***https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference***

Construct JavaScript Objects with Functions

We are also able to create objects using constructor functions.

A constructor function is given a capitalized name to make it clear that it is a constructor.

Here's an example of a constructor function:

var Car = function() {  
  this.wheels = 4;  
  this.engines = 1;  
  this.seats = 5;  
};

In a constructor the this variable refers to the new object being created by the constructor. So when we write,

  this.wheels = 4;

inside of the constructor we are giving the new object it creates a property called wheels with a value of 4.

You can think of a constructor as a description for the object it will create.

Have your MotorBike constructor describe an object with wheels, engines and seats properties and set them to numbers.

Make Instances of Objects with a Constructor Function

Now let's put that great constructor function we made in the last lesson to use!

To use a constructor function we call it with the new keyword in front of it like:

var myCar = new Car();

myCar is now an instance of the Car constructor that looks like the object it described:

{  
  wheels: 4,  
  engines: 1,  
  seats: 5  
}

Note that it is important to use the new keyword when calling a constructor. This is how Javascript knows to create a new object and that all the references to this inside the constructor should be referring to this new object.

Now, once the myCar instance is created it can be used like any other object and can have its properties accessed and modified the same way you would usually. For example:

myCar.turboType = "twin";

Our myCar variable now has a property turboTypewith a value of "twin".

In the editor, use the Car constructor to create a new instance and assign it to myCar.

Then give myCar a nickname property with a string value.

Make Unique Objects by Passing Parameters to our Constructor

The constructor we have is great, but what if we don't always want to create the same object?

To solve this we can add parameters to our constructor. We do this like the following example:

var Car = function(wheels, seats, engines) {  
  this.wheels = wheels;  
  this.seats = seats;  
  this.engines = engines;  
};

Now we can pass in arguments when we call our constructor.

var myCar = new Car(6, 3, 1);

This code will create an object that uses the arguments we passed in and looks like:

{  
  wheels: 6,  
  seats: 3,  
  engines: 1  
}

Now give it a try yourself! Alter the Carconstructor to use parameters to assign values to the wheels, seats, and engines properties.

Then call your new constructor with three number arguments and assign it to myCar to see it in action.

Make Object Properties Private

Objects have their own attributes, called properties, and their own functions, called methods.

In the [**previous challenges**](https://www.freecodecamp.org/challenges/make-instances-of-objects-with-a-constructor-function), we used the thiskeyword to reference public properties of the current object.

We can also create private properties and private methods, which aren't accessible from outside the object.

To do this, we create the variable inside the constructor using the var keyword we're familiar with, instead of creating it as a property of this.

This is useful for when we need to store information about an object but we want to control how it is used by outside code.

For example, what if we want to store the speed our car is traveling at but we only want outside code to be able to modify it by accelerating or decelerating, so the speed changes in a controlled way?

In the editor you can see an example of a Carconstructor that implements this pattern.

Now try it yourself! Modify the Bike constructorto have a private property called gear and two public methods called getGear and setGear to get and set that value.

[**Further explanation on this keyword**](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/this)

Iterate over Arrays with map

The map method is a convenient way to iterate through arrays. Here's an example usage:

var oldArray = [1, 2, 3];  
var timesFour = oldArray.map(function(val){  
  return val \* 4;  
});  
console.log(timesFour); // returns [4, 8, 12]  
console.log(oldArray); // returns [1, 2, 3]

The map method will iterate through every element of the array, creating a new array with values that have been modified by the callback function, and return it. Note that it does not modify the original array.

In our example the callback only uses the value of the array element (the val argument) but your callback can also include arguments for the index and arraybeing acted on.

Use the map function to add 3 to every value in the variable oldArray, and save the results into variable newArray. oldArray should not change.

Condense arrays with reduce

The array method reduce is used to iterate through an array and condense it into one value.

To use reduce you pass in a callback whose arguments are an accumulator (in this case, previousVal) and the current value (currentVal).

The accumulator is like a total that reduce keeps track of after each operation. The current value is just the next element in the array you're iterating through.

reduce has an optional second argument which can be used to set the initial value of the accumulator. If no initial value is specified it will be the first array element and currentVal will start with the second array element.

Here is an example of reduce being used to subtract all the values of an array:

var singleVal = array.reduce(function(previousVal, currentVal) {  
  return previousVal - currentVal;  
}, 0);

Use the reduce method to sum all the values in array and assign it to singleVal.

Filter Arrays with filter

The filter method is used to iterate through an array and filter out elements where a given condition is not true.

filter is passed a callback function which takes the current value (we've called that val) as an argument.

Any array element for which the callback returns true will be kept and elements that return false will be filtered out.

The following code is an example of using filter to remove array elements that are equal to five:

Note: We omit the second and third arguments since we only need the value

array = array.filter(function(val) {  
  return val !== 5;  
});

Use filter to create a new array with all the values from oldArray which are less than 6. The oldArrayshould not change.

Sort Arrays with sort

You can use the method sort to easily sort the values in an array alphabetically or numerically.

Unlike the previous array methods we have been looking at, sort actually alters the array in place. However, it also returns this sorted array.

sort can be passed a compare function as a callback. The compare function should return a negative number if a should be before b, a positive number if a should be after b, or 0 if they are equal.

If no compare (callback) function is passed in, it will convert the values to strings and sort alphabetically.

Here is an example of using sort with a compare function that will sort the elements from smallest to largest number:

var array = [1, 12, 21, 2];  
array.sort(function(a, b) {  
  return a - b;  
});

Use sort to sort array from largest to smallest.

Further explanation on the .sort() method can be found [**here**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/sort).

Reverse Arrays with reverse

You can use the reverse method to reverse the elements of an array.

reverse is another array method that alters the array in place, but it also returns the reversed array.

var myArray = [1, 2, 3];  
myArray.reverse();

*returns [3, 2, 1]*

Use reverse to reverse the array variable and assign it to newArray.

Concatenate Arrays with concat

concat can be used to merge the contents of two arrays into one.

concat takes an array as an argument and returns a new array with the elements of this array concatenated onto the end.

Here is an example of concat being used to concatenate otherArray onto the end of oldArray:

newArray = oldArray.concat(otherArray);

Use .concat() to concatenate concatMe onto the end of oldArray and assign it to newArray.

Split Strings with split

You can use the split method to split a string into an array.

split uses the argument you pass in as a delimiter to determine which points the string should be split at.

Here is an example of split being used to split a string at every s character:

var array = string.split('s');

Use split to create an array of words from stringand assign it to array.

Join Strings with join

We can use the join method to join each element of an array into a string separated by whatever delimiter you provide as an argument.

The following is an example of using join to join all of the elements of an array into a string with all the elements separated by word and:

var veggies = ["Celery", "Radish", "Carrot", "Potato"];  
var salad = veggies.join(" and ");  
console.log(salad); // "Celery and Radish and Carrot and Potato"

Use the join method to create a string from joinMe with spaces in between each element and assign it to joinedString.