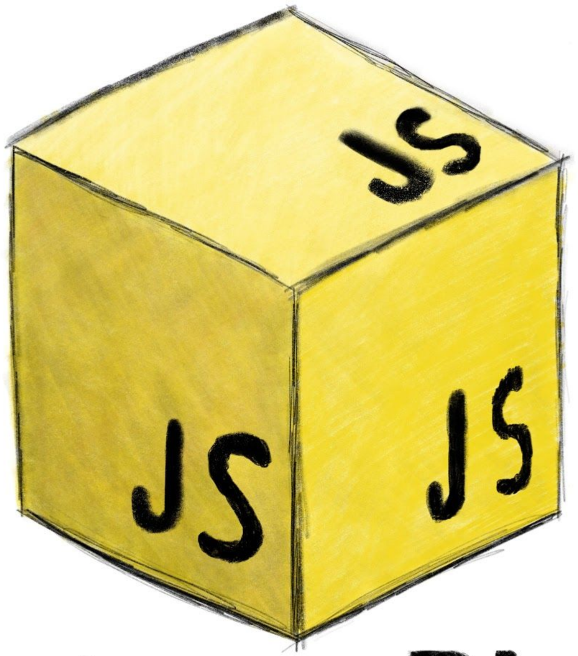
Javascript Beginner Blocks



JavaScript Beginner Blocks

Version 1.0

Written by Jared York

INTRODUCTION

JavaScript is commonly used for adding interactive features to websites. Many games, menus, button validation responses, animations, and themes use JavaScript. This free course aims to cover the basics to the language. No prior programming knowledge is required.

HELLO WORLD

Take a moment and open a text editor. Anything will do, even Notepad or Text Edit. There are other more dedicated text editors that are more well suited for programming. Create a new file, name it anything you wish, and save it with a .html extension. For example, I call mine index.html. Double-click on our new HTML file to open it in your browser. Personally, I find it easier to have my text editor on one side of the screen, and have the browser on the other side. At this point you should have your new HTML file opened in your text editor.

Add the following inside our new HTML file to declare that our file is an HTML file, and also declare a script.

```
<!DOCTYPE html>
<html>
	<head></head>
	<body>
		<script>

		</script>
	</body>
</html>
```

The most important thing to learn first would be how to display a line of text in the browser.

To accomplish this, we will be using a JavaScript alert. An alert is a small pop-up box that is displayed and originates from a web page.

In your text editor, between the opening <script> tag, and the closing </script> tag, type:

```
alert(“Hello world!”);
```

If we refresh the browser, we should see a pop-up that appears with our text, “Hello world!” If you see this, feel free to remove that line from our file.

FUNCTIONS

In JavaScript, alert is considered a function. A function is a block of code that can be reused elsewhere in the code by calling it. Calling a function simply means running the code within the function from a reference elsewhere in the code. We were also able to pass a value to the alert function, in this case it expected what is called a “string”. A string is simply a string of characters. A string can be as little as a word, or even as big as a novel. Strings are not restricted to containing only letters. Numbers and other characters can also be in a string. A sequence of characters are a string when there are quotation marks (single or double) around them. After our function call, we then add a semicolon, “;”, which is not required, but good practice. Many other languages interpret semicolons as the end of a line or statement.

Now, let’s try creating our very own function. To create a function, we first have to type the keyword “function”. After that, add a space, then we can add the name of our function. The only names we cannot define a function by are the keywords and functions that are already defined by

JavaScript and/or the browser. We can name a function almost anything. So far our function declaration should look like:

```
function sayHello
```

In the above example, “sayHello” can be almost anything you would like, as I mentioned. The next part of our function declaration is adding a beginning parenthesis, “(“ directly after the function name. Just for now, we won’t be adding anything after the opening parenthesis, so we will just add a closing one: “)”. To finish our function declaration we can add another space, and type an opening curly brace: “{“. All of the code that we want executed will go between an opening curly brace and an ending curly brace. Feel free to press enter or return on your keyboard a few times to open up some empty lines. After adding some empty lines, we can add our ending curly brace: “}”. Our new function should look similar to the following:

```
function sayHello() {



}
```

I previously mentioned that when a function is called, it runs the code within the curly braces. To test this, we can add a call to the alert function within the curly braces. If we do, your code should look similar to:

```
function sayHello() {
  alert(“Hello world!”);
}
```

If we try refreshing the page in our browser, you will notice that nothing happens. This is because we have to call our function. Under our function, after the closing curly brace, add the function call:

```
sayHello();
```

Now if we refresh the web page, you should see the alert pop up we've seen previously. If so, great work! We can then move on. If you still see no alert prompt, ensure you declared the function as described above, and call the function after the ending curly brace of the function.

