Review

Let’s take one more glance at the concepts we just learned:

* Data is printed, or logged, to the console, a panel that displays messages, with console.log().
* We can write single-line comments with // and multi-line comments between /\* and \*/.
* There are 7 fundamental data types in JavaScript: strings, numbers, booleans, null, undefined, symbol, and object.
* Numbers are any number without quotes: 23.8879
* Strings are characters wrapped in single or double quotes: 'Sample String'
* The built-in arithmetic operators include +, -, \*, /, and %.
* Objects, including instances of data types, can have properties, stored information. The properties are denoted with a . after the name of the object, for example: 'Hello'.length.
* Objects, including instances of data types, can have methods which perform actions. Methods are called by appending the object or instance with a period, the method name, and parentheses. For example: 'hello'.toUpperCase().
* We can access properties and methods by using the ., dot operator.
* Built-in objects, including Math, are collections of methods and properties that JavaScript provides.

Review Variables

Nice work! This lesson introduced you to variables, a powerful concept you will use in all your future programming endeavors.

Let’s review what we learned:

* Variables hold reusable data in a program and associate it with a name.
* Variables are stored in memory.
* The var keyword is used in pre-ES6 versions of JS.
* let is the preferred way to declare a variable when it can be reassigned, and const is the preferred way to declare a variable with a constant value.
* Variables that have not been initialized store the primitive data type undefined.
* Mathematical assignment operators make it easy to calculate a new value and assign it to the same variable.
* The + operator is used to concatenate strings including string values held in variables
* In ES6, template literals use backticks ` and ${} to interpolate values into a string.
* The typeof keyword returns the data type (as a string) of a value.

**JavaScript Versions: ES6 and Before**

**Ever heard of the term "ES6" and wondered what it's about? Click on this article to read and find out!**

You might have seen the term “ES6” or “Javascript ES6” and wondered what it actually means. Well wonder no further, because we’re going to dive into what ES6 is and how it relates to Javascript!

First, let’s bring in some history. JavaScript was introduced in 1995 by the company Netscape Communications as a scripting language for web designers and programmers to interact with web pages. The next year, Netscape submitted JavaScript to a standards developing organization called Ecma International to create standards for a scripting language (a type of programming language). In 1997, Ecma International released ECMA-262 which sets standards for the first version of a scripting language called ECMAScript, shortened to ES.

These new ECMAScript standards provided rules for the architecture of JavaScript features. As new programming paradigms emerged and developers sought new features, newer versions of ECMAScript provided a basis for a consistency between new and old JavaScript versions.

To fully distinguish the difference between JavaScript and ECMAScript: if you want to create an app or program you can use JavaScript — if you want to create a new scripting language you can follow the guidelines in ECMAScript. So, when you see ES6 or JavaScript ES6, it means that that version of JavaScript is following the specifications in the sixth edition of ECMAScript! You might also see ES2015 instead of ES6, but both terminologies are referring to the same 6th edition of ECMAScript that was released in 2015. Take a look at the timeline below to see how JavaScript has evolved over the years:

Now, you may be asking, what makes an update in 2015 still relevant today when there are more recent updates like ES7 and ES8?

Well, despite the release of newer versions, ES6 is actually the biggest update made to ECMAScript since the first edition released in 1997! Some developers even refer to ES6 as “Modern JavaScript” because of all the major additions. There were so many great features added to help JavaScript developers that include:

* new keywords like let and const to declare variables,
* new function syntax using Arrow functions,
* creation of Classes,
* parameters with default values,
* promises for asynchronous actions,
* and many more!

Up-to-date browsers now support most of the ES6 features which allow developers to take advantage of these new additions. ES6 ultimately allows programmers to save time and write more concise code. Take for example pre-ES6 syntax for function expressions:

var greeting = function() {

console.log('Hello World!');

};

With ES6 arrow functions, we can transform the expression above into:

const greeting = () => console.log('Hello World');

However, arrow functions are not just simply syntactical re-writes. As with other ES6 features, there are other underlying benefits and tradeoffs to consider. Nonetheless, there has been a strong adoption of ES6 in the development community. Benefits such as new ES6 syntax, makes it easier to utilize a popular programming paradigm, Object Oriented Programming (OOP). This change allowed for developers of other languages who are used to OOP can now transition into learning and using JavaScript. Another reason for the popularity of ES6 is correlated with the usage of ES6 in popular frameworks like React. So, if you want to learn the newest tools and frameworks, you will have to pick up ES6 along the way.

This being said, we shouldn’t disregard legacy code, i.e. older versions of JavaScript. In fact, there are still many projects that are built and maintained with legacy code! If you want the ability and freedom to work on any sort of JavaScript project, you should familiarize yourself with pre-ES6 and ES6 JavaScript syntax. But don’t worry, we cover both pre-ES6 and ES6 in our JavaScript course. Check it out to become a rockstar at JavaScript basics and learn fundamental programming skills!

