

Introduction

Welcome Back!

Every app (computer program) you use makes countless decisions as you interact with it. The ability to make decisions is one of the key factors that separates programming languages, like JavaScript, from markup languages, like HTML, and style sheet languages like CSS.

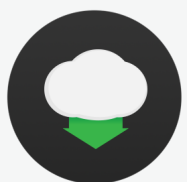
In this lesson, you're going to learn how to write JavaScript code that can make decisions. You'll need to be able to so you can write this kind of code to write more advanced interactive Web pages and applications. So the knowledge you gain here will pave the way for much bigger and better things to come—not just in this course, but for your entire software development career, should you choose to pursue this path.

We'll start by looking at the kinds of data you'll be using in these kinds of decision-making functions. As you'll see, JavaScript is quite picky with how you code your data in variables, whether the data be text, numbers, dates, or other types of information.

Then we'll look at how to code JavaScript to test data and make decisions. As you'll see, the common format of the decision you'll ask JavaScript to make takes the form of *if . . . then . . . else*: If *this* happens, then do *this*, or else do *this*.

But before we get too deeply into these decisions, let's learn about the types of data you'll be working with when you're writing this kind of code.

In this lesson, you'll get some hands-on practice with a variety of data types and decision-making processes.



Download

Below is a link to the ZIP file that contains three pages you can take for a spin in any browser:

- **Sample Numbers:** This sample demonstrates the use of numbers and arithmetic operators.
- **Sample Dates:** This sample demonstrates the use of dates.
- **Iftest:** This file incorporates decision-making using if statements



[Download Lesson-04_ClassProject.zip](#)

1 KB.(Gigabytes) ZIP

You're welcome to type these pages from scratch, if you're so inclined. Or, if you're not in the mood to practice typing right now, feel free to simply save them to your *Intro JavaScript* folder.

Chapter 1: Understanding Data Types

Data for Every Purposes

In our day-to-day lives, we all have some basic understanding of the difference between text, numbers, and dates. We don't necessarily think about them as being all that different, but we know to treat them a little differently. This is because we were taught to recognize and discern the difference between them as children—so as adults the process seems fairly intuitive.

Computers and programming languages don't have any intuition, so they can't handle different data types quite so effortlessly. Each data type needs to be defined, and then you have to spell out what needs to happen with it. That's what you'll learn to do in this chapter.

We'll be looking at how to work with some fairly simple kinds of data—like text and numbers—as well as some types of data that are a bit more involved. Let's keep it simple to start.

Strings (Text)

Most of us work primarily with text (words). In programming language, a chunk of text is often called a *string*. That's short for *a string of characters that have no specific numeric value*.

- A string can be any length and can contain any characters that you can type at the keyboard.
- When assigning a string value to a variable, enclose the text in single or double quotation marks.



Example

Here the code is assigning strings to variables named *name*, *address*, *csz*, *phone*, and *email*:

```
var name="Wanda O'Connor"
var address="123 Oak Tree Lane"
var csz = "Someplace, OH 43085"
var phone="555-555-1234"
var email="wanda555@gmail.com"
```

You can actually store anything as a string. Just about the only time you don't want to use the string is when you have data on which you want to perform arithmetic (or date arithmetic). We'll discuss this situation next.

Numbers

Numbers in JavaScript (and all programming languages) are quantities (also called *scalar values*) on which you can perform arithmetic calculations like addition, subtraction, multiplication, and division. Dollar amounts and other quantities are the most common uses. To qualify as a valid number in JavaScript, the data must meet the following requirements:

- The number can contain digits (0-9) and one decimal point (period).
- Commas, spaces, dollar signs, and other characters are not allowed.
- The first character in front of the number can be a hyphen to indicate a negative number. Hyphens are not allowed anywhere else.
- Never use quotation marks with numbers.

Here are examples of valid and invalid numbers.

Validity	Example	Why?
Valid Numbers	0	
	.5	
	1	
	-1	
	1.5	
	123456789.123456	
	1234.56	
Invalid Numbers	1,234.56	Contains a comma
	1234,56	Contains a comma

Validity	Example	Why?
	\$99.99	Contains a dollar sign
	18938-2802	Contains embedded hyphen
	(555)555-1234	Contains parentheses and embedded hyphens
	555-55-5555	Contains hyphens
	G4X 1A9	Contains letters and spaces
Valid and Invalid Numbers		



You can store an invalid number as a string.