The next concept that is critical to learn in JavaScript is that of variables. Variables are the fundamental building blocks of JavaScript. You can think of variables kind of like a basket. You can store values in them. The values assigned to a variable can be anything including, but definitely not all: strings, integers, decimals, functions, etc.

Before our function call to sayHello, we can declare a variable that stores a string of who we want to sayHello to, well, say hello to. To declare a variable, on the line above our call to sayHello, type "var". Much like typing "function" to declare a function, adding "var" will declare a variable. Add a space, then type the name you wish for the variable to be called. Like functions, the name can be almost anything, though there are some variable names that are reserved by the JavaScript engine and/or the browser. To "store" a value in a variable, we have to assign the value to the variable. If I want to name the variable, "name", I can then assign my name to the variable with the following code:

```
var name = “Jared”;
```

Now try giving it a try with your own name! We will be working with this "name" variable for the next little bit.

We can also add a string to an existing string. If we declare:

```
var name = “Jared”;    // replace “Jared” with your own first name
```

Perhaps we wanted to add our last name to the value of “name”. To do so, we can use the string operator: “+=”. Add the following right under where we declare the “name” variable:

```
name += “York”;    // replace “York” with your own last name.
```

Then, we can insert the name variable between the two parenthesis of our sayHello function call. The sayHello function call should appear as follows:

```
sayHello(name);
```

The goal will be to replace the alert text of “Hello world!” with “Hello <your name>!” To accomplish this we will have to use a term called string concatenation. String concatenation essentially means combining strings together.

Inside our sayHello function we want to add the name variable to our function declaration. Next we will remove the string inside the alert call. Then, inside the parentheses of the alert call we want to add:

```
“Hello ” + name + “!”
```

This is an example of string concatenation. Effectively we are combining “Hello ”, the value of our name variable, and “!”. If I set the name variable to my name, the output would be:

```
“Hello Jared!”
```

We can now refresh the browser and we should see an alert displaying:

"Hello <your name>!"

Pretty neat, right?

Whenever we provide a value in a function call, it is known as a parameter. Parameters can be given to function calls, and when a function is defined, can be accessed as variables within the parenthesis.

Another example of parameters would be if we created a function called "cookieStats". Perhaps we want to display how many chocolate chips are in a cookie when we provide a number into the cookieStats function:

```
function cookieStats(amountChocolateChips) {  // here is the function definition
  alert("This cookie has " + amountChocolateChips + " chocolate chips. Yum!");
}

cookieStats(8);  // here we call the function
```

If we break down how the code is interpreted when the page is refreshed, this is what would happen.

cookieStats(**8**); // we set the value of the first parameter (amountChoclateChips) to 8

function cookieStats(amountChocolateChips (**is now equal to 8**)) {
  alert("This cookie has " + **8** + " chocolate chips. Yum!");
}
Output: alert box showing "This cookie has 8 chocolate chips. Yum!"

We can also add multiple parameters to a function as well. Let's try this out by creating a new function named "add", but also add two parameters:

```
function add(firstNumber, secondNumber) {

}
```

The goal of this function is to calculate the sum of two numbers: "firstNumber", and "secondNumber", then add them together. To accomplish this, we can add the following code inside the function:

```
var sum = firstNumber + secondNumber;
```

The above line will add the first parameter, "firstNumber", to the second parameter, "secondNumber", then store the answer into the variable, "sum".

Now that we can store the sum into the "sum" variable, we can actually access the sum of any two numbers *outside* the function. One way we can access the sum outside the function, is if we "return" the value. Inside the function, on the next line after the previous line we added, we can insert the following:

```
return sum;
```

That concludes our "add" function! We can now call our function, and store the result (sum) into a variable. Under our function declaration for "add", let's call our function and store it to a variable.

```
var answer = add(2, 6);
```

We can then show an alert prompt with the value of the answer variable:

```
alert(“The answer is: ” + answer + “!”);
```

If we then refresh the page in the browser, we should see an alert box that displays:

```
“The answer is: 8!”
```

Fantastic! We now know how to write functions, call functions, and utilize return values. Let’s move on to arrays. An array can almost be thought of as a shopping cart (or trolley for all you British English people ;)). You can store multiple items in a shopping cart. Arrays act similarly, they can store multiple objects. First, clear out all the code between the <script> tags of our HTML. Then add the following code to declare an array.

```
var ingredients = [];
```

The preferred method to declare an array is use square brackets: “[]”. We can even add elements to our array when we declare it, for example:

```
var ingredients = [“eggs”, “flour”, “butter”, “milk”, “salt”, “vanilla”];
```

Arrays elements can be any type. There are six types in JavaScript: string, number, null, undefined, boolean, and symbol. All elements of an array do not need to be of the same type either.

