The CULM Guide to JavaScript Programming

(Self-education series: CompSci-301)

Book 1: "Easy Object oriented engineering course"

Pre-requisites:

"HTML1, 2, 3"

Object Orientation 1

Machine Grammar 1

Level:

Beginner/Power User

Jr.Developer

Tier:

Public/All (GPL)

The CULM Guide to JavaScript Programming: Part 1

By Troll X

JavaScript is a versatile language. It can be used to create menus, validate forms, provide interactive calendars, post the current day's headlines, produce background effects on a Web page, track a visitor's history on your site, and play games, among many other things. That's probably why it's one of the most popular languages on the World Wide Web.

Netscape created JavaScript in 1995. Originally called "LiveScript," it was designed to make Web pages more interactive. In the beginning the language was plagued with security problems which, for the most part, have been overcome. The current version of JavaScript is 1.5.

Because of its usefulness, I've long thought about whether I should learn JavaScript programming or not. One of the main reasons I kept putting it off was the availability of free, existing scripts. 'Why re-invent the wheel?'.

Today, there is a lot more that JavaScript can do and as a result, I've decided the time has come to teach it. My intention is not to become a JavaScript guru or programming genius but to be able to write and teach scripts that people can use in work, and perhaps share with others.

I know that I'm not the only one who feels this way and yet, learning JavaScript seems to be a monumental task. When you look at some of the JavaScripts out there, they can seem pretty intimidating. And that's true with most things that are new. Remember the first time you viewed the source code of an HTML document or took a look at a style sheet? Whoa! I'm sure you felt overwhelmed..

Still, there is only one way to get started; you just do it! To make the process easier, I decided we could do this together. Our goal in this study will be to learn how to competently write useful JavaScripts.

For instructional purposes, I will be using the following two references:

- *Sams Teach Yourself JavaScript in 21 Days*, Andrew Watt and Jonathan Watt, Sams Publishing, Indianapolis, IN, 2002
- *JavaScript: A Beginner's Guide - Second Edition*, John Pollock, McGraw-Hill/Osborne, Emeryville, CA, 2004

In addition, I will be using a few other books and Web sites that will help in our understanding of this intriguing language.

Before We Begin

There are a few things you will need before we begin our study. One of them is a basic knowledge of HTML/XHTML. While you don't need to be an expert, you will need to know the basics of using HTML elements and attributes. JavaScript can be used to dynamically create Web pages and a general knowledge of HTML is essential. A little knowledge of CSS won't hurt either.

Another thing you will need is a text (ASCII) editor. JavaScript code is just plain text so any good text editor will work well.. There aren't too many JavaScript editors out there right now (although I am reviewing one as we go through this process). There are, however, several good text editors. One editor I am very familiar with is NoteTab (which comes in three versions). I use it for all my HTML and text editing. You can also use WordPad or Notepad. **It would be best not to use** a word processor or a WYSIWYG HTML editor as they tend to add a lot of extra code that will cause the scripts to fail.

It's assumed that you have a Web browser (a.k.a. "client", "user agent"). You should test your scripts in both Internet Explorer and Netscape. It would also be wise to test them in Firefox and Opera, as these browsers are becoming more popular. There are other browsers also but these are the main ones being used today. Safari is another popular browser but it is used on the Apple system, which makes it hard to test using a Windows-based system.

Now let's get started!

The CULM Guide to JavaScript Programming: Part 1

An Introduction

First, there are a few things you need to understand.

JavaScript is *not* Java
Java is a ***compiled*** language. This means that the Java code must be transformed ("compiled") into a high-level programming language before it can run.

JavaScript is an ***interpreted*** language. It doesn't need to be compiled before it is run. In an interpreted language the instructions are ***parsed*** (divided into small components that can be analyzed). For instance, as the browser "reads" this page, it breaks down each of the page's components into individual components and interprets each component as it moves down the page. In linguistics it means to divide language into small components that can be analyzed. For example, parsing this sentence would involve dividing it into words and phrases and identifying the type of each component (such as a verb, adjective, or noun).

JavaScript can either be *client-side* or *server-side*
This refers to the place where the script is processed. We will be dealing with *client-side* scripts. This means that the script is processed ("runs") in the *client* (e.g., browser). The script will be processed directly in the browser without having to send or receive data to or from the server. Most JavaScript is *client-side*.

JavaScript is an object-based scripting language
Object-based means that JavaScript sees the existing data structure as objects. A JavaScript *object* is merely a thing. It's like the objects around us. In addition, each object has *properties*. For example, I have two cats. They are the *objects*. They have fur, ears, and tails. These are the *properties (more on this later*.)

JavaScript is cross-platform
The code will run on a wide variety of computer platforms. This means that users with all kinds of computers using different kinds of operating systems are able to obtain the same results as everybody else.

JavaScript comes in different 'flavors'

- *JavaScript*: This was the name given by Netscape, who invented it. It was first used in Netscape 2.0. This is the most popular version of the language and is what we will be learning. The current version is 1.5.
- *JScript*: This is the scripting language invented by Microsoft to compete with JavaScript.
- *ECMAScript*: The de facto international standard for JavaScript. This standard covers the core of the language.

Since version 4 of the Netscape and Internet Explorer browsers, JavaScript and JScript have had the same core functions. This means that they both utilize the core language features included in the EMCA standard. While IE recognizes both JScript and JavaScript, Netscape does not. However, since the tags are essentially the same, if the scripts are written in JavaScript, they will run on both browsers, for the most part. There are some differences between the JavaScript and JScript codes, but later on, we'll look at ways to overcome that within the context of the script itself.

A Few Things You Need to Remember

When writing scripts it's best to get in the habit of entering a semicolon at the end of each statement (e.g. `var x = range;`). While it's not required, it can help to eliminate errors.

