Introduction

Welcome Back!

Photo thumbnail galleries are a common website feature for showcasing pictures of products. Some galleries use simple CSS hover techniques to enlarge an image on mouseover. Some allow a user to click an image or arrows alongside the image to cycle through the images. The latter are sometimes referred to as *carousels* because the user can scroll through the images in a circular fashion.

In today's lesson, you'll create just such a photo gallery. In the process, you'll learn about using JavaScript to change the source of any image tag on the page. And you'll learn still more tools and techniques for JavaScript creativity, including global variables and string manipulation. So come on over to Chapter 2, and we'll get right to it.

Creating an Image Gallery

For our working example, we'll start with a simple thumbnail gallery where you have a bunch of small pictures and one larger expanded image. Your job will be to code it so that clicking one of the smaller, thumbnail pictures reveals the larger version of the picture.

You can do something similar with CSS *:hover* and thumbnail images, however we'll be using the action of clicking rather than hovering the mouse pointer over the small thumbnail image to see the larger version. The clicking action is being used as a starting point and vehicle to learn some new JavaScript tools and techniques. We'll even add some buttons to cycle through the images in a carousel fashion.

Download

Below is the link to the ZIP file which contains the following resources you'll need for this lesson:

- **Carousel.html:** You're welcome to type this page 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 it to your *Intro JavaScript* folder.
- **Birds:** This is a set of bird photos that were sampled from WikiMedia Commons (https://commons.wikimedia.org/wiki/Main_Page) If you have your own image files, you're welcome to use them instead.
- **Icons:** These icons from WikiMedia Commons will be used to highlight navigation in your image gallery

Download Lesson-07_ClassProject.zip

5.6 MB (Gigabytes) ZIP

Let's get to it!

Chapter 1: Creating an Image Gallery

Adding Images to Your Gallery

You'll notice your carousel images have been placed in a folder named *Birds*, this is important because it needs to be in the code. If you open *Carousel.html* in your editor, you'll see how it's been coded in the <body> section:

```
<!-- HTML to display the pictures -->
<div class="carousel">
<img src=" birds/birds01.png" alt="" id="bigpic">
<div class="thumbs">
<img src="birds/birds01.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds02.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds03.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds04.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds05.png" alt="" onclick="showbig(this.src)">
</div><!-- End thumbs -->
</div><!-- End carousel -->
```

Here the image source shows the folder name *birds* followed by the individual file name (i.e. *birds/birds05.png*). Let's try adding the rest of the images.

Here are the Steps

1. Open *Carousel.html* in your editor.
2. Make some space after the last image listed in the section by pressing **ENTER**.
3. Then add the following code for the sixth image:

```
<img src="birds/birds06.png" alt="" onclick="showbig(this.src)">
```

4. Now go ahead and use the same syntax to add the rest of the images:

    - birds07.png
    - birds08.png
    - birds09.png
    - birds10.png
    - birds11.png

5. Save your progress.

If you did everything correctly, and the src= attributes in your img tags point to pictures that you actually have, then when you open *carousel.htm* in the browser, you'll see all of the additional pictures you just added.

Right now, clicking a picture *won't* make it the large image, because we haven't written the JavaScript code for that yet. But it's important that you make sure you have all of the pictures in the page, before we move on to that step.

Click Thumbnail to Expand Image

Adding the showbig Function

If you look at the <img> tags in *carousel.html*, you'll notice that each has an onclick event handler that looks like this:

```
onclick="showbig(this.src)"
```

Translated to English, that bit of code tells the browser, "When the user clicks this picture, call a function named *showbig*, and pass to it the value of the .src property of the image that's making the call." To clarify, let's break down this tag:

```
<img src="birds/birds01.png" alt="" onclick="showbig(this.src)">
```

Text equivalent start.

Topic	Information
`<img>`	This let's us know that we are dealing with an image.
`src="birds/birds01.png"`	Like all img tags, this one uses an src= attribute to indicate the source (location and file name) of the image that the tag should show on the page. So here it's pulling the *birds01.png* file in the *birds* folder.
`alt=""`	This means that the alt text for the image is *null* (or empty).
`onclick="showbig(this.src)"`	The keyword *this* in JavaScript always means *this element (or object). It's the element that is calling the JavaScript code*. In this case, it's the img tag. You may have already guessed that *this.src* means "the value of the src= attribute of this tag." When you click the image that has this event handler, the onclick event will call a function named *showbig* and pass to it the value of the src= attribute of the tag that's doing the calling.

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

Text equivalent stop.

We don't have a *showbig* function yet to test it out, so let's add that now.

Remember

JavaScript functions need to go between <script> . . . </script> tags in the header section of the page (between the <head> . . . </head> tags). Right under the <title> . . . </title> tags is as good a place.