We can also dynamically add elements to the array by using the push method of an array. By the way, a method is just another name for a function, but it belongs to an object. Really, it’s just semantics. To try out the push method, let's declare an array and utilize the push method as follows:

```
var numbers = [];

numbers.push(1);
numbers.push(2);
numbers.push(3);
numbers.push(4);
numbers.push(5);
```

If we were to call the "alert" function and provide numbers as the parameter like:
alert(numbers);

We should see the following displayed in our alert prompt when we refresh the page:
"1,2,3,4,5"

Now that we learned how to add elements to an array, we can now cover how to refer to a specific element in an array.

If we go with the above example, say we want to get the 3rd element in the "numbers" array, we can refer to the third element like so:

```
numbers[2]
```

Keep in mind, JavaScript arrays start at index 0. This means, if we want to refer the the first element, we refer to index 0, if we want the second, we refer to index 1. If we want to the third element, we want to refer to index 2. Essentially, whatever element you want, you subtract one from it and that will be the index you want to use to refer to the element you want.

Now let's try replacing the parameter in our alert call with numbers[2], to refer to the third element in the array. Then let's refresh the browser. You should now see the following:

```
"3"
```

Now we should see "3" printed in our alert prompt, since the third element of the "numbers" array is "3". Great! Now let's cover how to removing an element from the array, as well as removing all elements from the array.

We can remove an individual element from the array by using the "splice" method. The first parameter the splice method requires is the index you want to start removing at. The second parameter is the amount of elements you want to remove after the index specified with the first parameter. The splice method returns an array of the elements that have been removed. In most cases you probably won't need this returned array, so don't worry about assigning the call to a variable.

Using our numbers array example again, just before the alert call we can add:

```
numbers.splice(2, 1);
```

The above code removes the third element from the numbers array. This is due to specifying the second index as the place we want to start removing, and adding "1" as the second parameter which only removes a single element.

Let's remove the square brackets from the numbers variable in the alert call, and refresh the page. We should see:

```
"1,2,4,5"
```

Perhaps you might want to find the amount of elements in an array, to do so you can refer to the length property of the array. Let's try it by changing the first parameter of the alert call from:

```
alert(numbers);
```

To:

```
alert(numbers.length);
```

If we refresh the page we should see that "4" is printed.

The last important part of arrays to learn, is how to remove all elements of an array. Removing all elements of an array is simple, just set the length of the array to zero:

```
numbers.length = 0;
```

OBJECTS

Fantastic! We can now move on to objects. An object is a group of key-value pairs, without any specific ordering. Objects are similar to arrays, but you assign a key to each "element". You can then refer to a value with it's key. It will make more sense once we cover creating an object. Let's erase everything between the script tags again. To create an object, type the following:

```
var pets = { };
```

Between the curly braces, we can then add our properties. Let's add two:

```
var pets = {
	cats: 3,
	dogs: 2
};
```

In the above example, if we look at the first property, the key is “cats”, and the value is “3”. If we look at the second property, the key is “dogs”, and the value is “2”.

If we want to get the value of the “dogs” property, add the following call to the “alert” function:

```
alert(pets.dogs);
```

Refreshing the page, we should see:

```
“2”
```

We can also get the value of an object’s property by using square brackets, similar to an array. If we wanted to get the value of the “dogs” property, we can change pets.dogs to pets[“dogs”]. When we refresh the page we should still see “2”. If we want to change the value of an object property of, say, “dogs”, we can add the following before the alert call:

```
pets.dogs += 4;
```

The above code will add 4 to the existing value of “dogs”. If we refresh the web page, we should see:

```
“6”
```

Before we proceed further with objects, it's important to go over how the different operations that can be made in JavaScript:

+ means addition

- Means subtraction

* Means multiplication

/ means division

% means modulus (remainder from division)

++ means increment

-- means decrement

There are also various assignment operators that are in JavaScript. Let's say x = 2, y = 5;

The "=" operator assigns x to y as in x = y. The result would be x = 5.
The "+=" operator adds y to the existing value of x. The result would be x = 7.
The "-=" operator subtracts y from the existing value of x. The result would be x = -3.
The "*=" operator multiplies y and the existing value of x. The result would be x = 10.
The "/=" operator divides x by y. The result would be x = 0.4.
The "%=" operator takes the modulus of x and y. The result would be x = 2.