Review

Way to go! Here are some of the major concepts for conditionals:

* An if statement checks a condition and will execute a task if that condition evaluates to true.
* if...else statements make binary decisions and execute different code blocks based on a provided condition.
* We can add more conditions using else if statements.
* Comparison operators, including <, >, <=, >=, ===, and !== can compare two values.
* The logical and operator, &&, or “and”, checks if both provided expressions are truthy.
* The logical operator ||, or “or”, checks if either provided expression is truthy.
* The bang operator, !, switches the truthiness and falsiness of a value.
* The ternary operator is shorthand to simplify concise if...else statements.
* A switch statement can be used to simplify the process of writing multiple else if statements. The break keyword stops the remaining cases from being checked and executed in a switch statement.

Review Functions

Give yourself a pat on the back, you just navigated through functions!

In this lesson, we covered some important concepts about functions:

* A *function* is a reusable block of code that groups together a sequence of statements to perform a specific task.
* A *function declaration* :
* A parameter is a named variable inside a function’s block which will be assigned the value of the argument passed in when the function is invoked:
* To *call* a function in your code:
* ES6 introduces new ways of handling arbitrary parameters through *default parameters* which allow us to assign a default value to a parameter in case no argument is passed into the function.
* To return a value from a function, we use a *return statement*.
* To define a function using *function expressions*:
* To define a function using *arrow function notation*:
* Function definition can be made concise using concise arrow notation:

It’s good to be aware of the differences between function expressions, arrow functions, and function declarations. As you program more in JavaScript, you’ll see a wide variety of how these function types are used.

Review: Scope

In this lesson, you learned about scope and how it impacts the accessibility of different variables.

Let’s review the following terms:

* **Scope** is the idea in programming that some variables are accessible/inaccessible from other parts of the program.
* **Blocks** are statements that exist within curly braces {}.
* **Global scope** refers to the context within which variables are accessible to every part of the program.
* **Global variables** are variables that exist within global scope.
* **Block scope** refers to the context within which variables that are accessible only within the block they are defined.
* **Local variables** are variables that exist within block scope.
* **Global namespace** is the space in our code that contains globally scoped information.
* **Scope pollution** is when too many variables exist in a namespace or variable names are reused.

As you continue your coding journey, remember to use best practices when declaring your variables! Scoping your variables tightly will ensure that your code has clean, organized, and modular logic.

Review Arrays

Nice work! In this lesson, we learned these concepts regarding arrays:

* Arrays are lists that store data in JavaScript.
* Arrays are created with brackets [].
* Each item inside of an array is at a numbered position, or index, starting at 0.
* We can access one item in an array using its index, with syntax like: myArray[0].
* We can also change an item in an array using its index, with syntax like myArray[0] = 'new string';
* Arrays have a length property, which allows you to see how many items are in an array.
* Arrays have their own methods, including .push() and .pop(), which add and remove items from an array, respectively.
* Arrays have many methods that perform different tasks, such as .slice() and .shift(), you can find documentation at the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) website.
* Some built-in methods are mutating, meaning the method will change the array, while others are not mutating. You can always check the documentation.
* Variables that contain arrays can be declared with let or const. Even when declared with const, arrays are still mutable. However, a variable declared with const cannot be reassigned.
* Arrays mutated inside of a function will keep that change even outside the function.
* Arrays can be nested inside other arrays.
* To access elements in nested arrays chain indices using bracket notation.

Learning how to work with and manipulate arrays will help you work with chunks of data!

Review

Great job! In this lesson, we learned how to write cleaner code with loops. You now know:

* Loops perform repetitive actions so we don’t have to code that process manually every time.
* How to write for loops with an iterator variable that increments or decrements
* How to use a for loop to iterate through an array
* A nested for loop is a loop inside another loop
* while loops allow for different types of stopping conditions
* Stopping conditions are crucial for avoiding infinite loops.
* do...while loops run code at least once— only checking the stopping condition after the first execution
* The break keyword allows programs to leave a loop during the execution of its block

VERY IMPORTANT:

Review

Great job! By thinking about functions as data and learning about higher-order functions, you’ve taken important steps in being able to write clean, modular code and take advantage of JavaScript’s flexibility.

Let’s review what we learned in this lesson:

* Abstraction allows us to write complicated code in a way that’s easy to reuse, debug, and understand for human readers
* We can work with functions the same way we would any other type of data including reassigning them to new variables
* JavaScript functions are first-class objects, so they have properties and methods like any object
* Functions can be passed into other functions as parameters
* A higher-order function is a function that either accepts functions as parameters, returns a function, or both