Here are the Steps

1. Open *Carousel.html* in your editor.

2. Click just after the </title> tag, and press **ENTER** once or twice to make room for some new code.

3. Below that </title> tag (but still above the <style> tag), add your <script> . . . </script> tags.

4. Between your <script> tags, type or copy/paste the following JavaScript comment:

```
//Show clicked image in the large img tag
```

5. Under the JavaScript comment add the code for your function:

```
function showbig(pic) {
alert(pic);
}
```

Remember

The name of the function is *showbig*. As always, *showbig* is just a name we made up. After the function name, we have *(pic)*. The *pic* part is also just a variable name we made up. That variable will store whatever value gets passed by the calling tag. The function and variable names don't really matter, the important thing is that we are consistent in using the names we've assigned.

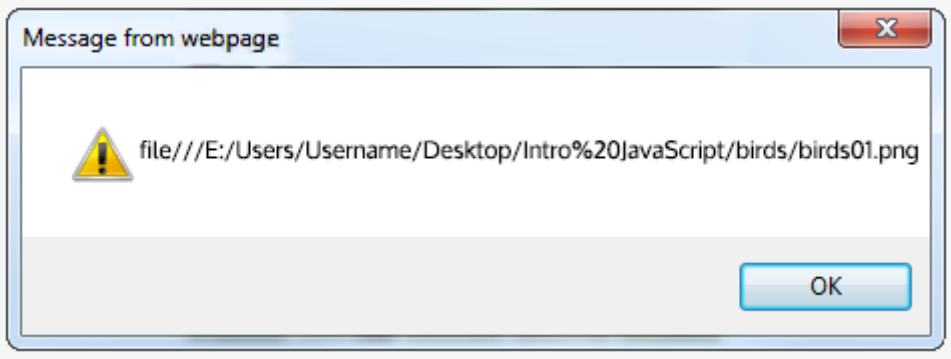
6. Save the page in your editor.

7. To verify that everything is working, let's take the code we have so far for a little spin. Go ahead and open (or refresh or reload) the *carousel.html* page in a browser.

8. Click a small thumbnail image.

Sneak Peak

If your code is right, an alert box should open showing the path to the file name of whatever image you clicked. You'll see the complete path, not just the relative path that you typed into the src= attribute. But that's not a problem or issue. That's just the way it works and will serve us well in our coding here.



Using Alerts to Debug Script

The alert we created is just something to allow you to see what value the *pic* variable receives when the user clicks a thumbnail. Using alert boxes to see what's going on behind the scenes is can be helpful in debugging code because it's an easy way to see what's in a variable. When you don't need the alert() line anymore, you can just delete it.

Defining the showbig Function

In real life, we don't want our function to show an alert box. Instead, we want it to trigger the selected thumbnail to display the larger image. To do this we need to change the source of the large image to whatever source information is being passed in from the calling tag so that the source (src) of the large image matches the source of the thumbnail image the user clicked.

Fortunately for us, that's easy to do because the JavaScript .src property is a Read/Write property. So you can use it both to see what source is currently assigned to an image and to change the source of an existing image. In our showbig function, we need to do two things:

- First, we have to tell the browser which image on the page needs to have its source changed.
- Second, we have to tell the browser what the source needs to be changed to.

The good news is that we've already taken care of the first step in *carousel.html* page. If you look at the * * you'll see that the larger image, is displayed using this tag:

```
<img src="birds/birds01.png" alt="" id="bigpic">
```

We gave that <img> tag an id= attribute of *bigpic*. The id attribute makes it accessible to JavaScript through a getElementById method. Once identified, we just have to set the src property of that image to the source that's being passed into the function. That we can do with a single line of code.

Here are the Steps

1. Open *Carousel.html* in your editor.
2. Go to the *function showbig(pic)* code.
3. Delete the temporary alert code we added earlier.
4. In its place, put in this line of code:

```
document.getElementById("bigpic").src = pic;
```

5. Save the page.
6. Open (or reload or refresh) *carousel.html* in your browser.
7. Click any thumbnail image.

If you did everything correctly, clicking a thumbnail image should make that same image show as the larger-sized image. So how you have a simple thumbnail gallery that allows the user to click a small thumbnail image and see it enlarged.

While our gallery is nice, there are lots of enhancements we can add. When writing code like this, it's always a good idea to write it so it's easily adaptable to other pages in case you want to reuse the code. We'll discuss this more coming up next.

Chapter 2: JavaScript Global Variables

Writing Adaptable Code

So, we've written some code for your image carousel. If there's even the slightest chance you'll want to use that same code elsewhere, you'll want to write it so it's easy to adapt it to other situations.

Our working example started out with five thumbnail images and we added six more. But there's no reason to lock yourself into any certain number of images. You can write the code so it's easy to adapt to pages where there might be more or fewer images.

Global vs. Local Variables

One way to increase the adaptability of your code is to store any value that might change from one application to the next in a variable. People sometimes refer to such variables as *global variables* because they're handled a little differently from most variables. To understand how and why requires a little general knowledge about the *scope* (or *lifespan*) of JavaScript variables, so let's start with that.

