JavaScript Skills Ladder, Lesson 1.3

# JavaScript Statements and Expressions

#### Lesson Objectives

### When you complete this skills ladder lesson, you will be able to:

- Use the JavaScript console.
- Incorporate basic JavaScript statements into your web pages.
- Use various JavaScript operators including arithmetic, increment, decrement, assignment, comparison and string concatenation operators.
- Use conditional expressions and statements.
- Nest your statements.
- Comment your JavaScript code appropriately.

## JavaScript Statement

Each line of JavaScript code that you write is called a statement. For example, x = 3 + 2i is a statement, as is alert(x)i. Throughout the course, you'll learn about other kinds of statements like if statements, while loops, and so on. A JavaScript program is a sequence of statements.

Take a look at this one:

### **OBSERVE:**

```
x = 3 + 2;
```

It's made up of four parts: x, =, 3 + 2, and the *i* at the end. All JavaScript statements end in a semicolon. The part of a statement that has a value, like 3 + 2, is called an *expression*, and its *result*, in this case the value 5, is *assigned* to the variable x.

Remember our first JavaScript program? It contained these two statements:

#### **OBSERVE:**

```
var age = 10;
var ageInDogYears = age * 7;
```

Notice that the right side of each statement is an expression. In the first statement, the result of the expression is just 10. In the second statement, JavaScript determines the result of the expression age \* 7 by taking the value of the variable age and multiplying it by 7. So the value of the variable ageInDogYears is 70 after JavaScript runs that statement.

These kinds of statements, with an = sign, are called *assignment* statements. There's a variable on the left side of the =, and a value or an expression that gets evaluated on the right side of the =. The resulting value is *assigned* to the variable.

var x = 3; is also an assignment statement. In this case you're declaring the variable x and initializing, or *assigning*, the value 3 to it.

# The JavaScript Console

Before you learn about any more JavaScript statements and expressions, it's time you learned how to use the JavaScript console. You can try all these statements and expressions by typing them into an HTML file and using alerts, but sometimes it's easier just to type them right into a console window. The other good reason to learn how to use the console window is that you can more easily find errors in your code! If you make an error in your JavaScript code, and load it into a browser window, chances are you'll never know you've got an error except for the fact that your page isn't doing what you expect. Using the console, you can get information about the error, and even what line number the problem is on.

One of the most common ways you'll use the console is with the console.log() statement. Until now, we've been using alert() to test our code and see values of variables and expressions, but console.log is a better way to do this. Let's create a quick program to see how it works.

## **CODE TO TYPE:**

```
<!doctype html>
<html lang="en">
<head>
    <title> Console Test </title>
    <meta charset="utf-8">
        <script>
        var x = 3;
        console.log("x is: " + x);
        </script>
</head>
<body>
        Testing the console!
</body>
</html>
```

Save it in your work folder as *consoletest.html*, and load the page in your browser. It displays the string "x is 3" in your console, showing you the value of the variable. All modern browsers have a console. How to open the console varies between browsers, and the menu options change over time. Generally you will find it under a menu options that has the word "tools" in the label. If you're struggling, just google, "How to open console in [browser name]."

Once you've got the developer console up and running in the browser of your choice, try typing in some statements and expressions and see what happens. Also, try making some errors, so you get used to the kinds of error messages you might see in your JavaScript programs.

Here are some examples of typing errors into the console:



If "undefined" shows up in the console, don't worry about it. It's just JavaScript showing you the result of typing in a statement like var x = 3; The result of the *statement* is undefined, even though the result of the expression is 3, and the variable x has the value 3. You can always test the value of x by typing x at the console prompt, and you should see 3 in this case. Here's how that would look in the console:

Net - CSS	🔹 🗖 JS 🔹 🔳 Web Developer 🔹
08:25:55.132	<pre>var x = 3;</pre>
08:25:55.135	▶ undefined
08:25:57.564	★ ×
08:25:57.567	▶ 3
	-
>	

Let's modify our program and use console.log to display the values of some expressions and variables in the console.

