.getTime()** returns the number of milliseconds since midnight, Jan 1, 1970.
- The **document** object. This represents the document in the browser, allowing JavaScript to modify and control it.
 - **document.write("text")** adds "text" to the document at this point in the source code. *Note* that this method can only be used while the initial document is loading into the browser; it can't be used once the document is loaded, so it can't be used from event handlers.
 - **document.getElementById("id")** returns a JavaScript object that represents a given element (e.g. one of the HTML tag in the document, such as a DIV, a P, an IMG, or any such part of a web page).
 - **document.getElementsByTagName("tag")** returns an array of JavaScript objects representing all the elements defined by a particular tag. For example, an array of every IMG in the document, or every paragraph...
- The [Math](#) object. This is just a way of organizing some built-in mathematical constants and functions:

Function	Examples	Result
square root	<code>x = Math.sqrt(9);</code>	<code>x == 3</code>
absolute value	<code>x = Math.abs(-23);</code> <code>y = Math.abs(99);</code>	<code>x == 23</code> <code>y == 99</code>
maximum	<code>x = Math.max(3, 14);</code>	<code>x == 14</code>
minimum	<code>x = Math.min(3, 14);</code>	<code>x == 3</code>
power	<code>x = Math.pow(2, 3);</code>	<code>x == 8</code>
floor	<code>x = Math.floor(3.14159);</code> <code>y = Math.floor(4.5623);</code>	<code>x == 3</code> <code>y == 4</code>
ceiling	<code>x = Math.ceil(3.14159);</code> <code>y = Math.ceil(4.5623);</code>	<code>x == 4</code> <code>y == 5</code>

round	x = Math.round(3.14159); y = Math.round(4.5623);	x == 3 y == 5
-------	---	------------------

User-Defined Functions

Like almost all computer languages, JavaScript allows programmers to [define functions](#). To recap this aspect of JavaScript, we have to talk about (1) definitions, (2) invocations, (3) arguments, and (4) values.

Definition

The **function** keyword introduces the definition of a new function. Don't re-define functions. The following example defines a function that takes no arguments and returns no values. It allows the programmer to put an alarming pop-up window on the screen.

```
function panicAttack() {  
    alert("Eeek! Help! I have no idea what's gone wrong!");  
}
```

Invocation

Function definitions are like dictionaries: they say what the words mean, but they don't *do* anything. To have any effect, you have to *use* the words or **invoke** the function. The following causes the alarming pop-up window to appear:

```
panicAttack();
```

Arguments

The **panicAttack** function was very inflexible, since it could only say the one message. A function can be made more flexible by supplying it some “inputs” that it can base its work on. These inputs, called *arguments*, are a series of values enclosed in the parentheses of the function invocation. The variable names in the parentheses of the function definition are set to the corresponding input values. The following functions could be used in more situations:

```
function reportTotal(total) {  
    alert("Your total is "+total+" Thank you for your purchase.");  
}  
  
function reportSale(subtotal,taxRate) {  
    var total = subtotal + subtotal * taxRate;  
    alert("Your total, including tax, is "+total+" Thank you for your purchase.");  
}
```

These functions would be invoked by giving their names and values for the arguments:

```
reportTotal(44); // spent $44  
reportSale(44,0.05); // MA tax rate used to be 5%
```

Values

Functions can work “for effect” (that is, for what they do) or for the value that they return. Some functions just compute something (possibly based on arguments) and return the result. The caller is expected to do something useful with the result. The keyword **return** indicates what the value of the function is, and also means to stop executing the function right now and return this value to the caller.

```
// prompt the user for an integer and return an integer  
function promptInt(description) {  
    return parseInt(prompt(description));  
}
```

```

// returns the arithmetic mean of the two arguments
function average(x,y) {
    return (x+y)/2;
}

// returns true if and only if the person is of legal drinking age
function legal(age) {
    return age >= 21;
}

```

They could be used as follows:

```

var age1=promptInt("how old are you?");
age2=promptInt("how old is your significant other?");
if( legal(average(age1,age2)) ) {
    alert("go have some Champagne to celebrate");
} else {
    alert("go have some seltzer water to celebrate");
}

```

Example

The following functions are a cumbersome way of representing all the integers, using the so-called [Peano axioms](#):

```

function zero() {
    return 0;
}

function next(n) {
    return n+1;
}

```

Which you could use like:

```

alert("three is " + next(next(next(zero()))));

```

Document Object Model (DOM)

One important use of JavaScript is to manipulate the structure and style of a document via the [Document Object Model](#) or DOM.

Our general strategy is to manipulate one element of the page at a time by looking up the element by its ID and then setting various properties to modify the object. Here are some examples:

- Changing the picture displayed by an IMG elt (used in rollovers and slideshows):

```

var elt=document.getElementById(id of IMG); elt.src=new URL;

```

- Changing the style of an element. Each CSS property has a corresponding DOM property, usually of the same name.

```

var elt=document.getElementById(id);
elt.style.border="2px solid red"; // CSS is border
elt.style.fontSize="16pt"; // CSS is font-size
elt.style.backgroundColor="gray"; // CSS is background-color
elt.style.display="hidden"; // CSS is display

```

- Changing the contents of an element using **innerHTML**

```
var elt=document.getElementById(id);
    elt.innerHTML = "<p>here is some text, which is "+
        "<ul>"+
            "<li>new"+
            "<li>long"+
            "<li>complex"+
            "<li>done"+
        "</ul>";
```

Events and Event Handlers

JavaScript code can be triggered by something that the user does, or, in general, JavaScript code can be executed as a consequence of some [event](#). The code that is triggered is called an event *handler*.

Some common event handlers are

- **onClick**: triggered when the user clicks on the element
- **onMouseOver** triggered when the mouse rolls over the element
- **onMouseOut** triggered when the mouse leaves the element

An example:

```
<p style="font-size: 10pt; border: 1px solid gray"
  onMouseOver="var n=Math.ceil((new Date()).getSeconds()/2);
  this.style.border=n+'px solid red';"
  onMouseOut="this.style.border='1px solid gray';"
  onClick="this.style.fontSize=(parseInt(this.style.fontSize)+1)+'pt';">Here
is the example in action. Mouse over this paragraph to change the
border,
and click on it to change the font size. Reload the page to re-set.
</p>
```

Here is the example in action. Mouse over this paragraph to change the border, and click on it to change the font size. Reload the page to re-set.

© Computer Science 110 Staff

This work is licensed under a [Creative Commons License](#)

Date Modified: Monday, 10-May-2010 01:45:10 EDT