- **Most variables in JavaScript have a very short lifespan.** They're created inside some function and are *local variables*. They're local to the function in which they're created—they exist only while that function is running. As soon as the function finishes, all variables created within that function are erased from memory.

- **Most variables in JavaScript have very limited scope.** The *scope* of a variable defines what other code on the page can access the contents of a variable. The scope is similar to the lifespan of a variable because any given variable is visible only to the function in which it's created. In other words, if you define a variable inside a function, other functions on the page can't "see" the variable or access the contents of that variable.

- **Most variables only exist while needed.** Variables and the values assigned to them are stored in memory (RAM). Restricting variables so that they exist only when needed is a very efficient way to use memory. However, there may be times when you want to create variables that are accessible to two or more functions in a page. Such variables are often called *global variables* because they're visible to, and accessible to, all the JavaScript code on the page.

Memory Usage

Don't worry that you'll run out of memory or slow things down by using global variables. You'd need to create thousands of variables in a single page for them to have any noticeable impact on the speed or memory of modern devices. It's extremely unlikely you'd ever need to.

Creating a Global Variable

For our working example, we'll create a global variable named *maximages* to store the number of images in the gallery. Our *carousel.html* page has eleven images in its gallery. So in this page, we'll start it off with a value of eleven. We'll also include a comment to serve as a reminder as to the purpose of the global variable.

Here are the Steps

1. Open *carousel.html* in your editor.

2. Move the cursor to just after the opening *<script>* tag.

3. Type or copy/paste the comments shown below:

```
//The maximages global variable must equal
// the number of thumbnails in the gallery
```

4. We have 11 images in our gallery, so we'll set the value of var maximages to 11:

```
var maximages = 11;
```

5. Press **ENTER** once or twice to insert a few blank lines.

6. Save your progress.

As you'll see, the rest of the code will be written to assume that *maximages* always reflects the number of images in the gallery. So if you used this code in a thumbnail gallery with, say, 5 images, you'd just have to change *maximages=11* to *maximages=5*.

Adding Buttons to the Slide Show

Naming the Image Files

As a nice enhancement, we can add buttons to the left and right of the thumbnails so users can click to cycle through the images.

There are a few different ways we could write the code to accomplish this. One of the easiest is to simply name the image files so that we need only change one number in the file name to change the image. Let's give this approach a spin.

Here are the Steps

1. **Create a file to house similar images.** If you look how we've set up the files for you, you have a *birds* folder that holds the image files for the slideshow.

2. **Name the files with a similar pattern.** You'll see we've already taken the liberty of naming the files using the pattern *birds##.png*, where *##* is a two-digit number. All in all there are 11 images.

3. **Adhere to the naming convention in the code.** Each image's relative path starts with *birds/birds* (the folder and start of each file name) and a .png extension. So the tags in *carousel.html* look like this:

```
<img src="birds/birds01.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds02.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds03.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds04.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds05.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds06.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds07.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds08.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds09.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds10.png" alt="" onclick="showbig(this.src)">
<img src="birds/birds11.png" alt="" onclick="showbig(this.src)">
```

Adjusting this Code for a Different Page

If you were to use this same code in some other page, you could use a different folder name, a different file name, and a different extension. That's fine, so long as you stick with a two-digit number for each file name.

4. Move the cursor to just below the *var maximages = 11*; line.

5. First, create a global variable for the start of each relative path by adding the following code:

```
//Folder name and start of file name of each image file
  var startpath = "birds/birds"
```

6. Below that create a global variable for the file extension of each image:

```
  //Filename extension of each image
  var extension = ".png"
```

7. Save your progress.

It might seem odd that we're even bothering with these global variables now, since we don't have a whole lot of JavaScript functions in our code yet. But as we add more capabilities to our carousel, you'll see how handy it is to have that information up top in global variables and how they'll make it easier to adapt the code to different pages with different image carousels.

Adding the Carousel Buttons

Our next goal is to add a couple of arrows users can click to cycle through thumbnail images. For the arrows, you can use buttons we've included in the sample *icons* folder.

Here are the Steps

1. Go back to *carousel.html* in your text editor.
2. Add a new blank line below the * * tag.
3. Type or copy and paste this tag in the space:

```
<img src="icons/LeftArrow.png" alt="Previous" onclick="calcslide(-1)" >
```

4. Just above the * <!-- End thumbs -->* *tag,* type or copy and paste this tag:

```
<img src="icons/RightArrow.png" alt="Next" onclick="calcslide(1)">
```

5. Save the page in your editor.

Sneak Peak

The new tags you added should look as follows if you placed them in their proper place in the code.

```
<div class="thumbs">
<img src="icons/LeftArrow.png" alt="Previous" onclick="calcslide(-1)" >
 <img src="birds/birds01.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds02.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds03.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds04.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds05.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds06.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds07.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds08.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds09.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds10.png" alt="" onclick="showbig(this.src)">
 <img src="birds/birds11.png" alt="" onclick="showbig(this.src)">
 <img src="icons/RightArrow.png" alt="Next" onclick="calcslide(1)" >
</div><!-- End thumbs -->
```

6. Go ahead and open, reload or refresh the page in the browser. The arrows should be at the left and right sides of the thumbnails.