After looking at the number options, you might realize that you can't store ZIP codes, postal codes, Social Security numbers, or phone numbers as numbers. But that's okay, because even though we often describe those things as *numbers* in everyday language, they're not true numbers in the sense of being scalar values. In other words, there'd never be any reason to do addition, subtraction, multiplication, or division with ZIP codes, postal codes, phone numbers, or Social Security numbers. So you don't lose anything by storing them as strings.

Dollar amounts should be stored as numbers if you intend to do any math with them—or combine them arithmetically. Just keep in mind that you can't put in a currency sign (such as a dollar sign) or commas. You have to store the dollar amount with just the numbers (0-9), one decimal point, and a leading hyphen for a negative number.

Data for Specific Purposes

Arithmetic Operators

To perform basic arithmetic on numbers, you'll need to use special symbols called *arithmetic operators*:

Operator	Function
*	multiplication
/	division
+	additon
-	subtraction (or negative number)
Arithmetic Operators	

JavaScript math can be as simple or as complex as you need it to be. But since it's usually pretty simple, we can start with the simple example you downloaded earlier: **Sample Numbers.html**

Sample Numbers.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Sample Numbers</title>
5 </head>
6 <body>
7   <script>
8     //Store a number
9     var qty = 5
10    //Store a number
11    var price = 4.99
12    //Multiply two numbers and store result in variable
13    var subtotal = qty * price
14    //Show multiplication result
15    document.write(subtotal)
16    //Type in a line break tag
17    document.write("<br>")
18    //Show multiplication result with 2 decimal places
19    document.write(subtotal.toFixed(2))
20  </script>
21 </body>
22 </html>
23
```

Let's look at the JavaScript code line by line, and then we'll discuss what happens when you view it in a browser.




Remember

While in this example we've provided specific names for each variable, they are just made-up names. You can use any names you like.

Text equivalent start.

Topic	Information

Topic	Information
<pre>var qty = 5</pre>	<p>First, we create a variable named <i>qty</i> (short for quantity). When executed, that line creates a variable named <i>qty</i> and puts the value <i>5</i> in as a number.</p> <div><h3>Number vs. String</h3><p>Here there are no quotation marks around the <i>5</i>. If there were, it would be treated as a string. That's not what we want here because we want do a little basic arithmetic, and you can't do arithmetic with strings. You can only do arithmetic with numbers.</p></div>
<pre>var price = 4.99</pre>	<p>This next line is similar but stores the number <i>4.99</i> in a variable named <i>price</i>.</p>
<pre>var subtotal = qty * price</pre>	<p>This line creates a variable named <i>subtotal</i>. This variable is defined as the product of multiplying the number in <i>qty</i> (<i>5</i>) by the number in <i>price</i> (<i>4.99</i>) using the <i>*</i> (multiplication) operator.</p>
<pre>document.write(subtotal)</pre>	<p>This line of JavaScript code types onto the page the value of the <i>subtotal</i> variable using the <i>.write</i> method of the document object. On the page, the result, is 24.950000000000003. This may seem like you did something wrong, but you didn't. JavaScript and most programming languages often default to carrying things out to extreme decimal places. Don't worry we'll show you how to fix that in a moment.</p>
<pre>document.write("br")</pre>	<p>To keep the numbers from appearing side by side, we used the <i>document.write()</i> to insert a <i>
</i> tag between the numbers.</p>

Topic	Information
<code>document.write(subtotal.toFixed(2))</code>	Here you'll notice that the JavaScript <code>toFixed()</code> method was added to show the subtotal number with only two digits after the decimal point.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.



Sneak Peek

So when you view the page in a browser, you see the JavaScript natural math result on one line, and the same number rounded to two decimal places right below it.