That’s enough of operators for now. The last important part of objects to keep in mind is creating new properties. To create a new property, we can simply just refer to a key and assign it:

```
pets.turtles = 5;
```

Or:

```
pets[“turtles”] = 5;
```

A property that belongs to an object can be removed by simply using the delete keyword:

```
delete pets.turtles;
```

Or:

```
delete pets[“turtles”];
```

STATEMENTS

At this point, we can pretty much do anything with objects now. The next important part of JavaScript to learn is if statements and switch statements.

Let’s keep the pets object we kept before. Perhaps we want to check if the property “cats” of the “pets” object is equal to “3”. The way we can write this is as follows:

```
if (pets.cats === 3) {
	alert(“There are three cats!”);
}
```

The above code checks that if the “cats” property of pets is the same type as the integer three, and is equal to three. Another way we could write the above condition is using two equal signs like the following:

```
if (pets.cats == 3) {
        alert(“There are three cats!”);
}
```

The code above behaves the same as the last code snippet, but the above code does not check that both types are the same. Usually in practical applications, you would want to check that the types are the same as well to ensure weird bugs don’t occur because of types. Keep in mind, both the object we are comparing or what we are comparing against can be any type.

What if the “cats” property is not equal to three? We can handle that with an else statement. After the following code:

```
if (pets.cats === 3) {
        alert(“There are three cats!”);
}
```

Add the following right under the above:

```
else {
        alert(“There are not three cats. :(“);
}
```

Now just above our if statement, add:

```
pets.cats++;
```

As we learned previously, using ++ will add one to the value of the property/variable. Incrementing the “cats” property will allow the if statement to evaluate to false, therefore moving to the code in our else statement.

If we refresh the browser, we should see:

“There are not three cats. :(“

This is great, but what if we want to check if there are three cats, if there are not, if there are five cats?

Change “pets.cats++” to “pets.cats += 2”. We will now have five cats to test the “else if” statement we will go over.

Between our initial if statement and else statement, add:

```
else if (pets.cats === 5) {
        alert(“There are five cats!”);
}
```

Then, change the parameter of the alert call in the normal else statement to:

“There are not three cats or five cats. :(“

The full section of code should appear as follows:

```
if (pets.cats === 3) {
        alert(“There are three cats!”);
}
```

```
else if (pets.cats === 5) {
        alert("There are five cats!");
}
else {
        alert("There are not three cats or five cats. :(");
}
```

It's important to point out that you can add as many else if statements as you like, just add them after the initial if statement. The else statement is always the last condition in the chain.

At this point, we can remove all the code between the HTML <script> tags. There are a number of common comparison operators that are important to understand. While we do not need to go over all of them here. Here is a list of comparison operators you can utilize:

Equals (==) - If operands are equal, return true.
Not equal (!=) - If operands are not equal, return true.
Strict equal (===) - If operands are equal and are of the same type, return true.
Strict not equal (!==) If operands are of the same type but not equal, or are a different type, return true.
Greater than (>) - If the left operand is greater than the right operand, return true.
Greater than or equal (>=) - If the left operand is greater than or equal to the right operand, return true.
Less than (<) - If the left operand is less than the right operand, return true.
Less than or equal (<=) - If the left operand is less than or equal to the right operand, return true.

Before moving on to switch statements, there is a really neat little "compact if statement", as I call it, called a ternary operator. Ternary operators return a value based on a condition, which can usually be fit on a single line.

Here is an example of a ternary operator:

```
var songs = 9;

var areThereMoreThanFiveSongs = songs > 5 ? true : false;
```

If you excuse me for the incredibly long and descriptive (and totally original) variable name, we can break down this ternary operator in to pieces.

The first piece is “songs > 5”, this is the condition we are checking for. The next part, “? true”, means that if the condition is true, return the value after the “?” mark. The final part, “: false”, declares the value that is returned if the condition is false. In the above example, the result of the ternary operator is assigned to the variable, “areThereMoreThanFiveSongs”.