## **CODE TO TYPE:**

```
<!doctype html>
<html lang="en">
<head>
  <title> Console Test </title>
  <meta charset="utf-8">
  <script>
    var x = 3;
    console.log("x is: " + x);
   var quote = "Thomas Jefferson once said, 'We should never judge a
president by his age, only by his works.'";
   console.log(quote);
   console.log(6.5 + 2);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser.

Net 🔹 🗖 CSS	• 📕 JS • 🔳 Web Developer •	Posit	on• Filter	🔎 Clear 🗙
07:24:43.837	GET <u>http://</u>	.com/javascript1/consoletest.html [HTTP/1.1 304 Not Modif	ied 110ms]	
07:24:43.971	GET <u>http://</u>	.com/A2EB891D63C8/avg_ls_dom.js [HTTP/1.1 200 OK 16ms]		
07:24:44.003	× is: 3			<pre>consolethtml:8</pre>
07:24:44.007	Thomas Jefferson once said,	'We should never judge a president by his age, only by h	is works.'	<pre>consolethtml:10</pre>
07:24:44.010	8.5			<pre>consolethtml:11</pre>
>				
E Contra de				

Note You're welcome to continue using **alert()** instead of **console.log()** if you prefer, or if you haven't been able to get the console working yet.

# **Arithmetic Operators**

You're probably familiar with many of the *arithmetic operators* available in JavaScript, like addition (+), subtraction (-), multiplication (\*) and division (/). Let's take a look at some of these operators in action:

## **CODE TO TYPE:**

```
<!doctype html>
<html lang="en">
<head>
  <title> Arithmetic Operators </title>
  <meta charset="utf-8">
  <script>
    var length = 100;
    var sideOfSquare = length / 4;
    console.log(sideOfSquare);
    var discountPercent = 3;
    var retailCost = 49.95;
    var cost = retailCost - (retailCost * discountPercent/100);
    console.log(cost);
    var radius = 3;
    var circumference = 2 * Math.PI * radius;
    console.log(circumference);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *arithmetic.html* and load the page in your browser. You should see three values in the console:



Let's look at the script in detail:

**OBSERVE:** 

```
var length = 100;
var sideOfSquare = length / 4;
console.log(sideOfSquare);
var discountPercent = 3;
var retailCost = 49.95;
var cost = retailCost - (retailCost * discountPercent/100);
console.log(cost);
var radius = 3;
var circumference = 2 * Math.PI * radius;
console.log(circumference);
```

The first value (25) is the result of the variable length value (100) divided by 4.

The second is more complex because we combined multiple operators in one expression. Do you remember rules of *precedence* from algebra class? It's a good idea in JavaScript to use parentheses when combining operators so there's no ambiguity about which operators to evaluate first! In the case of computing the value of the variable cost, we tell JavaScript to first evaluate the expression retailCost (49.95) \* discountPercent (3)/100, and then subtract the result of that from the value of retailCost (again, 49.95: 49.95 - (49.95 \* 3/100) = 48.4515.

Finally, in the last example, we multiply a built-in JavaScript value, Math.PI, by a radius value to compute the circumference of a circle. Math is available in all browsers and has some common constants that may come in handy if you do a lot of mathematical computation in your programs.

Try some other arithmetic expressions, including the *modulus* operator (%). This operator is used to find the remainder of a division. Since any even number divided by 2 gives no remainder, one way you can use this operator is to determine whether a number is odd or even.

## **OBSERVE:**

```
var age = 49;
var oddOrEven = age % 2;
console.log(oddOrEven);
```

Because the variable age is odd, the variable oddOrEven will contain the value 1 (the remainder of dividing 49/2).

## Increment and Decrement Operators

Two other operators that you'll find useful are the increment (++) and the decrement (--). Try this:

### **CODE TO TYPE:**

```
<!doctype html>
<html lang="en">
<head>
  <title> Increment and Decrement Operators </title>
  <meta charset="utf-8">
  <script>
    var length = 100;
    length++;
    console.log("Length after ++ is: " + length);
    length--;
    console.log("Length after -- is: " + length);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *increment.html* and load the page in your browser. Do the values in the console match what you expect? You should see the values 101 and 100 as the results of the two console.log statements.

++ just adds 1 to the value of a numeric variable, so x++; is equivalent to x = x + 1; and says, take the current value of x, add 1 to it, and store the new value back in x. Likewise, -- subtracts 1 from the value of a variable. You'll see these in use a lot with loops, which we'll get to in the next lesson.

# More Assignment Operators

You've already seen one assignment operator, the equals sign (=), which is used in assignment statements to assign a value to a variable. There are quite a few more assignment operators that all combine assignment with an arithmetic operator, like +. For instance, the += combines = and + into one operator. Modify your file as shown:

# CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
 <title> Assignment Operators </title>
  <meta charset="utf-8">
  <script>
    var length = 100;
    <del>length++;</del>
    length += 1;
    console.log("Length after += 1: " + length);
    <del>length--;</del>
    length -= 1;
    console.log("Length after -= 1: " + length);
  </script>
</head>
<body>
</body>
</html>
```

E

Save it, and load the page in your browser. Compare the results to the previous listing results. They should be the same, because we're adding 1 and subtracting 1 using the += and -= operators. So, length++; and length += 1; both yield the same result: adding one to the value of length.