Clicking the arrows won't do anything until we write some JavaScript code to handle those events. Each button image uses an onclick event handler to call a JavaScript function named *calcslide*. The left arrow button passes to that function a value of -1; the right arrow button passes a value of 1. The calcslide function will need to decide which large image to show next based on the number that's passed. We'll show you how to make that bit of magic work next.

Chapter 3: String Manipulation

How String Manipulation Works

In this chapter, we'll get to work on the code that makes the left and right arrow buttons on the page work properly. As mentioned before, we intentionally used the common file name *birds*, followed by a two-digit number (such as *bird01.png*). This approach allows our code to switch from one image to the next using a method known as *string manipulation*.

The tags list the files in numerical sequence, with no gaps (*birds01.png*, *birds02.png*, *birds03.png*, and so forth.) That naming sequence makes it relatively easy to write code to indicate which image to show when the user clicks the arrow buttons.

While we can see the number hidden inside each filename, and we can easily add or subtract 1 from the number hidden inside any filename—computers can't. Not without a little help anyway. We have to spell it out for them with very simple step-by-step instructions.
If you remember, we added an event handler to the two buttons we added. The event handler calls a function called *calcslide*:

- For the right arrow, this function has a value of 1. The idea here, is that if the user is currently viewing *bird01.png* and clicks the right arrow, it will add 1 to the number inside the filename and show *bird02.png*.

- For the left arrow, this function has a value of -1. So, if the user is viewing *bird02.png* and clicks the left arrow to go to the previous image, it will subtract 1 from the number in the filename and choose *bird01.png*.

All we have to do is set up the yet to be defined function to do so. Fortunately, JavaScript offers ways to do this using string manipulation methods.

Searching within Strings

Finding One String in Another

JavaScript (like many programming languages) has an *indexOf* method, which can be used for finding the starting point of a small string contained within a larger string. The syntax is:

```
"largestring".indexOf("smallstring")
```

In practice, you'd replace *largestring* with a string of text (or the name of a variable that contains a string of text). The *smallstring* is also a string enclosed in quotation marks (or the name of a variable that contains a string).

- If *smallstring* doesn't exist in the *largestring*, the value returned by the indexOf method is -1.
- If *smallstring* does exist in the *largestring*, the value returned by the indexOf method is a positive number indicating the location of the smaller string within the larger string.

Here are some examples of values returned by the indexOf method, which illustrate how it works:

Example	Returns	Why?
"abcdefg".indexOf("a")	0	“a” is the first item (0) in the string “abcdefg”
"abcdefg".indexOf("b")	1	“b” is the second item (1) in the string “abcdefg”
"abcdefg".indexOf("cd")	2	“c” is the third item (2) in the string “abcdefg”
"abcdefg".indexOf("z")	-1	“z” does not appear in the string “abcdefg”
"abcdefg".indexOf("moose")	-1	“moose” does not appear in the string “abcdefg”
"bird01.png".indexOf(".png")	6	“.” is the seventh item (6) in the string “bird01.png”

The *indexOf* Method

The location is zero-based.

This means JavaScript always starts counting with zero rather than 1, like this:

That zero-based counting is counterintuitive and takes some time to get used to.

Getting a Portion of a String

Substring

JavaScript also includes some methods for extracting a portion of a string. The smaller string that you extract is sometimes called a *substring*. One of the methods you can use to extract a substring uses this syntax:

```
"string".substr(startposition,length)
```

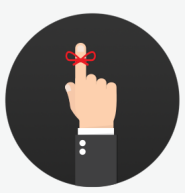
In that syntax, *string* is the larger string from which you're be extracting (or the name of a variable that contains the string). In your code, replace *startposition* with a number indicating the starting position of the substring (where 0 is the first character).

- If you include the .length argument, replace length with the number of characters to extract beyond that.
- If you omit the .length argument, the value returned is a substring starting at startposition extending to the end of the string.

Here are some examples of values returned by the .substr method, which illustrate how it works:

Example	Returns	Why?
"abcdefg".substr(0,1)	a	“a” is the first item (0) in the string “abcdefg”, and number of characters requested is “1”.
"abcdefg".substr(0, 3)	abc	“a” is the first item (0) in the string “abcdefg”, and number of characters requested is “3”.
"abcdefg".substr(2, 4)	cdef	“c” is the third item (1) in the string “abcdefg”, and number of characters requested is “4”.
"abcdefg".substr(6, 1)	g	“g” is the seventh item (6) in the string “abcdefg”, and number of characters requested is “1”.
"abcdefg".substr(4)	efg	“e” is the fifth item (4) in the string “abcdefg”, and the .length argument was omitted.
"bird01.png".substr(4,2)	01	“0” is the fifth item (4) in the string “bird01.png”, and the number of characters requested is “2”.
"bird01.png".substr(6)	.png	“.” is the seventh item (6) in the string “bird01.png”, and the .length argument was omitted.