The next important statement is called a switch statement. Switch statements are useful in cases where you would chain multiple else if statements with each checking if a variable is equal to a particular value. Take the following if statement with a chain of else if statements:

```
if (time === “12pm”) {
        alert(“It is 12 o’clock”);
}
else if (time === “1pm”) {
        alert(“It is 1 o’clock”);
}
else if (time === “2pm”) {
        alert(“It is 2 o’clock”);
}
else if (time === “3pm”) {
        alert(“It is 3 o’clock”):
}
else if (time === “4pm”) {
        alert(“It is 4 o’clock”);
}
else if (time === “5pm”) {
        alert(“It is 5 o’clock”);
}
else {
        alert(“The time is none of the above.”);
}
```

The equivalent of a above code block as a switch statement equates to the following:

```
switch (time) {
        case “12pm”: {
                alert(“It is 12 o’clock”);
                break;
        }

        case “1pm”: {
                alert(“It is 1 o’clock”);
                break;
        }

        case “2pm”: {
                alert(“It is 2 o’clock”);
                break;
        }
        case “3pm”: {
                alert(“It is 3 o’clock”);
                break;
        }

        case “4pm”: {
                alert(“It is 4 o’clock”);
                break;
        }

        case “5pm”: {
                alert(“It is 5 o’clock”);
                break;
        }

        default: {
                alert(“The time is none of the above.”);
                break;
        }
}
```

LOOPS

Now that we know how if statements and switch statements work, we can move along to loops! Loops provide a relatively painless way of being able to repeat a block of code. There are a large variety of loops that can be made with JavaScript, but they all behave essentially identical. The only main difference between the different types of loops are the different ways start and end points are specified. There are some situations where one loop works better than the others, etc.

There are two loops that are very common in many JavaScript applications: for loops, and while loops. We will cover both of these loops below.

For loops repeat a block of code using an index.

For example, if we have an array that holds cookies, we can continuously add cookies to the array using a for loop like so:

```
var cookies = [];

for (var i = 0; i < 24; i++) {
	cookies.push(“a cookie.  This is cookie: ” + i);
}
```

The first part of the above for loop, “var i = 0;”, initializes a variable called “i”. The second part of the for loop defines the end condition, in other words, repeat the following code until when. The third and final part of the for loop increments i, usually by one, however “i” can be manipulated in any way.

While loops are another common loop JavaScript provides, which acts similarly to a for loop. While a condition is true, the code in a while loop is executed. Take the following example:

```
var maxCupsCoffee = 10; // set the maximum number of cups of coffee that can be drank to 10.

var cupsDrank = 0;  // start with 0 cups of coffee

white (cupsDrank < maxCupsCoffee) {
	cupsDrank++;
```

```
    alert(“You can drink “ + (maxCupsCoffee - cupsDrank) + “ more cup(s) of coffee.”);
}
```

If you added the above code between the <script> tags in your HTML file, and refreshed the page, you should see a continuous stream of alert prompts one after another:

```
“You can drink 9 more cup(s) of coffee.”
“You can drink 8 more cup(s) of coffee.”
“You can drink 7 more cup(s) of coffee.”
“You can drink 6 more cup(s) of coffee.”
“You can drink 5 more cup(s) of coffee.”
“You can drink 4 more cup(s) of coffee.”
“You can drink 3 more cup(s) of coffee.”
“You can drink 2 more cup(s) of coffee.”
“You can drink 1 more cup(s) of coffee.”
“You can drink 0 more cup(s) of coffee.”
```

You may have found that alerts can be a bit annoying, especially when many are used at once. Don’t worry, it’s much more common to debug your code using the “console.log()” method. Let’s try it!

Change:

```
alert(“You can drink “ + (maxCupsCoffee - cupsDrank) + “ more cup(s) of coffee.”);
```

To:

```
console.log(“You can drink “ + (maxCupsCoffee - cupsDrank) + “ more cup(s) of coffee.”);
```

If you refresh the page, you may not notice anything happens. Press the “F12” key, and you should see the development tools pane open in your browser. By default the “Console” tab should be selected, if not, select the “Console” tab. Both of these methods of opening the dev tools and the console should be mostly browser agnostic. Try refreshing the page with the console open. You should see a line by line printout of the output we’ve seen before:

```
“You can drink 9 more cup(s) of coffee.”
“You can drink 8 more cup(s) of coffee.”
“You can drink 7 more cup(s) of coffee.”
“You can drink 6 more cup(s) of coffee.”
“You can drink 5 more cup(s) of coffee.”
“You can drink 4 more cup(s) of coffee.”
“You can drink 3 more cup(s) of coffee.”
“You can drink 2 more cup(s) of coffee.”
“You can drink 1 more cup(s) of coffee.”
“You can drink 0 more cup(s) of coffee.”
```

CONCLUDING THOUGHTS

There we have it! These are the fundamentals (and some more intermediate and advanced concepts sprinkled in) of JavaScript! I hope you found this course useful! Please send me your feedback, comments, and suggestions to jared.york@jaredyork.com. I would be more than happy to help if you have any questions.