```
24.9500000000000003
24.95
```

There's a lot more that you can do with numbers, as you'll learn later in the course. For now, it's sufficient to understand that numbers do exist in JavaScript and that you can do math with them. Now let's look at another, more specific use of numbers.

Dates

JavaScript provides the ability to store and work with dates. For example, it can do things like calculate the date *x* days ago, or *x* days in the future (where *x* is some number). It can calculate the number of days between two dates. But in order for a date to be recognized as a date, the data must be stored as a *date object*.

The syntax for storing a date is a little different from that for storing strings and numbers. To store the current date and time, use this simple syntax:

```
var name = new Date()
```

Let's take a look at the code in the **Sample Dates.html** file you downloaded earlier, and then look at the results in the browser.

Sample Dates.html

```
1 <!DOCTYPE html>
2 <html>
3 <head>
4 <title>Sample Dates</title>
5 </head>
6 <body>
7   <script>
8     //Store current date time
9     var today = new Date()
10    //Store dates and times using different syntaxes
11    var birthday = new Date(1960,0,20)
12    var halloween = new Date("October 31 2013")
13    var valentines = new Date("February 14 2015 18:35:00")
14    //Display date time values on the page on separate lines.
15    document.write(today)
16    document.write("<br>")
17    document.write(birthday)
18    document.write("<br>")
19    document.write(halloween)
20    document.write("<br>")
21    document.write(valentines)
22  </script>
23 </body>
24 </html>
25
```

Text equivalent start.

Topic	Information
<pre>var today = new Date()</pre>	Here the variable name of your own choosing was set as <i>today</i> . This allows you to store the current date and time.
<pre>var birthday = new Date(1960,0,20)</pre>	<p>This is one syntax option for storing a specific date in a variable. Here the variable <i>birthday</i> provides a specific date using three separate values separated by commas: the year, month, and day (in that order):</p> <div><pre>var name = new Date(year,month,day)</pre></div> <p>Again, the variable <i>name</i> is something you get to make up yourself—we used <i>birthday</i>. The <i>year</i> is entered as a four-digit year number, the <i>month</i> as a month number in the range of 0 to 11 (where 0 is January and 11 is December), and the <i>day</i> is replaced with a number in the range of 1 to 31.</p>

Topic	Information
<pre>var halloween = new Date("October 31 2013")</pre>	<p>There's another syntax you can use to store a date, and it bypasses the 0 for January problem. That syntax is:</p> <pre>var name = new Date(datestring)</pre> <p>Here, <i>datestring</i> is a month name, followed by a space, followed by the day number, followed by a space, followed by the four-digit year. The date string should be enclosed in quotation marks.</p>
<pre>var valentines = new Date("February 14 2015 18:35:00")</pre>	<p>When using the datestring method, if you want to specify a time along with the date, follow the year with a space and a time expressed using military time (24-hour clock) in the format hh:mm:ss. Stay within the same pair of quotation marks. This date string would display as February 14, 2015, at 6:35 PM.</p>

Read the topic in the first column. Then read the second column for the information.

Text equivalent stop.



Sneak Peek

When viewed in a browser, the dates are displayed something like this, though the exact format of the dates and times will vary depending on your browser.

```
Wed May 27 2020 16:49:56 GMT-0400 (Eastern Daylight Time)
Wed Jan 20 1960 00:00:00 GMT-0500 (Eastern Standard Time)
Thu Oct 31 2013 00:00:00 GMT-0400 (Eastern Daylight Time)
Sat Feb 14 2015 18:35:00 GMT-0500 (Eastern Standard Time)
```

While your browser might not show the dates in the exact format above, let's take a moment to discuss the information you're seeing in this example because it tells you a bit about how JavaScript is actually storing each date in its variable:

- **Day of the Week:** Each date starts with a day abbreviation, which is calculated automatically.
- **Date:** Then comes the date displayed as mmm dd yyyy.
- **Time:** Next the time is displayed in 24-hour military format. If no time was specified, then the time shows as midnight, which is 00:00:00.

- **Time Zone:** After that comes the letters GMT, which stands for Greenwich Mean Time, a standard against which any time zone can be compared. The number that follows is the number of hours that the expressed time varies from GMT. For example, Eastern Standard Time (EST) is Greenwich Mean Time minus five hours. Eastern Daylight Time (EDT) is Greenwich Main Time minus four hours.

For now, try not to worry about the appearance of the date and time on the screen. What's important is that each value is stored as a date—a moment in time—in a variable.

As you'll learn later in the course, you can control exactly how it looks on the screen. You can also do date arithmetic, such as determining the date *x* days from now, the number of days between two dates, and more.

Now we're going to look at one last type of data.

Boolean Values

JavaScript, like all programing languages, also supports *Boolean values*. This is the simplest data type because it can only have a value of *false* or a value of *true*. There's no *maybe* or any other acceptable value.



Boolean

The term *Boolean* comes from the name George Boole, an English mathematician, philosopher, and logician, whose algebra based on the numbers 0 and 1 is the basis upon which all modern computing is based.

To set the value to *false*, you want to use the value:

```
var name = false
```

To set the value to *true*, use the following syntax:

```
var name = true
```

As always, you can replace *name* with a variable name of your own choosing. If you assign a value other than *true* or *false*, the value you assign will be converted to *false* or *true*.



True or False Values

If you assign any of the following values to a Boolean variable, then the value will be set to *false*:

- 0
- -0
- null
- ""
- undefined
- NaN

Assigning any other value will set the value to *true*. Any text in quotation marks will set the value to *true*. So be careful not to use quotation marks in assigning a value to a Boolean value.

Despite their simplicity, Boolean variables can be useful because there are many situations in programming in which the code has to make a decision about what to do next based on some value being *true* or *false*. As we mentioned at the start of this lesson, on any given day when you're using your computer, tablet, cellphone, or other device, the programs behind the scenes are making things happen. They do that by making thousands of *if . . . then . . . else* decisions based on Boolean true and false values.

Now that you know about Boolean variables and our other types of data, you're ready to learn how you write the kind of code that makes these decisions. Join me in the next chapter to learn how.

Chapter 2: Decision-Making

How Does Decision-Making Work in JavaScript?



Decision-making in JavaScript and all other programming languages boils down to being able to write, in code, simple decision-making steps such as *If this is the case, do this; otherwise, do that.*

We all make countless decisions like that in our daily lives, so this is nothing new. The trick is just learning how to express the same idea using a language that the computer can execute.

In this chapter, we're going to walk you through the most basic steps for writing this kind of code. As you'll see, it rests on something called an *if* statement. Let's take a look.

JavaScript if Statements

There are a few different ways to post if . . . then questions. Perhaps the simplest and most common is this syntax:

```
if (condition) code to execute;
```

We'll talk about what all of that means and provide many examples throughout this course. But for now, let's just focus on the syntax of the *if* statement.

- **Condition:** The *condition* will be the name of a variable that contains a *true* or *false* value or an expression that results in a *true* or *false* result.
- **Code to Execute:** The *code to execute* is one line of JavaScript code that's executed if (and only if) the condition is true. If the condition proves false, that line of code is ignored.



Important

Here are a couple of things to remember, that matter a lot:

- The word *if* must be typed in lowercase (typing one or both letters in uppercase is a common mistake).
- The condition must be enclosed in parentheses (forgetting one or both of them is another common error).

Let's use a simple example to start testing and practicing *if* statements.

Here are the Steps

1. Open *iftest.htm* in your code editor. (You saved this file to your Intro JavaScript folder at the beginning of the lesson.)
2. Look between the `<script>` tags. You should see the following statements:

```
var condition = true;
if (condition) document.write("Condition is true");
```

3. Now the page in a browser. The page should show the phrase: *The condition is true*.



Why?

In this example, the variable named *condition* contains *true* so the if statement is valid. This means that the *document.write("Condition is true")* is executed, putting the words *Condition is true* onto the page.

4. Now go back to *iftest.htm* in your editor. Change the condition variable assignment from *true* to *false*. It should look like this:

```
var condition = false;
```



Important

Do not change the if() statement or the document.write text.

5. Save the page after making the change.
6. Now open the page in a browser, or click the **Reload** or **Refresh** icon to reload the page. The page should now be blank.



Why?

You changed the condition to *false*, which means the if statement is no longer valid. The line of code that writes the words *Condition is true* on the page is executed only when the condition variable contains *true*. Since that variable now contains *false*, that line of code is ignored and, therefore, nothing is written into the page.



Using *if* Statements with Multiple Code Lines

The syntax you learned for *if* statements is very simple, and there are more ways to do it. If you want to execute two or more lines of JavaScript code when a condition provides true, you can just enclose those lines in a pair of curly braces, like this:

```
if (condition) {  
    code to execute;  
    code to execute;  
    ...  
}
```

The syntax is basically the same as before. The curly braces just create an entire *code block* that is executed if the condition proves true or ignored if the condition proves false. The ... is just shorthand for "however many lines of code you want to type", and in real life you would never actually type ... into your code.

If not, then what?

With this syntax, you can also create a block of code that is executed if (and only if) the condition proves *false*. The trick is to add the word *else* and a pair of curly braces to enclose the lines of code that are executed when the condition proves false, like this:

```
if (condition) {  
    code to execute;  
    code to execute;  
    ...  
} else {  
    code to execute;  
    code to execute;  
    ...  
}
```



You don't have to break out each line of code.

How you break up the lines isn't really important, so long as you don't break a line in the middle of a word. All that matters is that you have the lowercase *if*, followed by the condition in parentheses, and the curly braces for code to execute if the condition proves true. If you are incorporating an else statement, you would also include the word *else* followed by the code to execute enclosed in curly braces, if the condition proves false. The following code structure is also a valid syntax:


```
if (condition) { code to execute; code to execute; ....}

else

{ code to execute; code to execute; ...}
```

Feel free to break lines as makes the most sense to you and prevents errors. Keep in mind that most errors in typing *if* statements stem from forgetting parentheses or curly braces. So it's always a good idea to type up the code with all the parentheses and curly braces in place first. Then go in and fill in the actual code as a second step.

Adding an *else* Option

Let's try another hands-on exercise, using an *if* with an *else* in it.

Here are the Steps

1. Open the *iftest.html* page in your editor.
2. Add curly braces around the *document.write("Condition is true");* statement.
3. Then add the word *else* followed by curly braces.
4. In between the curly braces, type the following code statement: *document.write("Condition is false");*
5. When you are done, your code should look something like this:

```
<script>

  //Create a Boolean variable and give it a value
  var condition = false;

  //Use the variable as a test condition
  if (condition) {

    document.write("Condition is true");

  }
  else {

    document.write("Condition is false");

  }
</script>
```



Remember

As always, syntax counts a lot, and even the slightest typographical error can cause the code to fail. You want to make it easy on yourself to check that you have all of your parentheses, curly braces, and quotation marks in place. This layout makes it easy to see and check on all of the required parentheses and curly braces. Be sure the word *if* is typed with two lowercase letters.

6. Now, save your changes to the page. Then open it or reload or refresh it in a browser to try it out. This time, instead of doing nothing when the condition is false, the *else* code is executed, so the page shows *Condition is false*.
7. Now, open the page in your editor, and change the condition back to *true* like this:

```
var condition = true;
```

8. Again, save the page. Then open or **Reload** or **Refresh** the page in a browser. The page should show the phrase: *Condition is true*.

Of course, in real life, the condition is rarely just a variable named *condition* that contains a *true* or *false* value. You can use *if* statements to test for all kinds of conditions. For now, getting a little practice typing the code and an overview of the syntax and possibilities is enough to get you started.

Multiple ifs

You can nest *if* statements to perform different actions for different conditions using this syntax:

```
if (condition1)
{
    code that's executed if condition1 is true;
}
else if (condition2)
{
    code that's executed if condition2 is true;
}
else
{
    code that's executed if none of above are true;
}
```

You can have any number of conditions above the final *else*. Of course, that means there have to be multiple conditions to test.



Remember

Within any pair of curly braces, you can have any number of lines of code to be executed.

Up next we'll explore the different kinds of comparisons and logic you can use to test for a condition.

Chapter 3: Testing for Conditions

Decisions Based on Comparison

The JavaScript *if* statement, as in any programming language, is all about having the code make a decision about what to do next based on some situation or *condition*. In fact, the generic term for *if/else* statement decision-making in all programming languages is *conditionals*.

The condition is usually a test to see if some condition (or set of conditions) is are met (true) or not met (false). This often involves comparisons. To perform tests for conditions, we use *expressions* with *comparison operators*.



Example

The following expression is used to state that *the score is greater than 59*:

```
score > 59
```

Here *score* is presumed to be a variable that contains some number.

- **If the score is greater than 59**, then the expression returns *true* (the condition is met).
- **If the score is less than or equal to 59**, then the expression returns *false* (the condition is not met).

The *greater than* symbol (>) is one of several comparison operators that you can use to write expressions.

In this chapter, you'll learn about other ways to write expressions that test for true or false conditions. Let's start by examining some of the comparison operators you'll be using.

How Comparison Operators Work

Comparison operators provide the means of comparing things so JavaScript code can make decisions. To help you wrap your head around this, let's start by looking at the most commonly used comparison operators in JavaScript.

Common Comparison Operators

We'll use `x` to indicate the name of some hypothetical variable to illustrate the syntax. In real life, you'd replace `x` with any variable name or value.

Operator	What it Means	Example
<code>==</code>	is equal to	<code>x==10</code>
<code>!=</code>	not equal to	<code>x!=10</code>
<code>></code>	is greater than	<code>x>10</code>
<code>>=</code>	is greater than or equal to	<code>x>=10</code>
<code><</code>	is less than	<code>x<10</code>
<code><=</code>	is less than or equal to	<code>x<=10</code>
Comparison Operators		

Put It Into Practice

Let's assume we assigned a value of 10 to the `x` variable. Based on the expression provided, determine if the following statements return *true* or *false*.

Text equivalent start.

Question	Choices	Answer
The expression <code>x<10</code> returns(blank).	<div>a. True</div> <div>b. False</div>	<div>a. True Incorrect. This statement returns false because <code>x</code> equals 10, therefore it's not less than 10.</div> <div>b. False Correct!</div>
The expression <code>x<=10</code> returns(blank).	<div>a. True</div> <div>b. False</div>	<div>a. True Correct!</div> <div>b. False Incorrect. This statement returns true because <code>x</code> equals 10, therefore it's less than or equal to 10.</div>

Question	Choices	Answer
The expression x>=10 returns(blank).	a. True b. False	a. True Correct! b. False Incorrect. This statement returns true because x equals 10, therefore it's greater than or equal to 10.
The expression x!=10 returns(blank).	a. True b. False	a. True Incorrect. This statement returns false because x does equal 10, so the statement x doesn't equal 10 is false b. False Correct!
The expression x==10 returns(blank).	a. True b. False	a. True Correct! b. False Incorrect. This statement returns true because x does equal 10
The expression x>10 returns(blank).	a. True b. False	a. True Incorrect. This statement returns false because x equals 10, therefore it's not larger than 10. b. False Correct!
Instructions: Read the question in the first column, and consider which response from the second column completes the blank in the sentence best. Then read the third column to find out whether you were right.		

Text equivalent stop.

Comparison Operator Syntax

Now let's take a look at how comparison operators look in a sample bit of JavaScript code:

```
var age = 17

if (age >= 18) {

    document.write("You are an adult")

}else{

    document.write("You are a minor")

}
```

Let's look at each line of code to make sure you understand.

Text equivalent start.

Topic	Information
<code>var age = 17</code>	The variable <i>age</i> has been assigned a value of 17.
<code>if (age >= 18)</code>	The condition <i>age</i> <i>>=18</i> is only true if the variable age contains a number that's greater than or equal to 18.
<code>{ document.write("You are an adult")}</code>	If the condition <i>age</i> <i>>=18</i> is true, the page will display the phrase: <i>You are an adult</i> .
<code>else{ document.write("You are a minor")}</code>	If the condition <i>age</i> <i>>=18</i> is false, the page will display the phrase: <i>You are a minor</i> .
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.

Sometimes testing for a condition isn't as simple as testing for equality, inequality, or greater than or less than. Sometimes, as in real life, there are multiple things to consider. For those situations, JavaScript (like all languages) offers *logical operators*. Let's look at them.

How Logical Operators Work

JavaScript logical operators allow you to test for multiple conditions and see if all or some of them are true or not true.

Common Logical Operators

In the examples below, x and y are hypothetical variables used only to illustrate the syntax of creating expressions using these operators

Operator	What it Means	Example	Meaning
&&	and	<code>x==10 && y < 50</code>	x equals 10 <i>and</i> y is less than 50 (both are true)
	or	<code>x==10 y < 50</code>	x equals 10 <i>or</i> y is less than 50 (either or both are true)

Operator	What it Means	Example	Meaning
!	not	!(x==10 && y<50)	Not true that x equals 10 and y is less than 50
Examples of how Common Logical Operators are used			

Logical Operator Syntax

Here's a hypothetical example of a more complex *if . . . then . . . else* scenario where the code is choosing to display some text based on the contents of a variable named *age*.