The *.substr* Method



Remember

> When counting characters, you always have to start counting at 0 with the first character. So in *bird01.png, b* is character 0, *i* is character 1, *r* is character 2, *d* is character 3, and *0* is character 4. The substring that starts at that fourth position and is exactly two characters long is 01.

Slice

Another handy string manipulation method served up by JavaScript is the *slice()* method. This one is often used to slice off the first or last characters from a string using this syntax:

```
string.slice(start)
```

As with all string methods, the *string* part can be any string (or the name of a variable that contains a string). The start part is a number (or the name of a variable that contains a number) indicating where to start the slice. As always, the first character in the string is at number 0, not number 1.

- **If you specify a positive number for slice,** it slices from the start of the string toward the end of the string.
- **If you specify a negative number for slice,** it slices from the end of the string toward the start of the string.

The number of characters that are sliced, is based on however many characters you specify. Here are some examples of values returned by the .slice method, which illustrate how it works:

Example	Returns	Why?
"abcdefg".slice(1)	bcdefg	The number is positive 1 so “a” is cut.
"abcdefg".slice(4)	efg	The number is positive 4 so “abcd” is cut.
"abcdefg".slice(6)	g	The number is positive 6 so “abcdef” is cut.
"abcdefg".slice(-1)	g	The number is negative 1 so “abcdef” is cut.
"abcdefg".slice(-2)	fg	The number is negative 2 so “abcde” is cut.
"bird09.png".slice(-4)	.png	The number is negative 4 so “bird09” is cut.
"bird09.png".slice(6)	.png	The number is positive 6 so “bird09” is cut.

The *.slice* Method

New programmers are usually perplexed by string manipulation methods like *substr* and *slice* because it's difficult to imagine where and when such things would ever be useful. But there are in fact many situations where such things prove useful, as you'll see shortly.

Switching Strings and Numbers

Converting Strings to Numbers

As you may recall, JavaScript (and computers in general) treat strings and numbers as two different types of information. Strings are usually letters, words, or bits of text that provide useful information. They're not the kinds of things upon which you'd perform arithmetic operations like addition, subtraction, multiplication, and division. Numbers, on the other hand, are true scalar values upon which you *can* perform such operations.

To a computer, a filename like *bird01.png* is just a string, and it can't see the number inside. If you wanted to do some math on that number, you have to take the following steps:

STEP 1: Extract the numeric digits as a substring. However, a substring is still a string.

STEP 2: Convert the substring to a number. Before you can do any math on the substring, you have to convert it to a number.

JavaScript offers two methods for converting strings to numbers:

- parseInt(*string*): Returns the integer portion of string as a number (if possible).
- parseFloat(*string*): Returns the floating point portion of a string as a number (if possible).

Integer vs. Floating Point

The difference between an integer and a floating point is that an *integer* is always a whole number with no decimal point (such as 1 or -1 or 0 or 100 or 1000 or 18). A *floating point* number can have a decimal portion (such as 5.5 or 1.23 or 0.5 or 100.11 or 1234.56).

The parseInt() and parsefloat() functions return a number *if possible*. The string can only be converted to a number if the string starts with one of the following:

- a numeric digit (0-9)
- a hyphen (minus sign)
- a dot (decimal point)

If the string can't be converted to a number, then the function returns NaN (Not a Number). Here are some examples of values returned by the parseInt() and parsefloat() functions, which illustrates how they work:

Example	Returns	Why?
parseInt(01)	1	The parseInt() function removes the "0" from the string "01" to convert it to the integer "1".
parseInt(0.9)	0	The parseInt() function removes the ".9" from the string "0.9" to convert it to the integer "0".
parseInt(-1)	-1	The parseInt() function converts the string "-1" to the integer "-1".
parseInt(123.45)	123	The parseInt() function removes the ".45" from the string "123.45" to convert it to the integer "123".
parseInt(abcdefg)	NaN	The string "abcdefg" contains no numbers, therefore the parseInt() function cannot convert it.
parseFloat(01)	1	The parsefloat() function removes the "0" from the string "01" to convert it to the floating point "1".
parseFloat(0.9)	0.9	The parsefloat() function converts the string "0.9" to convert it to the floating point "0.9".
parseFloat(-1)	-1	The parsefloat() function converts the string "-1" to the floating point "-1".
parseFloat(123.45)	123.45	The parsefloat() function converts the string "123.45" to the floating point "123.45".
parseFloat(abcdefg)	NaN	The string "abcdefg" contains no numbers, therefore the parsefloat() function cannot convert it.

The *parseInt* and *parsefloat* Functions

Note

The parseInt () function always *truncates* (removes) the decimal portion of a number. It never rounds numbers. That's why parseInt("0.9") returns 0 (zero) rather than 1.

