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

# ITERATOR DOCUMENTATION:

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global\_Objects/Array#Iteration\_methods

Review

Awesome job on clearing the iterators lesson! You have learned a number of useful methods in this lesson as well as how to use the JavaScript documentation from the Mozilla Developer Network to discover and understand additional methods. Let’s review!

* .forEach() is used to execute the same code on every element in an array but does not change the array and returns undefined.
* .map() executes the same code on every element in an array and returns a new array with the updated elements.
* .filter() checks every element in an array to see if it meets certain criteria and returns a new array with the elements that return truthy for the criteria.
* .findIndex() returns the index of the first element of an array which satisfies a condition in the callback function. It returns -1 if none of the elements in the array satisfies the condition.
* .reduce() iterates through an array and takes the values of the elements and returns a single value.
* All iterator methods takes a callback function that can be pre-defined, or a function expression, or an arrow function.
* You can visit the [Mozilla Developer Network](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array) to learn more about iterator methods (and all other parts of JavaScript!).

Review

Way to go! You’re well on your way to understanding the mechanics of objects in JavaScript. By building your own objects, you will have a better understanding of how JavaScript built-in objects work as well. You can also start imagining organizing your code into objects and modeling real world things in code.

Let’s review what we learned in this lesson:

* Objects store collections of *key-value* pairs.
* Each key-value pair is a property—when a property is a function it is known as a method.
* An object literal is composed of comma-separated key-value pairs surrounded by curly braces.
* You can access, add or edit a property within an object by using dot notation or bracket notation.
* We can add methods to our object literals using key-value syntax with anonymous function expressions as values or by using the new ES6 method syntax.
* We can navigate complex, nested objects by chaining operators.
* Objects are mutable—we can change their properties even when they’re declared with const.
* Objects are passed by reference— when we make changes to an object passed into a function, those changes are permanent.
* We can iterate through objects using the For...in syntax.

Review

Congratulations on finishing Advanced Objects!

Let’s review the concepts covered in this lesson:

* The object that a method belongs to is called the *calling object*.
* The this keyword refers the calling object and can be used to access properties of the calling object.
* Methods do not automatically have access to other internal properties of the calling object.
* The value of this depends on where the this is being accessed from.
* We cannot use arrow functions as methods if we want to access other internal properties.
* JavaScript objects do not have built-in privacy, rather there are conventions to follow to notify other developers about the intent of the code.
* The usage of an underscore before a property name means that the original developer did not intend for that property to be directly changed.
* Setters and getter methods allow for more detailed ways of accessing and assigning properties.
* Factory functions allow us to create object instances quickly and repeatedly.
* There are different ways to use object destructuring: one way is the property value shorthand and another is destructured assignment.
* As with any concept, it is a good skill to learn how to use the documentation with objects!

You’re ready to start leveraging more elegant code for creating and accessing objects in your code!

Review: Classes

Way to go! Let’s review what you learned.

* *Classes* are templates for objects.
* Javascript calls a *constructor* method when we create a new instance of a class.
* *Inheritance* is when we create a parent class with properties and methods that we can extend to child classes.
* We use the extends keyword to create a subclass.
* The super keyword calls the constructor() of a parent class.
* Static methods are called on the class, but not on instances of the class.

In completing this lesson, you’ve taken one step closer to writing efficient, production-level JavaScript. Good luck as you continue to develop your skills and move into intermediate-level concepts.

Review

In this lesson, you learned about browser compatibility and transpilation. Let’s review the key concepts:

* ES5 — The old JavaScript version that is supported by all modern web browsers.
* ES6 — The new(er) JavaScript version that is *not* supported by all modern web browsers. The syntax is more readable, similar to other programming languages, and addresses the source of common bugs in ES5.
* caniuse.com — a website you can use to look up HTML, CSS, and JavaScript browser compatibility information.
* Babel — A JavaScript package that transpiles JavaScript ES6+ code to ES5.
* npm init — A terminal command that creates a **package.json** file.
* **package.json** — A file that contains information about a JavaScript project.
* npm install — A command that installs Node packages.
* babel-cli — A Node package that contains command line tools for Babel.
* babel-preset-env — A Node package that contains ES6+ to ES5 syntax mapping information.
* **.babelrc** — A file that specifies the version of the JavaScript source code.
* "build" script — A **package.json** script that you use to tranpsile ES6+ code to ES5.
* npm run build — A command that runs the build script and transpiles ES6+ code to ES5.