```
var age = 65

if (age <= 12) {
    document.write("Child fare")
}

else if (age > 12 && age < 60) {
    document.write("Adult fare")
} else {
    document.write("Senior discount")
}
```

Let's step through the code so you can see that, despite the fact that it looks like total nonsense to the untrained eye, it's actually pretty simple.

Text equivalent start.

Topic	Information
var age =65	This creates a variable named <i>age</i> and stores the value <i>65</i> in that variable.
if (age <= 12) {document.write("Child fare")}	This says <i>if age is less than or equal to 12, write "Child fare" onto the page</i> . Since age is 65, well over 12, this action doesn't happen.
else if (age > 12 && age < 60) {document.write("Adult fare")}	This checks to see if <i>age</i> is greater than 12 <i>and</i> also less than 60. While 65 is greater than 12, it's not less than 60, so that's not true either. That means this bit of JavaScript is skipped over.
else {document.write("Senior discount")}	Since neither of the previous conditions proved true, that leaves only this option, and so the words <i>Senior discount</i> are written on to the page.
Read the topic in the first column. Then read the second column for the information.	

Text equivalent stop.



Lesson 4 Assignment

In the assignment for this lesson, you'll create a page similar to this one but make it a bit more interactive by allowing the users to type their age in a textbox on the screen.

The important thing to keep in mind is that there are many ways to write JavaScript code that makes decisions. Now, let's review the important concepts that you've learned in this lesson.

Review

To create highly interactive Web pages and applications, you need a programming language that stores and works with different kinds of data (information). In this lesson, we reviewed how to create code using JavaScript that makes the kinds of decisions needed to enable this kind of interactivity.

- **Understanding Data Types:** We started by covering the different kinds of data:
 - Strings are textual data like names and addresses.
 - Numbers are scalar values on which you'd likely perform arithmetic operations like addition, subtraction, multiplication, and division. Quantities and prices are common numbers.
 - Dates are moments in time like 12/31/2015 or 12:00 a.m.
 - Booleans are values that can either be true or false.
- **Decision-Making:** Then we examined the basic *if . . . else* logic that allows JavaScript code to make decisions about what to do based on values stored in variables and other conditions.
- **Testing for Conditions:** Finally, we explored the use of comparison operators and logical operators to allow JavaScript to test for multiple conditions. Understanding this logic will expand your creative prowess for developing all kinds of software for all kinds of applications, not just websites.

In the next lesson, we'll dig deeper into these concepts and start exploring practical applications for the many concepts you've learned so far.

Lesson 4 Assignment

For today's assignment, we're going to give you a little more hands-on experience with JavaScript, this time focusing on data types and *if* statements. You'll use some of what you've learned in previous lessons, and we'll show you a couple of new tricks as well.



Download

The pages that you created in the lesson were very simple. For this assignment, we're going to pull everything you learned together.

For this assignment, you'll need the following files:

- Basic Math.html
- Basic Math.txt



[Download L04-Assignment.zip](#)

1.3 KB (Gigabytes) ZIP

You can retype or copy and paste if you'd like to run through the code, or you can simply download and save the zip file to your *Intro to JavaScript* folder, then extract the files.

Right now the *"results"* table that you might have noticed in the code is initially invisible because we've made it that way with *display:none* in the CSS. Your challenge is to make the following happen when the user clicks the **Calculate** button:

- Put the extended price (quantity times unit price) in the cell named *tdTotalSale*.
- Put the sales tax amount (the extended price times the sales tax rate divided by 100 for Percent) into the cell named *tdSalesTax*.
- Put the grand total (extended price plus sales tax amount) in the cell named *tdGrandTotal*.
- Make the results table visible using this JavaScript statement:

```
document.getElementById("results").style.display="block";
```

Here are the Steps

1. Open the *BasicMath.html* page in a browser.

Enter a quantity:	<input type="text"/>
Enter a unit price:	<input type="text"/>
Enter a sales tax rate (%):	<input type="text" value="5.75"/>
<input type="button" value="Calculate"/>	

- 2. Enter a value in the **Enter a quantity** input box
- 3. Enter a value in the **Enter a unit price** input box.
- 4. Click the **Calculate** button. You'll notice that nothing happens. Let's fix this.
- 5. Open the *BasicMath.html* page in your code editor.
- 6. Between the script tags, add the function calculate:

```
function calculate() {}
```

7. Now we need to make sure our code knows when to perform this function. So, find the button tag and add the following event handler:

```
onclick="calculate() "
```

8. Ok, now we're ready to start defining our variables. Let's start by defining the variables that will be input in the table that is visible on the page:

```
var quantity = document.getElementById("tbQuantity").value;
var unitprice = document.getElementById("tbUnitPrice").value;

// Sales tax percent is textbox value divided by 100
var saletaxpercent = document.getElementById("tbSalesTaxPercent").value / 100;
```

9. Next, we need to define the math that needs to be performed:

```
// Do the math

var extendedprice = quantity * unitprice;

var saletaxamount = extendedprice * saletaxpercent;

var grandtotal = extendedprice + saletaxamount;
```

10. Now we want to make sure the results are displayed in the *"results"* table:

```
// Populate the results table cells

document.getElementById("tdTotalSale").innerHTML = "$" + extendedprice.toFixed(2);

document.getElementById("tdSalesTax").innerHTML = "$" + saletaxamount.toFixed(2);

document.getElementById("tdGrandTotal").innerHTML = "$" + grandtotal.toFixed(2);
```

11. Save the page and open it in your browser.

12. Now enter "0" in for both quantity and unit price.

13. Then click the **Calculate** button. This should cause a table to appear displaying the following:

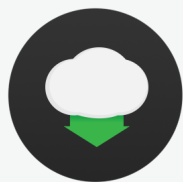
- Total Sale: \$0.00
- Sales Tax: \$0.00
- Grand Total: \$0.00

Enter a quantity:	<input type="text" value="0"/>
Enter a unit price:	<input type="text" value="0"/>
Enter a sales tax rate (%):	<input type="text" value="5.75"/>
<input type="button" value="Calculate"/>	

Total Sale:	\$0.00
Sales Tax:	\$0.00
Grand Total:	\$0.00

14. Go ahead and experiment with different number combinations to see how it changes your results.

How did you do? If you have any trouble, you can watch the following video doing it with VS Code:



Download

To check your code, you can download the correct code files here:



[Download L04-Answers.zip](#)

1.9 KB.(Gigabytes) ZIP

Feel free to visit the Discussion Area and ask your questions or share your concerns.

Lesson 4 Resources for Further Learning

JavaScript Data Types (https://www.w3schools.com/js/js_datatypes.asp)

https://www.w3schools.com/js/js_datatypes.asp

Here's a quick tutorial on the number, string, and Boolean data types (sometimes called the primitive data types).

Performing Basic Math in JavaScript (<http://www.scriptingmaster.com/javascript/performing-basic-math.asp>)

<http://www.scriptingmaster.com/javascript/performing-basic-math.asp>

This page offers a brief tutorial on JavaScript arithmetic with numbers.

JavaScript toFixed() Method (https://www.w3schools.com/jsref/jsref_tofixed.asp)

https://www.w3schools.com/jsref/jsref_tofixed.asp

Here's a brief reference to the JavaScript toFixed() method for numbers.

JavaScript Date Object (https://www.w3schools.com/js/js_dates.asp)

https://www.w3schools.com/js/js_dates.asp

Click this link for a tutorial on JavaScript date objects.

JavaScript Statements (https://www.w3schools.com/js/js_statements.asp)

https://www.w3schools.com/js/js_statements.asp

This page provides a brief tutorial on JavaScript statements and code blocks.

JavaScript if . . . else Statements (https://www.w3schools.com/js/js_if_else.asp)

https://www.w3schools.com/js/js_if_else.asp

Click this page for a reference to JavaScript *if . . . else* statements.

JavaScript if Statement (<http://www.tizag.com/javascriptT/javascriptif.php>)

<http://www.tizag.com/javascriptT/javascriptif.php>

Here's another brief tutorial on the JavaScript *if . . . else* statement.

JavaScript HTML DOM: Changing CSS (https://www.w3schools.com/js/js_htmldom_css.asp)

https://www.w3schools.com/js/js_htmldom_css.asp

Even though we didn't discuss changing CSS styles in the lesson, you saw in the assignment how that can come in handy. Here is a quick overview of that topic. You'll be seeing more of that in upcoming lessons.