In some cases the value returned by *parseInt* or *parseFloat* looks exactly like the string it converted. However, it's important to remember that how the value looks is not important. Computers don't have eyes, so they only distinguish a string from a number. The key is that the value returned is stored as a number, not as a string. Unlike strings, you can perform mathematical operations (like addition, subtraction, multiplication, and division) on numbers.

Converting Numbers to String

Sometimes you may need to convert a number back to a string. The way that's usually done in JavaScript is to simply concatenate, with a + sign, the number to some string. It can even be an empty string.

Example

Imagine x is a variable that contains the number 1. You set variable y to equal an empty string + variable x using this line of code:

```
var y = "" + x
```

Upon execution, the variable y ends up containing the value "1" as a string rather than a number. As you know, a string can be anything. It can even be a numeric digit in quotes. Continuing with our example, let's say you execute the following code:

```
var z = "0" + x
```

Here variable z ends up containing the string "01". You can then concatenate that to a couple other strings, like this:

```
var path="birds/birds" + z + ".png"
```

Here you end up with an even longer string that contains *birds/birds01.png*.

Adding String Manipulation to the Slide Show

Pulling It All Together

So, let's see how we can combine the techniques in this chapter so that the user can scroll through the thumbnails with the left and right arrows.

Remember

The left arrow and right arrow images use the event handlers onclick="calcslide(-1)" and onclick="calcslide(1)" respectively.

We need to create a JavaScript function named *calcslide* that can accept the incoming value in the parentheses (-1 or 1). Inside that function, we need to calculate the next image to show by adding that positive or negative 1 to the number of the image that's currently showing.

Here Are the Steps

1. Open *carousel.html* in your editor.

2. Move the cursor to just under the *var extension = ".png";* line.

3. Press the **ENTER** key a few times to add some space.

4. Type or copy/paste the following code to define the calcslide() function:

```
//Calculates which picture to show next based on x
 //which is either 1 or -1
 function calcslide(x) {
 //Get file name of image that's showing
 var currentimage = document.getElementById("bigpic").src;
 //Locate file name extension in current image source string
 var dotat = currentimage.indexOf(extension);
 //Grab two digits to the left of that file name extension
 var stringnumber = currentimage.substr(dotat - 2, 2);
 //Convert stringnumber string to number and add x
 var nextnum = parseInt(stringnumber) + x;
 //If nextnum is less than 1, wrap around to maximages
 if (nextnum < 1) {
 nextnum = maximages;
 }
 //If nextnum is greater than maximages, wrap around to 1
 if (nextnum > maximages) {
 nextnum = 1;
 }
 //Create two-digit string from number (leading zero if less than 10)
 var twodigitnum = ("0" + nextnum).slice(-2);
 //Create new file name from two-digit number string
 var showimg = startpath + twodigitnum +extension;
 showbig(showimg);
 }
```

5. Save the changes in your editor.

6. Open or reload or refresh the page in the browser.

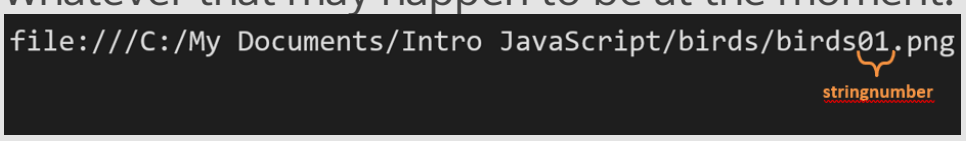
7. In the browser, you should be able to click the left and right arrows to cycle through the pictures. You can still click any picture to see it in the larger image; the arrows don't take that away.

Now let's talk about how it works. It all starts with the fact that the pictures are all in the same folder, and the filenames follow the pattern *birdXX.png,* where *XX* is a two-digit number. (So you could use that code with up to 99 images named *bird01.png* to *bird99.png*). You could use three digits and go to 999 pictures, but I don't think anybody in their right mind would try to show 100 or more thumbnails on a single page. You'd want 20 to 30 images per page, tops.

Text equivalent start.

Topic	Information
`function calcslide(x) {`	When the user clicks an arrow image, the browser calls the calcslide function and passes to it a 1 or -1, depending on which arrow was clicked. That number is stored in a variable named *x* in this line of code.
`var currentimage = document.getElementById("bigpic").src;`	Inside the function, this line is first executed to get the source (src) of the image that's currently showing as the large picture and stores it in a variable named *currentimage*. That source is a string that includes the filename of the image. Depending on the file path, the exact details of the source will vary. However, somewhere in that string is the filename of the image that's currently showing as the large picture. For an example, let's say *currentimage* looks something like this: `file:///C:/My Documents/Intro JavaScript/birds/birds01.png`

`var dotat = currentimage.indexOf(extension);`	This next line of code determines where the .png starts in that string and stores that number in a variable named *dotat*. In other words, the variable named *dotat* contains a number that pointing to the location in the string where the filename extension starts.
	`file:///C:/My Documents/Intro JavaScript/birds/birds01.png`
	`dotat`
	We already defined the extension variable as containing the value .png in earlier code. So the *dotat* variable ends up containing a number that points to the place where the .png extension starts.