What happens if you try numbers other than 1? Try that now. Make sure you get the results you expect. Here's a list of assignment operators you can use as a reference for later:

Assignment Operator	Equivalent Expression	
x += y	x = x + y	
x -= y	x = x - y	

x *= y	x = x * y
x /= y	x = x / y
x %= y	x = x % y

# Comparison Operators

*Comparison operators* are used to compare values. You can use comparison operators to generate a boolean value (true or false), like this:

## CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Comparison Operators </title>
  <meta charset="utf-8">
  <script>
    var frankAge = 14;
    var jimAge = 16;
    var isFrankOlder = frankAge > jimAge;
    console.log("Frank is older? " + isFrankOlder);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *comparison.html*, and load the page in your browser. What value do you see for isFrankOlder? Is Frank older than Jim?

We assign the result of the expression frankAge > jimAge (which asks, is the value of frankAge greater than the value of jimAge?) to the variable isFrankOlder. In this case, the variable isFrankOlder contains the value false, because the value of frankAge is *not* greater than the value of jimAge.

The most common way you'll use conditional operators is in conditional expressions in *if statements* (which we'll get to later in this lesson) and in loops.

Here's a list of conditional expressions you can use as a reference for later:

Comparison	Operator	Example
equal	==	var x = 3 var y = 4 x == y returns false
not equal	!=	var x = 3 var y = 4 x != y returns true
equal value and equal type	===	var x = 3 x === 3 returns true x === "3" returns false
greater than	>	var x = 3 var y = 4 x > y returns false
less than	<	var x = 3 var y = 4 x < y returns true
greater than or equal to	>=	var x = 3 var y = 4 x >= y returns false
less than or equal to	<=	var x = 3 var y = 4 x <= y returns true

# Conditional Expressions and Statements

You know that a conditional expression is one that results in a boolean value, often by including a comparison operator. For example, 3 > 4 and (4/2) == 2 are conditional expressions.

You can use conditional expressions in *conditional statements* to control the flow of your code. So you can say, "If such-and-such is true, then do this, else do that." Let's take a look at an example.

**CODE TO TYPE:** 

```
<!doctype html>
<html lang="en">
<head>
  <title> Conditional Statements </title>
  <meta charset="utf-8">
  <script>
    var chocolateBars = 3;
    if (chocolateBars > 0) {
        console.log("I've still got some chocolate left!");
    }
    else {
        console.log("Oh no! I'm out of chocolate!");
    }
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *conditional.html* and load the page in your browser. You see the message, "I've still got some chocolate left!" because the value of the variable chocolateBars is 3, so the result of the conditional expression, chocolateBars > 0 is true. That means the statements in the first part of the if statement will run. Try changing the value of chocolateBars to 0 and see what happens.

You can use the *if statement* with or without the else. Here's a closer look at the various parts of the if statement with the else, and without it:



## **Statement nesting**

You can make JavaScript statements more complex by *nesting* them inside other statements.

```
CODE TO TYPE:
```

```
<!doctype html>
<html lang="en">
<head>
<title> Conditional Statements </title>
<meta charset="utf-8">
<script>
var chocolateBars = 3;
<u>var hungry = true;</u>
if (chocolateBars > 0) {
    console.log("I've still got some chocolate left!");
    <u>if (hungry == true) {</u>
        console.log("Eat a chocolate bar!");
```

```
chocolateBars--;
}
else {
   console.log("Oh no! I'm out of chocolate!");
   }
</script>
</head>
<body>
</body>
</html>
```

Save your file, and load the page in your browser. What console messages do you see? What's the value of chocolateBars after the code has run?

## The String Concatenation Operator

One last operator before you're done with this lesson: the *string concatenation* operator. *Concatenation* means you take two strings and stick them together, so you could, for instance, combine the strings "chocolate" and "bar" into the string "chocolatebar". You can include spaces, so you could combine a first name, "Jim", with a space, " ", and a last name, "Smith" to create "Jim Smith."

The string concatenation operator is **+**. But wait, you say! Isn't that the arithmetic operator for adding two numbers together?

It is *both*. This is an example of an "overloaded" operator. In most circumstances, it's fairly easy for JavaScript to figure out which operator you mean. So, for example, if you type var x = 3 + 4; JavaScript knows you mean to add two numbers together. But if you type var fullName = "Jim" + " " + "Smith"; then JavaScript knows you mean to concatenate strings together.

Try these examples now, in the JavaScript console or in a JavaScript program in an HTML file:

## **CODE TO TYPE:**