For future reference, here is a list of the steps needed to set up a project for transpilation:

1. Initialize your project using npm init and create a directory called **src**
2. Install babel dependencies by running
3. npm install babel-cli -D

npm install babel-preset-env -D

1. Create a **.babelrc** file inside your project and add the following code inside it:
2. {
3. "presets": ["env"]

}

1. Add the following script to your scripts object in **package.json**:

"build": "babel src -d lib"

1. Run npm run build whenever you want to transpile your code from your **src** to **lib** directories.

Review

We just learned how to use JavaScript modules. Let’s review what we learned:

*Modules* in Node.js are reusable pieces of code that can be exported from one program and imported for use in another program.

* module.exports exports the module for use in another program.
* require() imports the module for use in the current program.

ES6 introduced a more flexible, easier syntax to export modules:

* default exports use export default to export JavaScript objects, functions, and primitive data types.
* named exports use the export keyword to export data in variables.
* named exports can be aliased with the as keyword.
* import is a keyword that imports any object, function, or data type.

Review

Awesome job! Promises are a difficult concept even for experienced developers, so pat yourself on the back. You’ve learned a ton about asynchronous JavaScript and promises. Let’s review:

* Promises are JavaScript objects that represent the eventual result of an asynchronous operation.
* Promises can be in one of three states: pending, resolved, or rejected.
* A promise is settled if it is either resolved or rejected.
* We construct a promise by using the new keyword and passing an executor function to the Promise constructor method.
* setTimeout() is a Node function which delays the execution of a callback function using the event-loop.
* We use .then() with a success handler callback containing the logic for what should happen if a promise resolves.
* We use .catch() with a failure handler callback containing the logic for what should happen if a promise rejects.
* Promise composition enables us to write complex, asynchronous code that’s still readable. We do this by chaining multiple .then()‘s and .catch()‘s.
* To use promise composition correctly, we have to remember to return promises constructed within a .then().
* We should chain multiple promises rather than nesting them.
* To take advantage of concurrency, we can use Promise.all().

Awesome work getting the hang of the async...await syntax! Let’s review what you’ve learned:

* async...await is syntactic sugar built on native JavaScript promises and generators.
* We declare an async function with the keyword async.
* Inside an async function we use the await operator to pause execution of our function until an asynchronous action completes and the awaited promise is no longer pending .
* await returns the resolved value of the awaited promise.
* We can write multiple await statements to produce code that reads like synchronous code.
* We use try...catch statements within our async functions for error handling.
* We should still take advantage of concurrency by writing async functions that allow asynchronous actions to happen in concurrently whenever possible.

Review Requests I

You’ve done an amazing job navigating through making XHR GET and POST requests! Take some time to review the core concepts before moving on to the next lesson.

1. JavaScript is the language of the web because of its asynchronous capabilities. AJAX, which stands for Asynchronous JavaScript and XML, is a set of tools that are used together to take advantage of JavaScript’s asynchronous capabilities.
2. There are many HTTP request methods, two of which are GET and POST.
3. GET requests only request information from other sources.
4. POST methods can introduce new information to other sources in addition to requesting it.
5. GET requests can be written using an XMLHttpRequest object and vanilla JavaScript.
6. POST requests can also be written using an XMLHttpRequest object and vanilla JavaScript.
7. Writing GET and POST requests with XHR objects and vanilla JavaScript requires constructing the XHR object using new, setting the responseType, creating a function that will handle the response object, and opening and sending the request.
8. To add a query string to a URL endpoint you can use ? and include a parameter.
9. To provide additional parameters, use & and then include a key-value pair, joined by =.
10. Determining how to correctly write the requests and how to properly implement them requires carefully reading the documentation of the API with which you’re working.

Review Requests II

Let’s recap on the concepts covered in the previous exercises:

1. GET and POST requests can be created a variety of ways.
2. Use AJAX to asynchronously request data from APIs. fetch() and async/await are new functionalities developed in ES6 (promises) and ES8 respectively.
3. Promises are a new type of JavaScript object that represent data that will eventually be returned from a request.
4. fetch() is a web API that can be used to create requests. fetch() will return promises.
5. We can chain .then() methods to handle promises returned by fetch().
6. The .json() method converts a returned promise to a JSON object.
7. async is a keyword that is used to create functions that will return promises.
8. await is a keyword that is used to tell a program to continue moving through the message queue while a promise resolves.
9. await can only be used within functions declared with async.