Topic	Information
`var stringnumber = currentimage.substr(dotat - 2, 2);`	This next line of code grabs a substring that starts at two characters to the left of that extension and then goes for a length of two from there. In other words, after that line executes, the variable named *stringnumber* contains the two digits to the left of the dot, whatever that may happen to be at the moment.  In the case of our example, the stringnumber is 01. However, keep in mind that even though the 01 looks like a number to our human eyes, it's still a string to the computer.
`var nextnum = parseInt(stringnumber) + x;`	We plucked stringnumber out of a larger string using the .substr() method, which always returns a string. However, we can calculate the next image to show if we convert that string value to a number using the parseInt () function and then add the value of x (which was passed from the image button) to that. This line of codes handles that step.
`if (nextnum < 1) {nextnum = maximages;}`	The variable *nextnum* contains a number that's equal to *stringnumber*'s numeric equivalent plus the value of x. So it's either 1 greater or 1 less than *stringnumber*. However, now we're faced with a new problem. There is no image number 0, so *nextnum* cannot equal 0. In that case, we should cycle around to the last thumbnail, whose number equals the *maximages* variable. To make this happen we set it so that if nextnum is less than 1, it equals maximages.
`if (nextnum > maximages) {nextnum = 1;}`	Additionally, if that *nextnum* number is greater than the number of thumbnail images to show (maximages), then we should really wrap around and go back to show image 1. To do this we set it so that if nextnum is greater than 10, it equals 1.

Topic	Information
`var twodigitnum = ("0" + nextnum).slice(-2);`	At this point, the *nextnum* variable contains a number indicating the image to show as the big picture. We just need to get that number back to a string that we can use in the filename. This line of code converts *nextnum* to a two-digit number that has a leading 0 (if necessary).  **Note** That's a zero, not the letter O, in "0" + nextnum. So if *nextnum* were 1, then twodigitnum would be 01. But we don't want to limit the code to numbers less than 10, and that's where the slice comes in. If *nextnum* were a two-digit number like 10, then at first *twodigitnum* would be 010. But the slice(-2) takes only the last two characters of that result, so *twodigitnum* would end up being 10. Now we have a two-digit string that we can put into a filename.

Topic	Information
`var showimg = startpath + twodigitnum +extension;`	We need the full path to the image, which means we need the value that we previously stored in the *startpath* global variable, followed by the *twodigitnum*, and then the contents of the *extension* global variable.  **Remember** To concatenate (join) strings, you can use the + operator. After this line executes, the variable named *showimg* would contain something like birds/birds##.png, where ## is the two-digit number of the image to show next.  **Important** If your folder were named something other than *birds* and your filenames started with some letters other than *bird,* you'd have to change that code accordingly. Likewise, if your images weren't .png files, you'd have to adjust all the code to your image filename extensions as well. The logic would still be the same, but the specific text strings would change.

Topic	Information
`showbig(showimg);`	Anyway, at this point, the *showimg* variable contains the path of the next image to show, so this line of code passes that name to the showbig() function we created earlier. This is why, in your browser, you can use the left and right arrow buttons to go from thumbnail picture to thumbnail picture in either direction.
`}`	Don't forget! You need to make sure to close the code for the function.

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

Text equivalent stop.

Keep in mind . . .

The code we just showed you is *one* way to cycle through the images using JavaScript. It's not the *only* way. There are other methods that might be better suited to your style of thinking.

Many beginning programmers look at code like this and fear they'll never understand it. It takes time to train your mind to think like this, we're just not used to spelling things out in such small, discrete steps. However, its this sort of thinking and coding that allows us to take a blind, mindless machine and transform them into the complex, handy devices we know and love. Once you master this mode of thinking, you can make computers, tablets, and smartphones do anything you what you want. Each of these gadgets processes millions of tiny little steps every time we use them.

Let's head wrap things up and review what you've learned.

Review

In today's lesson, you learned about global variables and string manipulation.

- **Creating an Image Gallery:** We created a picture carousel that allows users to click a thumbnail to view an enlarged picture.

- **JavaScript Global Variables:** We used some global variables to define certain assumptions that the code makes about the number, location, and names of the picture files. That allows you to reuse the same code in a different page, changing only the values of the global variables to adapt the code to the new situation. This allowed us to add arrow buttons that we could use to cycle through images.

- **String Manipulation:** You learned how to search strings to isolate a specific segment (substring), and then convert it to a number. We used string manipulation to determine which large picture is currently showing. The code then does a little arithmetic and uses a little logic to decide which image to show next as the user cycles through the pictures.

We're not done with the picture carousel yet, though. In the assignment for this lesson, your challenge will be to adapt the code to another situation. We'll also add some new capabilities to the carousel in the next lesson.