```
console.log(fullName);
   </script>
   </head>
   <body>
   </body>
   </html>
```

Save it, and load the page in your browser. Do you see what you expected in the console? Now try this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
<title> Concatenate strings </title>
<meta charset="utf-8">
<script>
var stringOrNumber = "3" + "2";
console.log(stringOrNumber);
</script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. What do you see in the console? Now change the code just a little:

### **CODE TO TYPE:**

```
<!doctype html>
<html lang="en">
<head>
    <title> Concatenate strings </title>
    <meta charset="utf-8">
        <script>
        var stringOrNumber = 3 + "2";
        console.log(stringOrNumber);
        </script>
</head>
<body>
</body>
</html>
```

Now what do you see in the console? Do you know why?

In the first example, "3" + "2", the double quotes make it pretty clear that we've got two strings we're concatenating together. The second example, however, isn't so clear. In this

case, JavaScript can't easily make the string "2" into a number, so it assumes you mean to concatenate strings together and converts the number 3 into a string "3", and then concatenates it with the string "2".

## Comments

One more quick thing before we close up this lesson. As you start writing more JavaScript code, it's good to know how to *comment* your code. That is, add lines of documentation right in your code so you can describe what you're doing. This helps both you and others read your code and understand it.

There are two ways to add comments. The first is used to comment out one line of code:

**CODE TO TYPE:** 

```
<!doctype html>
<html lang="en">
<head>
<title> Comments </title>
<meta charset="utf-8">
<script>
<u>// This is a comment!</u>
var stringOrNumber = 3 + "2";
console.log(stringOrNumber);
</script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. It should run just as before. Any text after the // is ignored! Try typing some other comments the console, like this:

```
> // This is a comment!
    var stringOrNumber = 3 + "2";
    console.log(stringOrNumber);
32
< undefined
> stringOrNumber
    "32"
> // I'm a comment
    undefined
> // I'm another comment
    undefined
>
```

Notice that again the result of the comment is "undefined," which is just fine. It just means that there's no value resulting from the comment. The // characters in a line indicate that everything following them is a comment, through to the end of the line. So you can write:

var stringOrNumber = 3 + "2"; // is the value a string or a
number?

And all the text after the // is a comment, which means it doesn't get executed.

What if you need more than one line of comments? You can use a multi-line comment:

## CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
<title> Comments </title>
<meta charset="utf-8">
<script>
/*
var stringOrNumber = 3 + "2";
console.log(stringOrNumber);
*/
</script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. Look at the console window. You should see nothing at all! Because both lines of code are commented out, neither of these lines is executed. You can also use this method to add multi-line documentation:

### **OBSERVE:**

```
/*
 * Add some documentation here.
 * More documentation here.
 *
 */
var stringOrNumber = 3 + "2";
```

Try adding some comments to your JavaScript code. Use them whenever you want to help you remember what you did and why.

You've learned about a lot of operators in this lesson. Operators are an important part of JavaScript and you'll use them a lot as you continue through the course. In the next lesson we 'll be using operators quite a bit, so make sure you understand the concepts discussed here before moving on to the next lesson.

You've also learned how to use the JavaScript console. This is an invaluable tool for learning JavaScript and, especially when combined with console.log, a great way to debug your JavaScript code if you've got errors. It's common to make typing errors as you learn a new language, so now you have a good way to track those down.

Play with the console, try out the statements, expressions, and operators you learned here. Have fun!

## More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.

![](_page_17_Picture_2.jpeg)

Copyright © 1998-2015 O'Reilly Media, Inc.

This work is licensed under a Creative Commons Attribution-ShareAlike 3.0 Unported License. See <a href="http://creativecommons.org/licenses/by-sa/3.0/legalcode">http://creativecommons.org/licenses/by-sa/3.0/legalcode</a> for more information.

The original source document has been altered. It has been edited to accommodate this format and enhance readability.