JAVASCRIPT

Introduction to JavaScript

Up until now we've been making websites using **HTML**and **CSS**and while we can make websites that look impressive and contain all of the information we want them to say they will need something more to be truly interactive and to be functional. This is what JavaScript was purpose built to do... it allows us to write the logic and to add in the functionality that make our websites interactive for our users. JavaScript is very good at selecting the parts of the website we want to modify and attach events that listen for user input and manipulate the **DOM**(Document Object Model) of the website in a dynamic and efficient way.

Before we bring **JS**into the mix with **HTML**and **CSS**we need to establish some basics of what we can do with it. The next section is going to get us up to speed with some general programming concepts and how they work in **JS**. We'll do our best to ground our examples in real life problems but some sections will have to be heavy on theory so don't be surprised if you have to go back and re-reference the sections later. There is a lot of new syntax, the way we express the desired operation in code, that we need to be acquainted with so do expect to have to reference the exact way to write things again and again. This isn't like learning a foreign language, we will always have access to references and the internet to when expressing ourselves in code and even seasoned programmers have to look things up all the time. And where we can we'll leave out the little details that aren't important for our general understanding now, but know there is always another layer another wrinkle, some exceptions that tend to come up and they'll be explored when the time is right.

# Variables, Types, Operators

### **Objectives**

* Learn how to create and use variables
* Learn about basic data types in JavaScript
* Learn about operators in JavaScript

## **Variables**

In JavaScript we can define variables, basically a name we can use to store and modify some information. Some examples of variables look like

*var* count = 5;

*var* message = "Hello World";

*var* likesJS = true;

copy

When declaring a variable we need to start by using the var keyword followed by giving it some sort of unique name (the more descriptive the better). In the examples above the variables are count, message, and likesJS. We have also set each of them equal to some values as well with 5, "Hello World", and true respectively. A couple things to know about this:

1. We can set variables equal to different types of data (in the above example we have a Number, a String, and a Boolean)
2. We don't have to give it a value (it can be left undefined and/or we can choose to define it later but reserve the name and space ahead of time)
3. The values are allowed to be reassigned and changed

While there are different types of data in JavaScript variables are \*loosely\* typed meaning we can change their type at will.

*var* count = 5;

count = "I'm now a string!";copy

In the above example we change a Number into a String. While this isn't something we'll typically want to do in our code we should be aware that we can change it.

## **Operators**

There are many different operators in JS, and we should already be familiar with some of them from our knowledge of math. For instance we have already used = to set our variables equal to some values and also in the second example to change what a variable is equal to. Unlike in math, the = operator isn't making sure each side of the statement is balanced, but rather in programming it is used to set the left side of the statement equal to whatever is on the right side of the statement. So if we were to take an example like var count = 5; and translate it into exactly what our JavaScript engine is being told it would be:

1. Create a variable if it doesn't already exist called count
2. Set the value of the count variable to 5

A line like count = "I'm now a String!"; would be understood as:

1. Set the value of an already existing variable called count to the string "I'm now a String!"

Naturally, var count = 5; and count = "I'm now a String!"; are a lot shorter for us to write. Much of programming syntax is about expressing a specific operation in a consistent and convenient way. We don't want our JS engine to have to guess what we want it to do or make assumptions about what we meant.

Another operator we need to talk about in some detail is +

When we are working with Number type variables + will work as we would expect adding the two values together

*var* product = 3.14159 + 2.71828; // 5.85987copy

**Note:** JS doesn't differentiate between whole integers and numbers with decimals

But what about if we are working with String values?

*var* name = "Marisa";

*var* message = "JS is actually the trickiest language";

console.log(name + " says " + message);

// "Marisa says JS is actually the trickiest language"copy

When working with strings + is used to concatenate them together, think of concatenation as gluing the strings together to make a new one. In the above example, console.log() is a command that will tell the program to print out whatever the value in between the parenthesis evaluates to. We'll be using console.log() quite a bit to get a peak at what the computer is doing. It's very common to use console.log() when checking our work or when debugging (trying to figure out why our code doesn't behave as we'd expect).

What if we mix the types we're using?

*var* name = "Phil";

*var* luckyNumber = 8;

console.log(name + "'s lucky number is " + luckyNumber);

// "Phil's lucky number is 8"copy

When we add numbers and strings together they result in a new string. One more tricky example...

console.log("the number is " + 12 + 3);         // "the number is 123"

console.log(12 + 3 + " is a different number"); // "15 is a different number"copy

If we do mix the types remember that order matters and that JS will read from left to right.

### **Some operators**

|  |  |  |
| --- | --- | --- |
| Operator | What it does | Example Operation(s) |
| a + b | add two numbers together, concatenates strings | 12 + 3 // 15  "12" + "3" // "123"copy |
| a - b | subtracts b from a | 12 - 3 // 9copy |
| a \* b | multiplies a by b | 12 \* 3 // 36copy |
| a / b | divides a by b | 12 / 3 // 4  12 / 5 // 2.4copy |

### **Order of operations**

What will happen if the code contains both multiplication and addition?

*var* answer = 2 + 4 \* 10;copy

What does this give us? Click for the answer / explanation

## **Combined Operations**

Sometimes we will combine operations together using convenient shorthand.

*var* total = 10;

total += 5; // equivalent to total = total + 5;copy

The += operator can also be used with strings for concatenation

*var* message = "hello";

message += " world"; // message is now "hello world"copy

Similar to the above examples, we can also use the following: -=, \*=, and /= as combined math and assignment operations.

## **Types of Data**

Some types of data are considered [primitive](https://developer.mozilla.org/en-US/docs/Glossary/Primitive) they essentially just store information whereas there are other types that can provide structure as well. For now we should be aware of Number, String, and Boolean. We will soon learn about structural types as well with Object and Array (which **JS** also considers as Object) types.

Graphical user interface, text, application, Teams

Description automatically generatedGraphical user interface, text, application

Description automatically generated

# Functions

Up to this point, our code is executed from top to bottom, and everything is executed immediately. With functions, we are able to write a block of code that will only be executed when we call on it. This also means we are able to call on it as many times as we need. Let's look at the syntax of a function:

*function* name\_of\_function() {

    // code to be executed

}copy

So if we now move our original counter code into our function, we have now given this set of steps a name and can recall it whenever we need with rewriting the whole algorithm.

*function* counter() {

    for(*var* num = 0; num <= 5; num++){

        console.log(num);

    }

}copy

However, as it stands, our counter code **is not being run**. That is because we have yet to call it.  Think of functions like a recipe.  Once we have the recipe for chocolate chip cookies we now know how to make cookies, but that doesn't mean we immediately have cookies once we get the recipe.  We have to actually read through the steps and make the cookies for that.

## **Calling or Invoking a Function**

When the computer is reading the code and gets to a function, it does not execute it immediately. If we want the code to be executed, we call it by its name and add ().

name\_of\_function();copy

So now we have defined our counter function and are calling it so our computer know to run that code within the function.

*function* counter() {

    for (*var* num = 0; num <= 5; num++) {

        console.log(num);

    }

}

counter(); // run the function

counter(); // run the function againcopy

Now that we've written the function, we can call on it as many times as we'd like! This means that we don't have to write that for loop again, but can simply call the function whenever we'd like it to run, and it will re-execute all the function's code.

## **Adding Parameters**

Functions are great, but they become even more useful when we're able to send in different values so that the same code runs, but it may vary slightly depending on our input! We indicate that a function requires input by specifying parameters in the parentheses next to the function's name. Then, when we call on the function, we pass in arguments, or actual values to be used in the function. Supposed we wanted the same counting functionality, but just wanted it to start at a different number every time:

*function* counter(startNum) { //The function is expecting some PARAMETER in order to run

    for (*var* num = startNum ; num >= 0 ; num--) {

        console.log(num);

    }

}

counter(6); // Passing in an ARGUMENT of 6, the console would display: 6, 5, 4, 3, 2, 1, 0

counter(3); // Passing in an ARGUMENT of 3, the console would display: 3, 2, 1, 0copy

With this one block of code, we can call this function with different inputs and then see different messages!

We have been using console.log a lot because it allows us as developers to see what's going on. While useful, besides printing something to the screen, it doesn't really do anything.

## **Returning vs Console logging**

Now that we know about functions, it's important to learn about the return statement. Suppose we wanted to have a function that, when given a number, created an array with values from 0 to that number:

*function* createArray(num) {

*var* newArray = [];

    for (*var* i = 0; i <= num; i++) {

        newArray.push(i);

    }

}

createArray(5);

//The newArray only exists inside of the function. Out here it no longer exists!copy

With this code, calling the function creates an array [0,1,2,3,4,5], but then, when the function ends, this array is promptly lost because we have no reference to it. Even if we print it to the console, we wouldn't be able to do anything with the array later. This is where returns are really important--the thing that we return is the true output of the function. A return statement also serves to end the function immediately.

If our function made ice cream cones, console.log would be the equivalent of the ice cream man saying, "I've got some ice cream!", where a return statement is the ice cream man actually handing you an ice cream cone!

When a function is called, the value that is returned gets sent back to the piece of code it was called from. Let's add a return statement and a variable to our code from above:

*function* createArray(num) {

*var* newArray = [];

    for (*var* i = 0; i <= num; i++) {

        newArray.push(i);

    }

    return newArray; // added the return statement

}

*var* y = createArray(5); // now y is the variable that is calling on createArraycopy

What do you think the value of y is?

If you said the array [0,1,2,3,4,5], you are correct! Remember,**the value of a function call is whatever it returns.**In this case, since y called on createArray with the value 5, the value that createArray returns (in our case, the array [0,1,2,3,4,5]) is then assigned to the variable y.

# Click Event

When a user visits our website they are presented with information in our **HTML** with a look and feel shaped by our **CSS**. In order to interact with our website, we will need to provide the user with elements they can click, elements they can scroll or swipe, and forms they can fill out. As the programmer we have no way of knowing beforehand how long it will take a user to fill out a form or exactly when a user might click a button so we need to write code that can react to the user. We can do this through events. The simplest event we can react to is the onclick event which is triggered by a user left clicking their mouse or tapping on a touchscreen over the element.

<button onclick="alert('hello')">Click Me!</button>copy

If we add an additional attribute called onclick on the element in our **HTML** we can set the onclick equal to a **JavaScript** function that will then be executed. In the example above it will trigger an alert which will display a dialog window showing whatever message we provide to it.

## **Linking to Scripts**

In the previous example we show how easily we can use a built in function like alert, what if we want to use a custom function that we have written ourselves?

We can write our own custom function in a **JavaScript** file maybe naming it something like script.js.

*function* custom() {

// we can include more code here if we want to

console.log("this message is coming from script.js");

}copy

We have used a <link> tag to include our **CSS** into our **HTML**, with **JavaScript** we need to use a <script> tag.

<script src="script.js"></script>copy

# What is "this"?

### **Objectives**

* Learn what the this keyword is used for
* Learn some basic ways we can manipulate elements

As we've seen in the previous lesson, we can use alert() to react to users clicking on elements in our page. If we want to do something more, like manipulate the element that is clicked on we can make use of this, a special parameter to pass into the custom **JavaScript** functions we can write.

<button onclick="example(this)">

Click Me

</button>copy

*function* example(element) {

console.log("element clicked", element);

}copy

When this button is clicked on it should console log something that looks like below.

The power of this is that we can then use **JavaScript** to read the content of the element or to even manipulate it if we like. There are a number of things we can do to manipulate the element like changing its style or changing its content. Text is one of the most common things we can manipulate, and we can do this by using the .innerText property.

<button onclick="turnOff(this)">On</button>

<button onclick="turnOff(this)">On</button>

<button onclick="turnOff(this)">On</button>copy

*function* turnOff(element) {

element.innerText = "Off";

}copy

Notice how in each case the button that is changed is the one that is clicked? That's the magical property of this, it let's us know exactly which element we are clicking. Another interesting thing we can do with the element is to actually remove it from the page. We can do this by using a method of the element called .remove().

<img src="ninja.png" alt="ninja" onclick="hide(this)">

<img src="ninja.png" alt="ninja" onclick="hide(this)">

<img src="ninja.png" alt="ninja" onclick="hide(this)">copy

*function* hide(element) {

element.remove();

}copy