Lesson 7 Assignment

For today's assignment we'll use techniques you've learned in this assignment to create another slide show. This one will be different in that it uses only small arrows in the images, not thumbnails, to scroll left and right. You may have noticed similar slideshows online.

Download

So you don't have to start from scratch, we've constructed a webpage that is already partially programmed for you. Below is the link to the ZIP which contains following files you'll need:

- The **Assignment-07_Webpage** folder: This folder contains *Assignment-07.html* and *Assignment-07.txt* which are the files that give you the HTML and CSS code you'll need to set up the initial appearance.
- The **Assignment-07_Images** folder: This provides you with 10 images you'll be programming into the slideshow.

Download L07-Assignment.zip

468.6 KB (Gigabytes) ZIP

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

If your head is spinning trying to figure out how to turn this into a functioning slideshow, that's to be expected. Watch the following video to learn how to write the appropriate JavaScript code, then follow the steps.

Here are the Steps

1. Open *Assignment7.html* in your code editor.

2. Go to the tag at the bottom of the page.

3. Add the following onclick event handler to the tag:

```
onclick="slide(-1)"
```

4. Then, add the following onclick event handler to the tag:

```
onclick="slide(1)"
```

5. Next, go to the <script> . . . </script> tags below the <style> block, but still in the <head> block.

6. Add the following JavaScript comment and code to create a image number global variable:

```
//These are global variables
 var imgnumber = 1;
```

7. Next, add a comment and code that creates a global variable for the number of images images:

```
// This must indicate the number of images in slideshow
 var imagecount = 10;
```

8. Now it's time to define the slide() function:

```
function slide(num) {}
```

Remember

Don't forget to add some space between the curly braces, so you have a few empty lines to work with.

9. Add the following code to your function so that the image number will increase or decrease depending on which arrow is clicked:

```
//Increment or decrement in range 1 to imagecount;
 imgnumber = (imgnumber + num) % imagecount;
```

10. Next, add the following code to your function so that if the image number is less than 1 it will go to the last image in the series:

```
// If number reaches zero, go to imagecount
 imgnumber = imgnumber < 1 ? imagecount : imgnumber;
```

11. Now you need to define file name and path for the new image to be displayed:

```
// Add two-digit number to image file name and path
 var newimg = "Assignment7/image" + ('0' + imgnumber).slice(-2) + ".jpg";
```

12. Finally, you need to have to program the code to swap out the current image for the new image:

```
// Change background image to the one
 //with the two-digit number in its filename
 document.getElementById("picture").style.backgroundImage = "url(" + newimg + ")";
```

13. Save your progress.

14. Then open *Assignment-07.html* in your browser to see if it works.

Download

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

Download L07-AssignmentAnswers.zip

2.2 KB (Gigabytes) ZIP

Lesson 7 Resources for Further Learning

The this Keyword (http://unschooled.org/2012/03/understanding-javascript-this/)

http://unschooled.org/2012/03/understanding-javascript-this/

Using the *this* keyword in JavaScript inside a tag to refer to the element that's calling the function is the easiest way to use that feature. But there are other uses, as this article explains.

Image src Property (https://www.w3schools.com/jsref/prop_img_src.asp)

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

This link takes you to a page that explains the JavaScript .src property.

JavaScript Functions (https://www.w3schools.com/js/js_functions.asp)

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

This page offers a nice discussion of JavaScript functions and includes some discussion on the lifetime (scope) of variables.

JavaScript Variable Scope (https://stackoverflow.com/questions/500431/what-is-the-scope-of-variables-in-javascript)

https://stackoverflow.com/questions/500431/what-is-the-scope-of-variables-in-javascript

This post is actually an answer to a question about the scope of JavaScript variables. Unlike most technical programming documentation, the answer to the original question presents a set of examples that illustrate the basic concepts with a few lines of code.

Variable Scope and the var Keyword (http://www.mredkj.com/tutorials/reference_js_intro_ex.html)

http://www.mredkj.com/tutorials/reference_js_intro_ex.html

Here's a brief tutorial on the scope of JavaScript variables.

JavaScript String Object (https://www.w3schools.com/jsref/jsref_obj_string.asp)

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

Click this link for a brief but useful tutorial on JavaScript string manipulation.

JavaScript indexOf() Method (https://www.w3schools.com/jsref/jsref_indexof.asp)

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

Here's a quick reference to the simple yet powerful JavaScript .indexOf method.

JavaScript substr() Method (https://www.w3schools.com/jsref/jsref_substr.asp)

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

This page serves up a nice explanation of the JavaScript .substr() method.

JavaScript slice() Method (https://www.w3schools.com/jsref/jsref_slice_string.asp)

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

As you may have guessed, this page is a good reference to the JavaScript .slice() method.

Font Awesome (https://fontawesome.com/icons?d=gallery)

https://fontawesome.com/icons?d=gallery

This site includes all the icons you can use from Font Awesome. The dark ones are free, and you can use them just by adding a link to the Font Awesome style sheet to your page, as we did in the Lesson 7 Assignment.