There is no "proper" format for writing scripts. However, breaking up lines and indenting some of them does make it easier to read and debug. For instance, the layout here:

```
function getObject(obj) {
  var theObj;
  if(document.all) {
    if(typeof obj=="string") {
      return document.all(obj);
    } else {
      return obj.style;
    }
  }
  if(document.getElementById) {
    if(typeof obj=="string") {
      return document.getElementById(obj);
    } else {
      return obj.style;
    }
  }
  return null;
}
```

is much easier to read than this one:

```
function getObject(obj) {var theObj;
if(document.all) {if(typeof obj=="string") {return document.all(obj);}
else {return obj.style;}}
if(document.getElementById) {if(typeof obj=="string") {return
document.getElementById(obj);}
else {return obj.style;}}return null;}
```

You also may need to edit a script several months after writing it. If it's cleanly formatted it will be an easier job. In addition, if someone else uses your script it will make it easier for them to understand how the script works. For the most part, JavaScript ignores tabs and spaces. A little later on in our study we will look at when it is important to pay attention to these "whitespaces."

It's also a good idea to add comments to your scripts, especially if the script is long. There are two methods for adding comments. For a single comment line use two forward slashes, "`//`". You can even do this on the same line as the code:

```
// This whole line will be ignored by the JS interpreter

var myOldCar = Ford     // The first part will be interpreted; this
part will be ignored.
```

If you need to comment out several lines, you would begin with a forward slash and an asterisk, "`/*`", and end with an asterisk and a forward slash, "`*/`". For example,

```
/* All of this, all three lines,
will be ignored in their entirety
by the JavaScript interpreter */
```

Just a couple of cautions when placing comments in a script. Make sure you don't nest comments (one inside of another) and keep the comments concise to save download time. Everything you write in a script is downloaded to the browser. Although the comments are not viewed by the user, they still take up precious bandwidth and RAM.

Also, remember that JavaScript is *case-sensitive*. This means that it sees `myAuto` and `myauto` as two completely different variables. Be very careful when entering code. Using the wrong case will cause your script to fail.

The CULM Guide to JavaScript Programming: Part 1

Where Does the Script Go?

There are three places where a script can be located:

1. In the top section of the page ("head"), between the `<head>` and `</head>` tags;
2. In the middle section of the page ("body"), between the `<body>` and `</body>` tags; or
3. In a separate file.

Placing the Script on the Page

When the script is placed on the page, it is located between a set of container tags that look like:

```
<script type="text/javascript">
<!--
... The script is placed here ...
//-->
</script>
```

The opening tag, `<script type="text/javascript">`, tells the browser:

- it has encountered a script;
- it is in text format; and
- it is to be interpreted as JavaScript.

Note: There are several different ways to write this, but the W3C recommends the method shown here so that's what we'll use).

The next line, `<!--`, is the opening comment tag. It's used to hide the code from older browsers. The actual script starts on the third line. After the script ends, the comment tag is closed, `//-->` and the `</script>` tag tells the browser that the code is finished. The `<script type="text/javascript">` and `</script>` tags are known as "container" tags because they "contain" the script within them.

Placing the Script in an External Page

When the script is located in an external file, the opening and closing tags shown above are not included. Only the script itself is included in the file. The file should have an extension of ".js." The HTML page would then contain a link to the JavaScript file, which would look like this:

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

The opening tag, `<script type="text/javascript" src="/javascript/scriptfile.js">`, tells the browser:

- it has encountered a script;
- it is in text format;
- it is to be interpreted as JavaScript; and
- the script is located in an external file located at the given URL.

The closing tag, `</script>`, tells the browser that the code is finished. Nothing should be written between the opening and closing tag.

The link is usually located in the head of the document. In some cases it may be necessary to locate it within the body. It will depend on the script.

If the script is small and used only once, it's usually best to put it in the actual document. Otherwise, it's preferable to place the script in an external file.

By placing the script in an external file, it can be used on other pages without having to duplicate the code. The browser will then cache the script and use it on the other pages, making them to load faster. Using an external file also makes it easier if changes need to be made to the code. You only have the one file to change instead of making changes to each individual page.

Reserved Words

There are 59 "reserved" words. These are words that are used to give instructions to the JavaScript interpreter. They cannot be used for anything else and are listed below. We'll be looking at them in more detail as we use them. For now, just remember not to use them as variables.

abstract	boolean	break	byte
case	catch	char	class
const	continue	debugger	default
delete	do	double	else
enum	export	extends	false
final	finally	float	for
function	goto	if	implements
import	in	instanceof	int
interface	long	native	new
null	package	private	protected
public	return	short	static
super	switch	synchronized	this
throw	throws	transient	true
try	typeof	var	void
volatile	while	with	

Wrap-Up

That should just about do it for this part. So far we've:

- gained a little insight into the history of JavaScript;
- found out what we needed to write the scripts;
- gained some insight into what JavaScript is and what it isn't;
- learned how to format a script;
- learned how to make comments in a script;
- and found out where to place a script in relation to the Web page.

Next time we'll be looking at variables and writing our first script.

Review Questions

Multiple Choice

1. JavaScript is:
    a. an interpreted language
    b. a compiled language
2. JavaScript is:
    a. subjective
    b. object based
    c. objective
3. To comment out a line:
    a. Precede it with an asterisk, i.e. "*"
    b. Precede it with an asterisk and a forward slash, i.e. "*/"
    c. Precede it with two forward slashes, i.e. "//"

True or False

1. JavaScript can only run on Windows.
2. Semicolons are optional at the end of a statement.
3. JavaScripts should not be formatted.
4. It is best to place the JavaScript on each separate page.

Essay

1. What is JavaScript, JScript, and EMCAScript?
2. What is the purpose of the `<script type="text/javascript">` tag? Explain how it works.
3. What is the purpose of the `<script type="text/javascript" src="/javascript/scriptfile.js">` tag? Explain how it works.
4. What extension should be used for an external JavaScript file?
5. What are "reserved words" used for?

The CULM Guide to JavaScript Programming: Part 2

By Troll X

In the first part, we looked at some general information and guidelines to help prepare us for our study of JavaScript. In this section, we'll begin to delve into parts of the language and we'll also write our first script.

Data Types

Scripts manipulate data in order to perform tasks that (hopefully) produce a desired effect. In a nutshell, that's really all there is to JavaScript.

A "**data type**" is just that: a type of data (e.g., letters, numbers) used within a script. JavaScript is a "**loosely typed**" language, meaning you don't need to specify the type of data being declared in a variable. The type of data can be changed later in the script without causing an error message (*you'll understand this better as we go along*). The JavaScript interpreter will determine the data type when it's processed within the script. If you've never done any other programming, don't worry too much about specifying data types. Suffice it to know that, for the most part, the JavaScript interpreter will take care of interpreting the type of data being used.

