Learn More

For loops aren't the only way to iterate through an array in JavaScript. While loops, for instance, operate on the same principle: if a particular condition remains true, keep looping. If you feel confident, check out these resources and try rewriting the code in the exercise with another type of loop.

Task

DIRECTIONS SUBMISSION HISTORY

Estimated Task Time: 1 Hour.

For this Exercise, we ask that you submit the following:

1. A Gist with the contents of your JavaScript file demonstrating the following:
    - Dynamically generate an overview over your work examples with a for-loop in jQuery to dynamically change CSS properties in the DOM

2. With your submission, answer the following in a few short sentences:
    - What is the difference between comparison (=== and ==) and assignment (=)?
    - What does the bang operator (!) do?
    - What does the modulus operator (%) do?

Rubric

Refer to the categories below to see how to meet the requirements of the approved stage

Learning Goals

- Dynamically generate an overview of your work examples with a for-loop in jQuery
- Create, access, and iterate through objects in an array
- Explain the difference between Comparisons (=== and ==) vs. Assignment (=)
- Use the bang operator (!)
- Apply modulus to retrieve even numbered values
- Apply logic to for-loops

Estimated Read Time: 30 Minutes.

Introduction

Welcome back! You are over halfway through this Achievement, and are doing a great job making your way through this JavaScript-focused portion. Last time, we enhanced your event listeners, went over conditional statements, and more, to allow your users a nice flow when they navigate your Contact page.

This time we will cover arrays and more.

Accessing Array Elements

Remember how you can store values in variables to tell your computer program to remember them? An array is basically a list of items you need to store. Arrays are a data structure. You can put just about any type of data—string, integer, Boolean, object—into an array for later access.

Accessing Array Elements

Remember how you can store values in variables to tell your computer program to remember them? An array is basically a list of items you need to store. Arrays are a data structure. You can put just about any type of data—string, integer, Boolean, object—into an array for later access.

```
var arrayName = [item1, item2, ...];
```

Arrays are distinguished by three properties:

1. Array items are enclosed by square brackets [].
2. The items are comma-separated, like so: `["a", "b", "c", "d", "e"]`. There is no comma after the last item.
3. They're **zero-indexed**. That means each element in the array is located at a numbered position, and they start at zero instead of one. The index acts as a key: give us the key, and we will give you back the value at that index.

Open a new tab in Chrome and go to the Developer Tools Console. Set the array of letters above to a variable called "array":

```
var array = ["a", "b", "c", "d", "e"];
```

> TYPE OUT YOUR CODE INSTEAD OF COPYING AND PASTING!
> This will help your learning, and help you avoid random syntax errors caused by formatting.

Now, access the element at index 0, by typing the following right below, in your console, and pressing Enter:

```
array[0];
```

What will be at array[2]?

Let's see an array in real life. Go to this CodePen example and open your Console. Use jQuery to return an array of paragraph elements from the page, like so:

```
$("p");
```

See how these might be helpful? What happens if you access the array elements by index, like this?

```
$("p")[1];
```

How many `<p>` elements does this CodePen example have, anyway? Set the array of paragraph elements you just made to a variable, then call the `length` method on that variable, like so:

```
var pElements = $("p");
pElements.length;
```

Now you know!

Iterating Through Arrays; Comparison vs. Assignment

One way you might use an array is to do something to all or some of the elements stored in it. Stepping through each element of an array is called **iterating** through it.

A way to iterate over an array in JavaScript is a type of **loop** called a **"for" loop**. Open the Console in a new tab. Set that array of letters from before to a variable called "array" again. Enter the following into your console, making sure to press Shift + Enter to go to a new line in your console without submitting your input:

```
for(var i=0; i < array.length; ++i) {
  array[i] = array[i].toUpperCase();
};
```

Now type:

```
array;
```

You'll see the letters are all uppercase now.

What happened here? How did we loop through the array and apply toUpperCase() to each letter in it? Inside the parentheses, you'll find 3 statements, separated by semicolons, which tell the loop how to behave.

