“Advanced” JavaScript

Hunter Jorgensen

Classes and Prototypes

In 2016, classes were introduced to JavaScript. However, classes are a LIE as JavaScript uses a novel version of object-oriented behavior based on prototyping (classes are just syntactic/syntax sugar [what does this mean?]). This means that you can access all of the objects of a similar type and give them the same functionality.

Please never use prototypes to modify the classes of JavaScript

Demo

JavaScript Event Loop

This is probably the most important topic to cover in JavaScript.

JavaScript was created to handle a very specific problem: handling logic on websites.

But websites make many calls to other websites, services, APIs, etc. (look to see Amazon’s network calls.)

JavaScript Event Loop ELI5

(also, this might be a bit clearer if you've taken Operating Systems and learned about concurrency)

We can think of website logic like a recipe: do one step, then another, then another.

JavaScript Event Loop ELI5

Recipe! Easy to follow

1. Chop some carrots.
2. Chop some onions.
3. Add water into a pot, turn the stove on and wait for it to boil.
4. Add carrots into a pot and let it cook for 5 minutes.
5. Add onions into a pot and let it cook for another 10 minutes.

JavaScript Event Loop ELI5

But a good cook will be able to save time by doing several things at a time:

1. Start boiling a pot of water
2. **While waiting for the pot of water to boil,** start chopping up some carrots.
3. **By the time you finish** chopping carrots, the water should be boiling, so add the carrots.
4. **Whilst the carrots are cooking in the pot,** chop up the onions.
5. Add the onions and then cook for another 10 minutes.

JavaScript makes this behavior very easy, but using callbacks, Promises or async/await. (They're all similar ways of doing the same thing - *syntactic sugar* - but we'll just focus on callbacks for now.)

JavaScript Event Loop and Callbacks

With callbacks, JavaScript allows us to easily perform logic once it's done waiting for whatever response or operation. This can be incredibly hard to understand if you're new to ideas like concurrency or API calls.

We'll be exploring these ideas more in the coming weeks! We're going to dive a bit more into it now, but don't worry if you don't understand it quite yet: it's just to get you aware of these ideas.

JavaScript Event Loop ELI5

In other words, JavaScript will sometimes performs functions that allows the code to wait and JavaScript is smart enough to do something else while waiting.

Note - remember that you can pass functions as arguments into other functions: that’s what’s happening here.

Demo

Callback Hell

In the previous example, we saw an example of a callback. But what if we needed to make a call in order to make another call? We would have to nest our queries:

Demo

Now, imagine this with numerous calls all dealing with complicated data. An example of this might be a website getting a user's login information, and then using that to get all the books they've bought, and then all the reviews for those books!

Promises!

JavaScript has something called a promise that allows us to pipe the request into a chain of easier to read logic calls.

Promises are just a different way of handling callbacks, but one that is hopefully more readable!

Demo

Syntactic Sugar

In the last few years, JavaScript has developed functionality to make it more readable for developers from a variety of background. In order to accomplish this, JavaScript has a lot of functionality that isn't new, but is rather an existing functionality wrapped in an easier to read or easier to code way.

Depending on your background, you may find some of these ideas easier to read and approach.

These are all things you don't need to know, but may be fun to try out!

Syntactic Sugar: Ternary Operators

Ternary operators (which are common in many modern programming languages) allows you to do simple if statements in a single line

Demo

Syntactic Sugar: Array/Object Spread

When you spread with an object or array, you use an ellipsis to pull all the values our and basically reassign them.

This is a good way to clone or combine multiple arrays/objects without iterating over them (which is going on behind the hood.)

Demo

Syntactic Sugar: Destructuring

Object Destructing allows you to quickly pull values out of an array or object by re-assigning them to new variables.

Demo

The use of the ellipsis to grab the remaining values is known as *rest*, and works similarly to spreading!

Syntactic Sugar: Object Shorthand

The opposite of destructuring is known as object shorthand. This is a bit funky to see, but effectively allows us to shortcut the creation of objects if we set the variable to be the name of the keys

Syntactic Sugar: Arrow Functions

Arrow Functions (similar to lambdas in other programming languages; also known as anonymous functions) are a compact way to write a function declaration. They are only suited for extremely simple method calls however, due to readability.

Demo

Advanced Array Functions

Aside from things like ‘put’, ‘pop’, ‘shift’, etc., we can do some pretty clever interactions with JavaScript arrays with functions like “map”, “filter”, etc. These take a function and apply it somehow to the array.

Syntactic Sugar: Async/Await

Async/Await is another way of handling Promises in JavaScript.

These are both keywords:

- async tells JavaScript that the method getting called in asynchronous (i.e., JavaScript can do other work while waiting for a response)
- await tells JavaScript the specific piece of logic that may cause the delay in execution

Demo

We will spend more time with async behavior (both async/await and promises) in the next module, so don’t worry if this isn’t clear yet.

Funky Behavior

JavaScript gets a lot of flak for some odd behaviors. If you want to become a more advanced user of JavaScript, you are welcome to try and understand why it behaves this way, but typically logic like this is considered fairly esoteric.

WTF JS

Strictly speaking - Strict Mode

Often times in JavaScript, you may see the line: 'use strict'; at the top of your code. This is a change to the actual functionality of JavaScript and is generally recommended. Some of the changes includes:

- Catching some common coding errors and throwing exceptions.
- It prevents, or throws errors, when "unsafe" actions are taken, like accessing the global object.
- It disables confusing or poorly implemented features.