There are four basic data types used in JavaScript: *strings*, *numbers*, *booleans*, and *nulls*.

Strings

A string is merely a word or combination of words, including numbers, grouped (strung) together. Any type of character can be stored in a string. While there is no actual limit on the number of characters that a string can hold, keep in mind that many older browsers have a limit of 255 characters. In general, this shouldn't be a problem unless your visitors will be using older browsers. The following are examples of a string:

```
  "cheeseburger"
  "123 Main Street, Anytown, MA 01970"
  "His e-mail address is: bdylan@desolationrow.com"
```

A string is enclosed in quotes (" ..... ") to set it apart from the actual code. When the JavaScript interpreter encounters the first quotation mark, it will treat everything after it as a string until it reaches the closing quotation mark. Then it will again begin to interpret everything as code.

Either single or double quotes can be used but you must be consistent. If you start the string with a double quote, it must end with a double quote.

```
  var favfood="a big juicy hamburger";  // Correct
  var favfood='a big juicy hamburger';  // Correct

  var favfood="a big juicy hamburger';  // Incorrect
```

Sometimes you may need to use several sets of quotes in a string. In order to use a double quote within another double quote (or a single within a single) you would need to precede it with a backslash, "\". This is called an "escape sequence". It would then look like this:

```
  document.write("<a href=\"http://www.domain.com\">Check this
out!</a>");
```

The backslash tells the JavaScript interpreter that the next character should be displayed as it is. You could also combine single and double quotes. (It doesn't matter which set of quotes is used on the outside or the inside. You just need to be consistent.)

```
  document.write("<a href='http://www.domain.com'>Check this
out!</a>");

  document.write('<a href="http://www.domain.com">Check this
out!</a>');
```

In addition, there are several special characters that can be used in the escape sequence:

Sequence	Name	Sequence	Name
\b	Backspace	\f	Form feed
\n	New line	\r	Carriage return
\t	Tab	\'	Single quote
\"	Double quote	\\	Backslash
\xNN	Latin-1 character set.	\uNNNN	Unicode character set

These escape sequences are used when you are want to display information on a screen but need a little formatting.

HTML elements can also be used inside of strings. Doing so will help you to create more dynamic pages.

```
document.write("This is <strong>really</strong> fantastic!");
```

Be sure to remember that JavaScript strings are case sensitive. `"Fender and Gibson"` and `"fender and gibson"` are two different strings as far as the JavaScript interpreter sees them. This is especially important when you are comparing strings.

Numbers

JavaScripts recognizes two types of numbers: *integers* and *floating point numbers*. Integers are whole numbers (e.g., 1, 20, 546) and floating point numbers are numbers that have a decimal point (e.g., 23.8, 23.890). Unless otherwise specified, JavaScript treats all numbers as floating point numbers, but if a calculation is performed using integers, the answer will be an integer unless the calculation itself changes it (e.g., the number is divided unevenly).

Boolean

A Boolean data type is used for true or false statements. It's mostly used in `if()` statements:

```
var myCar=Chevy;
if (myCar==Chevy)
  {
    ... task to be performed if statement is true
  }
else
  {
    ... task to be performed if statement is false
  }
```

We'll look at these types of statements in more detail later but for now, the statement above is executed in the following sequence:

- the variable `myCar` is declared and initialized;
- a conditional statement asks if the variable `myCar` is equal ("==") to "Chevy";
- if it's true, the code on the next line would be executed;
- if it's false, (not equal to "Chevy"), then it will move on to the following line (after "`else`") and execute that code.

This is an excellent method for checking the validity of a statement, such as checking for a valid e-mail address. We'll look at this in more depth when we get into conditional statements.

Null

A null variable has no value. The variable does not return a space or a zero. It just means nothing. A null variable is useful for testing input in scripts. Since it's a keyword, it doesn't need to be enclosed in quotation marks:

```
var nothingNow=null;
```

The CULM Guide to JavaScript Programming: Part 2

Variables

Variables are a basic technique used for storing data in a script. A variable "holds" a specific value. In HTML coding, elements such as `<em></em>` and `<h2></h2>` are also known as "containers." The code affects the data which is "contained" within the elements. So it is with a variable. Anything done to a variable affects the value that is contained within it. Variables make data manipulation easier as the value can be acted upon without having to re-enter the original data each time.

When a variable is first stated in the code, it is said to be "declared," using the `var` reserved word. When data is given to the variable to be stored, it is "initialized." If the data is later changed, the new value is "assigned."

A variable can be declared and initialized using one of two methods. With the first method, you would declare the variable on one line and then initialize it on another line:

```
var myName;
    myName="Troll";
```

Basically these two lines say that I am declaring a variable named `myName` and the variable named `myName` contains the value "Troll."

Using the second method, you would declare the variable and initialize it all on the same line:

```
var myName="Troll";
```

This line says I am declaring a variable named `myName` and its value is "Troll." It's the same as above only shorter. The benefit becomes more apparent when you are using several variables.

Naming Variables

Variable are case sensitive. `myValue` and `MyValue` are two different variables according to JavaScript. The same case must be used throughout the entire script.

There is no real "correct" way to name variables, but there is an "unofficial" method. When naming a variable that is only one word, use lowercase: `var auto`. If a variable is two or more words, capitalize the first letter of each word, `var MyAuto` or capitalize all the words except the first: `var myFavoriteCar`. The method of capitalization is a personal choice but must be consistent!

A variable can only begin with either a letter or underscore character (_). The rest of the variable name may contain letters and numbers, the underscore character (_), and a dollar sign ($). Do not use a reserved word for a variable name.