1. `var i = 0;` This is just a simple integer assigned to a variable. It will be our counter.

counter.

2. `i < array.length;` This expression must be true for the loop to keep going. When it's false, the loop stops running. `length` is a method that works on arrays as well as strings. Try `array.length;` in your console. Be careful with these statements, because you want to make sure your loop does stop at some point. A loop that goes on forever is called an infinite loop, and it can crash whatever is running your program.
3. `++i`, is JavaScript shorthand for `i = i + 1;` Every time the loop runs, it gets redefined, allowing it to act as a counter.

What about inside the curly braces? Every time the loop runs, `array[i]` refers to a different element in the array. The first time around, i is 0, so `array[i]` means `array[0]` and gives us "a". The second time, it returns "b", because i is now 1 and the item at index 1 in this array is "b". Each time, redefine whatever gets returned to make it uppercase.

To specify conditions for a particular thing to take place, as in a conditional or a loop, we often need to check if a statement is true. The single = sign in JavaScript is used for assignment: you assign a value to a variable with the single = sign. For comparison—to ask about a statement, "Is this true?"—we use the triple equal sign: ===.

Open your console and type:

```
2 + 2 === 4;
```

Now mix it up. Try `2 + 5 === 4;` Still good? Try the comparing the string "4" with the integer 4, like so:

```
"4" === 4;
```

You should get false because strings and integers are different types of data.

The double equals (==) also exists in JavaScript. It gives you a form of comparison that ignores type. `"4" == 4;` would return true. For a lot of purposes, this is not strict enough and should be avoided. Check out this chart showing how weird and random type conversions can be when using two equal signs instead of three.

Type `!false;` in your console. Now type `!true;` You can see that the **bang** (the exclamation point `!`) before the statement means "not-", so that a bang before a true statement, like `!(5 < 6)`, would return false, and vice versa.

Now to the real world.

Dynamically Display Your Work Samples

Remember the work samples we added to our website when we made it responsive? Now we are going to use JavaScript to make it easier to dynamically add and remove projects to and from your portfolio without having to change any of the HTML code.

In your index.html file go to the "Work" section of your website. It should look similar to this:

```
<div class="container">
  <div class="row" id="work">
    <div class="col-sm">
      <img class="img-responsive" src="img/kitty.jpg">
    </div><!-- / col -->
    <div class="col-sm">
      <img class="img-responsive" src="img/kitty.jpg">
    </div><!-- / col -->
    <div class="col-sm">
      <img class="img-responsive" src="img/kitty.jpg">
    </div><!-- / col -->
    <div class="col-sm">
      <img class="img-responsive" src="img/kitty.jpg">
    </div><!-- / col -->
  </div><!-- / row -->
</div><!-- / container -->
```

This is what mine looks like. You probably used different IDs, classes, image links, and/or even column widths. So don't worry if it doesn't look exactly the same.

All this HTML code makes it a little bit of a hassle to quickly add an image if I have a new work sample to show. We want to generate all the columns dynamically and store the images in an array inside a separate file. This way we can easily access it to make changes quickly.

In your *js* folder create a new file and call it *work.js*. Now link to it at the bottom of your *index.html* right before (!) the link to the *scripts.js* file.

```
...
  <script src="js/work.js"></script>
  <script src="js/scripts.js"></script>
</body>
```

In the *work.js* file create an array and call it `works`. This array is going to contain all our

```
  <script src="js/work.js"></script>
  <script src="js/scripts.js"></script>
</body>
```

In the *work.js* file create an array and call it `works`. This array is going to contain all our work samples. Whenever we want to add, remove or change one of our images, we can now do that inside this array. For demonstration purposes add at least four paths to images to this array. The paths can be local (e.g. `img/kitty.jpg`) or an internet link. If you need some example images you can take mine from the example:

```
var works = [
  "https://www.petfinder.com/wp-content/uploads/2012/09/Blog-Kitty-Cam-solo.jpg",
  "http://www.animal-photography.com/thumbs/red_tabby_long_hair_kitten_~AP-0UJFTC-TH.jpg",
  "http://www.animal-photography.com/thumbs/silver_tabby_kittens~AP-0JO6Y9-TH.jpg",
  "http://www.animal-photography.com/thumbs/silver_tabby_kitten_looking_up~AP-0DLVMB-TH.jpg"
]
```

Each image URL is inside a string and separated with a comma. Make sure there is no comma behind the last one, though!

Now let's test if everything worked and we can access the array inside the rest of our JavaScript. Inside the document ready function in the *scripts.js* file add a comment at the bottom to indicate the beginning of the "work" section. Below write a console.log to display the `works` array:

```
...
  // work section
  console.log(works);
});
```

Now reload the page and watch the console. You should see the array of image URLs displayed. If not, try to figure out what the problem is. Is there an error message? What does it say? Did you put everything in the right order?

If everything works we can go ahead and use what we learned about loops. We want to write a loop that starts counting at 0. It should run as long as the counter is lower than the amount of elements in the **works** array. And after each iteration it should add 1 to the counter variable.

Remove the `console.log` and replace it with the loop:

Remove the `console.log` and replace it with the loop:

```
// work section
for(var i = 0; i < works.length; ++i ) {
};
```

Remember, `.length` on an array, will give you the amount of elements in that array.

Next we are going to use jQuery to select the `div` with the class `row` and the ID `works` (your ID might be different). Then we will use the jQuery function .append() to add HTML content to our DOM inside that `div`. `append()` will add the content below all the other content, that might already exist inside that `div`. .prepend() would add it before the other content. `html()` would replace all content inside the `div`.

Copy one of the column divs (that contains one of your work samples) and paste it between quotes inside the parentheses of `append()`. There are a two things you need to pay attention to:

1. Between the parentheses, you technically can't have any line breaks, unless you use a backslash `\`. So whenever you have a line break, you need to add the backslash at the end of the line (see example below).
2. You will probably have to fix your quotes. In JavaScript technically it doesn't matter whether you use single or double quotes. The important part is, that if you use quotes inside quotes you can NOT use the same type of quotes. For example: `var text = "Bob says "Hi" !"` JavaScript will think the string will go from the first to the second quote: "Bob says ". It will not know what to do with `Hi" !"` anymore since the string is technically already over and `Hi` is not a JavaScript command that exists. Thus, the correct way to use the second pair of quotes is either this: `var text = "Bob says 'Hi' !"` or this `var text = 'Bob says "Hi" !'`

After finishing those steps go ahead and delete or comment out all the columns (including the one you just copied) in your index.html file, since are are not going to use those anymore. This is what your JavaScript should look like now:

```
for(var i = 0; i < works.length; ++i ) {
  $("#work").append("\
    <div class='col-sm-6 col-md-3'>\
      <img class='img-responsive' src='img/kitty.jpg'>\
    </div>\
  ");
};
```

And this is essentially what should be left of your work section in the index.html:

```
<div class="container">
  <div class="row" id="work">
  </div><!-- / row -->
</div><!-- / container -->
```

Now, we are going to purposely interrupt the string to add the content of a variable to it. If you were to look at your website now you will see, that four times the same image has been added. Let's add the images from our works array instead. Between the quotes of the `src` attribute remove the value. Instead, we are going to add the elements of the works array one by one:

```
$("#work").append("\
  <div class='col-sm'>\
    <img class='img-responsive' src='" + works[i] + "'>\
  </div>\
");
```

The `+` adds the content of the variable to the string. The second `+` adds the last part of the string. `works[i]` contains one of the elements of the works array. `i` was defined in the loop. During the first iteration, it will equal to 0. In the second iteration, it will be 1 and so on. Thus, in each iteration the next element of the array will be displayed.

Well done! Now, if you reload the page you should see the four images from your works array on your website. Whenever you have new work samples you can now simply add them to the array or replace old ones.