Be sure and give meaningful names to variables. Later, when you begin writing complex scripts and you have several variables, a meaningful name helps you to remember what the variable is used for. For instance, if you are adding together the cost of accessories for a car, the statement:

```
totalPrice=750+250+300
```

doesn't mean too much just looking at it, but if you use meaningful names for your variables, the statement would then make more sense:

```
totalPrice=airCond+cdPlayer+magWheels
```

Since JavaScript, for the most part, ignores whitespace, you can write the variables either way:

```
var TheFirst="This string is stored in a variable.";
var TheFirst = "This string is stored in a variable.";
```

The CULM Guide to JavaScript Programming: Part 2

Our First Script

Let's take a break and write our first script. First, open a new document in your text editor and insert the following HTML elements:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Demo</title>
</head>

<body>

<h3>Just a Demo</h3>

</body>
</html>
```

You might want to save this as a template since we'll be using it to test our scripts.

In this script we'll use the `alert()` function. I'm sure you have seen the little popup windows that give you a warning about something you are getting ready to do. One type of these windows is called an *alert box*. First, here are a few quick rules. The message to be shown on the page must be enclosed in parentheses and, because it is a string, the message must be enclosed within quotation marks. If the message is a number it does not require quotation marks since it's not a string.

The code for our first script is below. While you could just do a cut-and-paste, it would be better if you actually got into the habit of writing the code yourself. That way, if any errors are made, you can see what you did wrong. It should be placed in the head of the document. (For now we'll place the script in the document instead of creating a separate file.)


```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML
4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">

<html>
<head>
<title>Demo</title>

<script type="text/javascript">
<!--
  var firstScript="My very first JavaScript!";
  alert(firstScript);
//-->
</script>

</head>

<body>

<h3>Just a Demo</h3>

</body>
</html>
```

Save this document and open it in your browser. The first thing that will happen is that the popup window (the alert box) will be displayed. After you click on the "OK" button, the page itself will display. The reason the box pops up before the page displays is because the browser reads from the top of the page down and encounters the script located in the head section before it gets to the actual body of the document. If the script were placed after the *<h3>Just a Demo</h3>* line, it would be displayed last. Go ahead and try it:

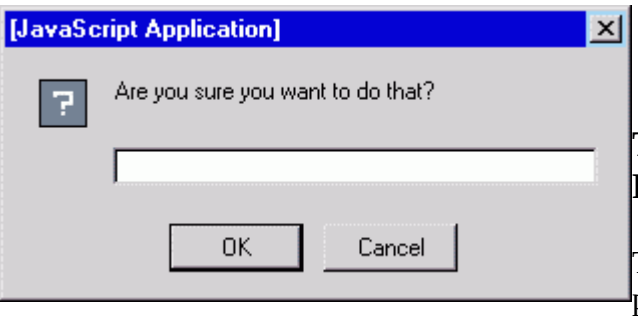
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Demo</title>

</head>

<body>

<h3>Just a Demo</h3>

<script type="text/javascript">
<!--
  var firstScript="My very first JavaScript!";
  alert(firstScript);
//-->
</script>

</body>
</html>
```

Now, let's take a look at the script and see what is actually happening. The first line of the script, `<script type="text/javascript">`, is the opening tag. As we've seen before, it tells the browser it has encountered a script, it's in text format, and it's to be interpreted as JavaScript. The next line, `<!--`, is the opening comment tag used to hide the script from older browsers.

The actual script begins on the third line. The first part is the `var` keyword. This tells the browser that the following text is the name of a newly declared variable. The variable's name is then declared, `firstScript`. The JavaScript assignment operator (=) is then given, followed by the declared value for the new variable, `"My very first JavaScript!"`. Since it's a string it's enclosed in quotes. (In JavaScript coding the "assignment operator" [=] does not mean "equal to" as in math. Two equal signs [==] are used to mean "is equal to". The assignment operator "assigns" a value to the variable.)

The next line calls the `alert()` function and references the variable `firstScript`. This tells the JavaScript interpreter to open an alert box window using the contents of the string in the variable `firstScript`. The HTML comment element is then closed and the script itself is ended on the last line.

Not too complicated, huh? Let's try another one.

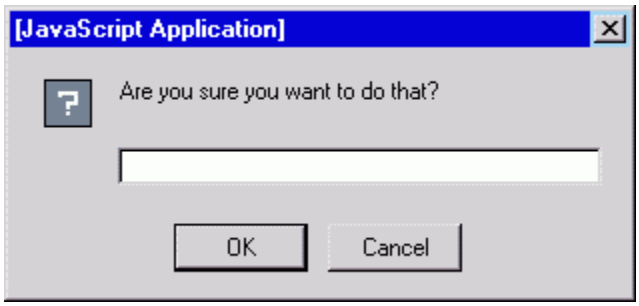


The CULM Guide to JavaScript Programming: Part 2

This time, let's get a little more sophisticated and use a prompt box. Don't let the script scare you. We'll look at each step in detail. Remember to start with your template and add the script to it.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
 "http://www.w3.org/TR/html4/strict.dtd">
<html>
<head>
<title>Demo</title>

</head>

<body>

<h3>Just a Demo</h3>

<script type="text/javascript">
<!--
  var ans=prompt("Are you sure you want to do that?","")
  if (ans) {
    alert("You said "+ans)
  }
  else {
    alert("You refused to answer")
  }
//-->
</script>

</body>
</html>
```

When you open the page in your browser and the box pops up, give an answer to the question, such as "I sure would," and check the result. Also, try just pressing "OK" or "Cancel" without entering an answer and see what it does.



Let's look at the script and see what exactly is happening. (We don't need to go over the open and closing tags and comment lines since you already know how they work. From now on I'll only be highlighting the actual script itself.)

On the first line (after the opening tags) the reserved word `var` is used to declare the variable "ans." The variable is also initialized using the `prompt()` method and two parameters ("Are you sure you want to do that?" and ""). The second parameter, "", is the default answer, which is null or nothing.

We then encounter an if/else statement, known as a "conditional statement." If you entered something in the prompt box (even a space), an alert box will be displayed with "You said" followed by your answer.

If you pressed "OK" without answering the question or if you pressed "Cancel," then it will skip to the next part of the statement and execute that line. That will open a prompt box with the words "You refused to answer."

Here is how this conditional statement works. The statement actually reads, "If the variable `ans` contains any data (what you would have entered in the prompt box) then open an alert box and print 'You said' and the data contained in the variable. If the variable `ans` does not contain any data, then open an alert box and print 'You refused to answer.'

Using Variables and Text Strings Together

There will be times when you might want to print out the results of a variable in a sentence. To do so, you would use the `document.write` command along with a text string and the variable. Here's how it works.

Let's say I want to write about music. I can list an artist in the variable and then use it to display information on the web page. Open your template and follow along with me. (You can place the code within the body of the document.) First I will declare the variable:

```
<script type="text/javascript">
<!--
  var dylan="Bob Dylan";
//-->
</script>
```

Next, I add the `document.write` command:

```
<script type="text/javascript">
<!--
  var dylan="Bob Dylan";
  document.write("I like music by "+dylan);
//-->
</script>
```

The result would then be: `I like music by Bob Dylan`. The script is very simple. First, the variable "`dylan`" is declared and initialized. Then it's added to the text string in the `document.write` command. The `document.write` command tells the script to write on the Web page whatever is in the parentheses.

In the `document.write` statement above, the variable is added to the text string by means of *concatenation* (kon-kat-uh-NAY-shuhn). This is, very simply, the means of uniting (linking together) two or more items. This permits us to join (concatenate) several items together to form sentences, commands, even entire documents. The items are not added together as in mathematics; rather they are *joined* to form an entirely new item. The "items" can be different data types or variables.

When you're writing the text be sure to add the proper spacing within the text string — in this case it would be after the word "by." This will leave a space before the word in the variable. Otherwise the last word in the text string and the variable will be printed together. i.e., `I like music` **`byBob`** `Dylan`. This is because JavaScript does not generally recognize whitespace. If you added the space after the string and before the variable, it wouldn't make any difference to the JavaScript interpreter; it would be ignored.

There are other ways that we can write this. We could add the variable in the middle of a sentence:

```
<script type="text/javascript">
<!--
  var dylan="Bob Dylan";
  document.write("Many people do not understand "+dylan+"'s music.");
//-->
</script>
```

This would print out `Many people do not understand Bob Dylan's music`. Notice here that I did not add a space in the text string after the variable since I wanted to add a letter to the variable.

We can also add HTML elements to the strings. For instance:

```
<script type="text/javascript">
<!--
  var dylan="Bob Dylan";
  document.write("Are you <strong>sure</strong> you don't like
"+dylan+"'s music?");
//-->
</script>
```

You could put each part of the HTML on the page in a variable and create the entire page (the part displayed in the browser) from within the script. We'll look at this in more depth later.

The CULM Guide to JavaScript Programming: Part 2

That will wrap up our study for this part. Practice writing variables and use the examples I have given you here. Remember to use your template and try them to make sure they work.

Review Questions

Multiple Choice

1. A data type is:
    a. a letter
    b. a number
    c. both
2. The four basic data types are:
    a. strings, numbers, BooBoos, and nulls
    b. strings, text, Booleans, and nulls
    c. strings, numbers, Booleans, and nulls
    d. strings, numbers, Booleans, and zeros
3. To display a backslash, using an escape sequence, it would be written:
    a. /\
    b. //
    c. |backslash
    d. \\
4. A variable is used to store:
    a. an escape sequence
    b. hidden script
    c. data

True or False

1. Comment tags are used to hide script from new browsers.
2. Variables cannot be used together with text strings.
3. A string cannot have more than one set of quotes.
4. HTML elements can be used inside of a string.
5. A null data type returns a zero.
6. Bob Dylan writes good music (*I just wanted to make sure you were awake*).
7. A variable must be declared and initialized all on the same line.
8. A variable name can begin with a number.

The CULM Guide to JavaScript Programming: Part 3

By Troll X

JavaScript Operators

In this part we'll take a look at JavaScript operators, which are used to accomplish many different tasks. This may be a review for some of you; though others you may want to study this subject in more depth.

An operator is a tool used to manipulate data. It can perform mathematical calculations, data comparisons, and assign data values to variables. Generally, operators are represented by mathematical symbols but in a few cases they are actual words. The term "operator" is derived from the action that is performed on a piece of data. The action is called an "operation:" the tool used to perform the operation is called an "operator." The values (i.e., numbers assigned to a variable) that are used in the operation are called "operands" (I'll use these two terms — *value* and *operand* — interchangeably).

The most common operators used in JavaScript can be divided into six different categories:

- ***Mathematical***: Used to perform mathematical calculations.
- ***Comparison***: Used to compare two values, variables or statements.
- ***Assignment***: Used to assign values to variables.
- ***Logical***: Used to compare two conditional statements to see if one or both are true.
- ***Bitwise***: Used to perform logical operations on the binary ("bit") level.
- ***Special***: Used for specific tasks, these operators can also help eliminate unnecessary code.

As we discuss these operators below, go ahead and try out the code using the template that we made last time. This way you'll see exactly what the code is doing. Remember also to type them in yourself as that will help you when you write your own. As we go through each example, try to figure out what is happening before you read the explanation.

Mathematical Operators

Operator	Name	Description
+	Addition	Adds two values together
-	Subtraction	Subtracts one value from the other
*	Multiplication	Multiplies two values
/	Division	Divides one value by another
%	Modulus	Divides one value by another and returns the remainder
++	Increment	Adds 1 to a value
--	Decrement	Subtracts 1 from a value
-	Negation	Changes the sign of the value [*Makes a positive value negative and a negative value positive.*]

Addition Operator (+)

This operator is used to add together numbers or strings.

```
var theSum=2+5;
alert(theSum);
```

In this script we are declaring a variable called `theSum` and initializing it with a value of 2+5. Next we call an alert window to display the results of the variable. This would result in an alert window displaying the number 7. Go ahead and try it.

```
var myNum=2;
var yourNum=5;
var theSum=myNum+yourNum;
alert(theSum);
```

This script declares and initializes three variables, `myNum`, `yourNum`, and `theSum`. The first two are given the values of 2 and 5, respectively. The third variable is a calculation, adding the first and second variables. Next, the script calls an alert window to display the results of the third variable, `theSum`. This would result in an alert window with the number 7. This script is basically the same as the previous one except that we can change the value of each variable individually.

```
var theSum=5;
var newNum=theSum+3
alert(newNum);
```

In this last example, we declare and initialize two variables, `theSum` and `newNum`. The first variable is given a value of 5. In the second variable, a value of 3 is added to the value of the first variable. An alert window is then called to display the results of the second variable, `newNum`, which produces the value of 8.

Subtraction Operator (-)

This operator is used to subtract the value on the right of the operator from the value on the left, i.e., "7 - 2" would read "seven minus two".

```
var newNum=7;
var newNum2=2;
var result=newNum-newNum2;
alert(result);
```

This script declares and initializes three variables, `newNum`, `newNum2`, and `result`. The first two are given the value of 7 and 2, respectively. The third variable, `result`, is a calculation, subtracting the variable `newNum2` from the variable `newNum`. An alert window is then called to display the results of the third variable, `result`, which produces a value of 5.

The CULM Guide to JavaScript Programming: Part 3

Multiplication Operator (*)

This operator is used to multiply the value on the right of the operator by the value on the left, i.e., "7 * 2" would read "seven times two."

```
var newNum=7;
var newNum2=2;
var result=newNum*newNum2;
alert(result);
```

By now this script should be self-explanatory. Try it in your template.

Division Operator (/)

This operator is used to divide the value on the left of the operator by the value on the right, i.e., "8 / 2" would read "eight divided by two."

```
var newNum=8;
var newNum2=2;
var result=newNum/newNum2;
alert(result);
```

Be careful that you do not divide by zero. The result will be either *infinity* or *undefined*, depending on the browser you are using. Try the following in your template and see what answer you get:

```
var newNum=8;
var newNum2=0;
var result=newNum/newNum2;
alert(result);
```

Calculation Conversions

An integer added to a floating point number will result in a floating point number. It's important to remember that JavaScript treats all numbers as *floating point numbers* (numbers that have a decimal point). This is important because if you add 13 and 27.235, you will get a floating point number as the result: 40.235. (There is a way to control the display of numbers after a decimal point which we will look at later.)

If you add a number to another number which is in quotes (which is actually a string and not a number), it will return a string as there is no way to add a "non-number" to a number:

```
var myNum=10;
var noNum="13";
var theSum=myNum+noNum;
alert(theSum);
```

The result here would be 1013 as the script would convert the variable `myNum` to a string due to the other variable, `noNum`, already being a string. It would then place the two together, just as when we added text to a string:

```
var dylan="Bob Dylan";
document.write("I like music by "+dylan);
```

(The processing of joining two or more strings together is called "concatenating".)

These calculation conversions apply to addition, subtraction, multiplication, and division operators.

Modulus (aka 'Modulo') Operator (%)

This operator is used to divide the value on the left of the operator by the value on the right, and then give the result of the remainder of the calculation:

```
var newNum=11;
var newNum2=2;
var result=newNum%newNum2;
alert(result);
```

This script declares three variables, `newNum`, `newNum2`, and `result`. The first two variables are given the values of 11 and 2, respectively. The third variable is instructed to divide the first variable, `newNum`, by the second variable, `newNum2`, and return the remainder. An alert window is then called to display the results, which in this case would be 1.

Increment/Decrement Operator (++/--)

The increment and decrement operators are used to either increase or decrease the value of a variable, element, or property, usually within a loop (more on those later). Basically, an increment/decrement operator is a kind of shorthand. For instance, using the increment operator ++, we could say that x++ is the same as x = x + 1 and, using the decrement operator `--`, then x-- is the same as x = x - 1.

It's important to note that it *does* make a difference where you put the operator. If the operator is put *before* the value, it increases the value by 1, returns that value to the variable and *then* performs the rest of the operation:

```
var theNum=4;
alert(++theNum);
```

The variable `theNum` is declared and initialized with a value of 4. An alert window is then given a parameter which will increase the value of the variable `theNum` by 1, return it to (reset) the variable, and then display the result of the variable, which now has a value of 5. Try it in your template.

The same would be true if we used a decremental operator:

```
var theNum=4;
alert(--theNum1);
```

The parameter sent to the alert window here would decrease the value of the variable `theNum` by 1, reset the variable to a value of 3, and then finish the process, in this case an alert window would be displayed. The variable has now been assigned (reset to) a new value of 3.

To put it in plain English, when the operator is in front of the value it is read as: "add/subtract 1 to/from the value of the named variable, reset the variable to the new value, then continue to process the script."

On the other side (literally), if the operator is put *after* the value, the calculation returns the value of the variable to the script, then it increases/decreases the value of the variable (resets it) by 1, and resets the variable:

```
var theNum=4;
alert(theNum++);
```

In this case, the alert window will display a value of 4, and the incremental operator will then increase (reset) the value of the variable `theNum` to 5. The next time the variable `theNum` is used, its value will be 5. Try this again, only add the last line below to the script and see what happens:

```
var theNum=4;
alert(theNum++);
alert(theNum);
```

You should get an alert window with a value of 4, then another alert window with a value of 5.

The same would be true if we used a decremental operator:

```
  var theNum=4;
  alert(theNum--);
```

This script would return a value of 4, and then decrease the value of the variable `theNum` by 1 to 3.

Here is another way of looking at it:

```
  var a = 10;   // initialize a to 10
  var x = 0;   // initialize x to 0
  x = a;   // a = 10, so x = 10
  x = ++a;   // a becomes 11 before assignment, so a = 11 and x becomes
11
  x = a++;   // a is still 11 before assignment, so x = 11; then a
becomes 12
  x = a++;   // a is still 12 before assignment, so x = 12; then a
becomes 13
```

To recap: if the incremental/decremental operator is *before* the variable, it will increase its value by 1 and return the value to the script; if the incremental/decremental operator is after the variable, it will return its value to the script and then increase its value by one.

The CULM Guide to JavaScript Programming: Part 3

Comparison Operators

Operator	Name	Description
==	Equal	Returns true if both values are equal
!=	Not Equal	Returns true if both values are not equal
===	Strict Equals	Returns true if both values are equal and are the same data type (Added in JavaScript 1.5)
!==	Strict Not Equal	Returns true if both values are not equal or not the same data type (Added in JavaScript 1.5)
>	Greater Than	Returns true if the value on the left side of the operator is greater than the value on the right side
>=	Greater Than or Equal	Returns true if the value on the left side of the operator is greater than or equal to the value on the right side
<	Less Than	Returns true if the value on the left side of the operator is less than the value on the right side
<=	Less Than or Equal	Returns true if the value on the left side of the operator is less than or equal to the value on the right side

These operators are used for making comparisons which are true or false. These are pretty self-explanatory so I won't go into too much detail here. Notice that the single equal operator ("=") is not shown here. That is because it is an ***assignment*** operator. It's used for assigning a value to a variable or other type of container. i.e. `var myNum = 35`.

Comparison operators can be used for both numbers and strings. A value in a variable (number or string data type) can also be compared. The JavaScript interpreter will retrieve the value of the variable and then compare it with the other value.

Below are some comparisons. I've used several to give you a broader understanding of these comparison operators. Look at them very carefully and they should make sense.

- **25 == 25** // ("25 is equal to 25") *true*
- **25 == "25"** // ("25 is equal to 25, after it has been converted from a string to a number" [The JavaScript interpreter will convert the string "25" to a number, 25.]) *true*
- **25 != 25** // ("25 is not equal to 25") *false*
- **25 != 34** // ("25 is not equal to 34") *true*
- **25 === 25** // ("25 is strictly equal to 25") *true*
- **25 === "25"** // ("the number 25 is strictly equal to the string '25'" [In this case, the data type of the values is taken into account

and is not converted.]) *false*

- **25 !== "25"** // ("the number 25 is strictly not equal to the string '25'") *true*
- **25 > 24** // ("25 is greater than 24") *true*
- **25 >= 25** // ("25 is greater than or equal to 25") *true*
- **25 >= 15** // ("25 is greater than or equal to 15") *true*
- **25 < 37** // ("25 is less than 37") *true*
- **25 <= 25** // ("25 is less than or equal to 25") *true*
- **25 <= 15** // ("25 is less than or equal to 15") *false*

When comparing two strings, JavaScript converts each character to its ASCII value, beginning on the left and continuing to the right. Using the ASCII equivalent, a lower case letter is greater in value than an upper case one. For instance, in ASCII code "A"=65 and "a"=97. So, "A" > "a" would be false while "a" > "A" would be true.

Assignment Operators

Operator	Name	Description
=	Assign	Assigns the value on the right side of the operator to the variable on the left
+=	Add and Assign	Adds the value on the right side of the operator to the variable on the left, and then assigns the new value to the variable on the left
-=	Subtract and Assign	Subtracts the value on the right side of the operator from the variable on the left, and then assigns the new value to the variable on the left
*=	Multiply and Assign	Multiplies the value on the right side of the operator by the variable on the left, and then assigns the new value to the variable on the left
/=	Divide and Assign	Divides the variable on the left side of the operator by the value on the right side, and then assigns the new value to the variable on the left
%=	Modulus and Assign	Divides the variable on the left side of the operator by the value on the right side, and then assigns the remainder to the variable on the left

These operators are used for assigning values to variables, elements, and properties. Many of these operators are useful when utilized within loops, as some of them perform simple calculations on the variables. They are pretty much self-explanatory. They will come into play more when we begin studying loops. Examples of their use are shown below.

- Assign (=): `x = y`

(We've already used this in assigning a value to a variable, i.e. `var myNum = 15`)

- Add and Assign (+=): `x += y means the same as x = x + y`
- Subtract and Assign (-=): `x -= y means the same as x = x - y`
- Multiply and Assign (*=): `x *= y means the same as x = x * y`
- Divide and Assign (/=): `x /= y means the same as x = x / y`
- Modulus and Assign (%=): `x %= y means the same as x = x % y`

The CULM Guide to JavaScript Programming: Part 3

Logical (Boolean) Operators

Operator	Name	Description
!	NOT	Returns false if its single value can be converted to true; otherwise returns true
||	OR	Returns true if either value is true
&&	AND	Returns true if both values are true

Logical operators (also know as Boolean operators) are used for comparing two conditional statements. They are useful for checking a condition and then directing the script to proceed based on the outcome of that decision.

NOT Operator (!)

This operator is used to reverse the logical value of an expression. For instance, `!true` would evaluate as false. The expression says, "NOT true," so it is false. Then also, `!false` would evaluate as true. `!(25>13)` would evaluate as false. The expression says, "the statement 25 is greater than 13 is false."

OR Operator (||)

This operator returns true if the expression on either side of the operator is true. For instance, `23>43 || 25==25` is true as the expression on the right side is true. Again, `23>43 || 23==43` is false because both expressions are false.

AND Operator (&&)

This operator returns true if the expressions on both sides of the operator are true. For instance, `5==5 && 16==16` is true as both expressions are true. `27>=32 && 33==33` is false because the first expression is false.

Bitwise Operators

Operator	Name
&	Bitwise And
^	Bitwise XOR
|	Bitwise Or
~	Bitwise Not
<<	Left Shift
>>	Right Shift
>>>	Zero Fill Right Shift

For the most part, these are used in advanced scripts in context with CGI scripts and Java applets. We won't go into detail about these here.

Special Operators

Operator	Name	Description
?:	Conditional	A type of if...else statement. A condition is placed before the (?) and a value is placed on each side of the (:)
,	Comma	Used to process a series of expressions
delete	Delete	Deletes an object, property, or element from an array
in	In	Used to inspect the contents of a specified object
instanceof	Instanceof	Tests an object to see if it is of a specified object.
new	New	Creates an instance of an object
this	This	Refers to the current object
typeof	Typeof	Identifies the type of value being evaluated
void	Void	Evaluates an expression without returning a value

These operators are used for specific tasks and can help eliminate unnecessary code.

Order of Precedence

Mathematics and computer operations are very orderly. Everything has a certain function and an order in which it is performed. So it is in JavaScript, where each operation is

performed in a prescribed manner. Below is a chart showing from the highest level of precedence (that performed first), to the lowest (that performed last), all the operators we have looked at in this series (plus a few others). Those operators grouped together have the same level of precedence. You don't need to memorize them but do remember that each operation does has a precedence. If you're unsure about the order of precedence, you can use parentheses to define the order yourself.

Operator	Comments
( )	Overrides all others. Calculations done from the inside to the outside.
[ ]	
function()	
!	Not (Boolean)
~	Not (Bitwise)
-	Negation
++	Increment
--	Decrement
new	
typeof	
void	
delete	
*	Multiplication
/	Division
%	Modulus
+	Addition
-	Subtraction
<<	Left Shift (Bitwise)
>	Greater Than
>>	Right Shift (Bitwise)
<	Less Than
<=	Less Than or Equal To
>=	Greater Than or Equal To
==	Equals

!=	Not Equal
&	And (Bitwise)
^	XOR (Bitwise)
|	Or (Bitwise)
&&	And (Boolean)
||	Or (Boolean)
?	
=	Assign
+=	Add and Assign
-=	Subtract and Assign
*=	Multiply and Assign
/=	Divide and Assign
%=	Modulus and Assign
<<=	Left Shift by Value (Bitwise)
>>=	Right Shift by Value (Bitwise)
&=	And by Value (Bitwise)
^=	XOR by Value (Bitwise)
|=	Or by Value (Bitwise)
,	Comma

The CULM Guide to JavaScript Programming: Part 3

Let's take what we've learned and apply it to a few scripts.

First let's take a look at a script using the assignment operator (=) and the addition operator (+). This short script will hide your e-mail address from spammers but still allow your visitors to see it and click on it as a `mailto` link.

```
<script type="text/javascript">
<!--
  var user = "Troll";
  var site = "ilovemusicchurch.org";
  document.write('<a href=\"mailto:' + user + '@' + site + '\">');
  document.write(user + '@' + site + '</a>');
//-->
</script>
```

Let's look at the script, starting from the top. First, we initialize and declare two variables, `user` and `site`. We use the assignment operator (`=`) to assign the values, `lunderwood` and `jupitermedia.com`, to the two variables. These values are the user name ("`user`") and domain name ("`site`") of the e-mail address, without the "@" symbol.

Next, we use the built-in JavaScript function `document.write` to create a `mailto` link using our variables. First, we'll build the actual link. (Note that each string is enclosed in single quotes.) The `document.write` function is declared and then we give the value to be used by the function. We begin by creating a string with the HTML element used for e-mail links, `'<a href=\"mailto:'`. We have used the backslash (" `\` ") to tell the JavaScript interpreter that the next character (in this case a double quote), is to be interpreted literally. e.g. We have "escaped" the double quote. In other words, "print the double quote on the page." We then use the addition operator to join together ("concatenate") the rest of the link: the variable `user`, the "@" symbol, the variable `site`, the end of the HTML link element, `\">`, and a semi-colon. Notice that, again, we have used the escape sequence for the double quote.

On the next line we will display the text used for the link and input the end tag for the link element. Once again the `document.write` function is declared. From there it is just a matter of joining the pieces together. The variable `user`, the "@" symbol, the variable `site`, the closing HTML tag, and a semi-colon.

Let's look at another one. (This one is taken from *JavaScript: A Beginner's Guide*):

```
<script type="text/javascript">
<!--
  var paycheck=2000;
  document.write(paycheck+"<br>");

  paycheck+=2000;
  document.write(paycheck+"<br>");

  paycheck=paycheck-500;
  document.write(paycheck+"<br>");

  paycheck=paycheck*0;
  document.write(paycheck+"<br>");

  paycheck=paycheck+500;
  document.write(paycheck+"<br>");

  paycheck-=80;
  document.write(paycheck+"<br>");
//-->
</script>
```

Each line here will be displayed on the page using the built-in JavaScript function `document.write`.

- From the top of the script, we declare the variable `paycheck` and initialize it using the (=) operator with a value of 2000. We then print out the value of the variable `paycheck`. (Each line ends with the HTML element `<br>` to give a line return.)
- Next, we apply the add and assign operator (+=), which will add the number 2000 to the variable `paycheck` and assign the new value to the variable, making the value of the variable 4000.
- Next, the variable is assigned a new value of itself minus 500, giving the variable a value of 3500.
- Next, the variable is assigned the new value of itself multiplied by 0, resulting in a value of 0.
- Next we assign the variable the value of itself plus 500, giving the variable a new value of 500.
- Finally, we use the subtract and assign operator to subtract 80 from the value of the variable and then assign it to the variable, making the value of the variable now 420.

The CULM Guide to JavaScript Programming: Part 3

By Troll X

Let's look at one more script, using the increment operator (`++`). (This one is taken from *JavaScript: The Definitive Guide*.) This one looks simple but is a little more complex to describe. The main thing is to look at the work done by the increment operator:

```
<script type="text/javascript">
<!--
var count=0;
while (count<10) {
  document.write(count + "<br>");
  count++;
}
//-->
</script>
```

This little script will write the numbers 0-9 on your screen, each on its own separate line.

- The first line in the script declares the variable `count` and initializes it with a value of 0.
- Then a `while` statement is executed on the second line.
    - Basically, a `while` statement evaluates an expression, which is contained within the parenthesis following the reserved word `while`. In this case the expression would be `(count<10)`. If the expression is false (if the variable `count` is equal to or greater than 10), the script then moves on to the next section in the program (in this case it would end). If it is evaluated as true, a loop is begun which includes the statement within the brackets following the expression. The expression is then evaluated in a continual loop until it returns a value of false. At that time, the script moves on to the next section in the program.
- In this script, the variable `count` is evaluated the first time as being 0, which is less than 10 (`count < 10`), making the `while` statement true.
- The value of the variable is then printed to the document, a line break is entered on the page, and the value of the variable `count` is increased by one by the increment operator (`count++`).
- This is done 10 times (0-9). On the 11th time, the variable will be equal to the number shown in the `while` expression. (The variable `count` will have been incremented to a value of 10.)

- The statement is then evaluated as false because the variable `count` is now equal to 10. The script, in this case, is ended. Try it yourself and see what happens on your Web page.

That wraps it up for our guide. You might want to try analyzing some smaller, ready-made scripts on your own. Many can be found in the JavaScript section of Script Search and at JavaScript Source. Perhaps in the future we will take a look at some of these scripts and apply some of what we've learned. Even if we can't make sense of the entire script right away, we can pick out sections that we are familiar with to obtain a better understanding.

Further JavaScript classes and documents are available through the church as well as beginner and advanced classes/documents on all computer science topics, with an emphasis on object oriented programming.

Troll X
Principal MIS
CULM IT/IS Group
Church of Universal Love and Music
www.Ilovemusicchurch.org
closingtimeproductions@yahoo.com
Admin