Practice and Review

Let's practice what we have learned about jQuery selectors, conditionals, arrays and loops one more time.

We want to use two different border colors for our images while every other one has a different color.




In your web browser, open the console in the Developer Tools. Now type in a jQuery selector to select all `img` tags inside `#work` (or whatever ID you used on the div with the class `row`).

> Remember, jQuery selectors are in many ways exactly like CSS selectors. You write them in `$("here")`. And to select elements inside other elements you just put a space in between.
>
> `$("#parent .child")` - In this example we select all elements with the class `child` that are inside an element with the ID `parent`.

When you press enter you will see all the all the image elements we added earlier. Notice, how all the image elements are stored in an array? (You recognize an array by the square brackets `[ ]`.) This means we can address each individual element the same way we did earlier to select elements of an array.

Let's use this in our code.
Inside the loop we just wrote, below the `append();` define a variable called "images". Set it equal to the jQuery selector we just used to select all the image elements. Now `images` should be an array of those image elements. If you want to try it, put `images` inside a console.log. Try `images[0]` or `images[2]` as well. Look at the console to see what happens.

Great! Now we add the colored borders.

Below, use a jQuery selector to select one element from the `images` array. Inside the square brackets use `i`. This way you select one image at a time during each iteration (like we did with the works array above.)
On the selected image element use the jQuery function `.css()` to add some CSS to it. Set the border to `2px solid salmon`.

```
$(images[i]).css("border", "2px solid salmon");
```

This works, but it's boring. Every other one should have border color of "DodgerBlue". To do this, we will add a conditional to our for loop to give a different border color to the even elements — the ones with index 0, 2, 4, and so on - and leave the odd ones alone.

So instead of the line above we can use a conditional inside our loop and then put the code blocks changing the background-color inside that (don't forget to indent at each level of nesting!), like so:

```
if (true) {
  $(images[i]).css("border", "2px solid DodgerBlue");
} else {
  $(images[i]).css("border", "2px solid salmon");
};
```

But what should go in place of the word "true"? How do we check for an even index? Meet the **modulus** operator. The modulus is a math operator that gives us the remainder when two integers are divided.

```
20 % 3 === 2; // 20 divided by 3 is 6, with a remainder of 2.
10 % 2 === 0; // 10 divided by 2 is 5, with a remainder of 0.
```

If the index `%` 2 `===` 0, the index is even and we want that border color to be "DodgerBlue". The border color of all other elements should be "salmon". In our case, the counter variable i is always the index. So `(i % 2 === 0)` is the true statement we are looking for:

```
if(i%2 === 0){
  $(images[i]).css("border", "2px solid DodgerBlue");
} else {
  $(images[i]).css("border", "2px solid salmon");
};
```

Well done!

This is what the JavaScript part of this Exercise should look like:

```
// work section
for(var i = 0; i < works.length; ++i ) {
  $("#work").append("\
    <div class='col-sm'>\
      <img class='img-responsive' src='" + works[i] + "'>\
    </div>\
  ");
```

```
// work section
for(var i = 0; i < works.length; ++i ) {
  $("#work").append("\
    <div class='col-sm'>\
      <img class='img-responsive' src='" + works[i] + "'>\
    </div>\
  ");
  var images = $("#work img");
  if(i%2 === 0){
    $(images[i]).css("border", "2px solid DodgerBlue");
  } else {
    $(images[i]).css("border", "2px solid salmon");
  };
};
```

Summary

In this Exercise, you expanded upon your JavaScript knowledge by looking at arrays and more. But we are not done with JavaScript just yet! Stay tuned for object literals, in the next Exercise.

Learn More

For loops aren't the only way to iterate through an array in JavaScript. While loops, for instance, operate on the same principle: if a particular condition remains true, keep looping. If you feel confident, check out these resources and try rewriting the code in the exercise with another type of loop.

Task

DIRECTIONS SUBMISSION HISTORY

Estimated Task Time: 1 Hour.

For this Exercise, we ask that you submit the following: