Important JavaScript Concepts to Learn

**1. Control Flow**

Probably the most basic topic on the list. One of the most important, maybe the most important one. If you do not know how to proceed with your code, you will have a hard time. Knowing the ins and outs of basic control flow is definitely a **must**.

* if else — If you don’t know these, how did you write code before?
* switch — is basically if else in a more eloquent way, use it as soon as you have multiple of different cases.
* for — Do not repeat yourself, this is what loops are for. Besides the normalfor -loop `for of` and for in come in very handy. The big advantage of for -loops is that they are blocking, so you can use async await in them.
* Advanced conditionals — Using the ternary and logical operators can make your life a lot easier, especially when you try to do things inline, meaning that you don’t want to save values to use them later. Example:

// ternaryconsole.log(new Date().getHours() < 12 ? 'Good Morning!' : 'Time for a siesta')  
// logical operatorsconst isJsMaster = prompt('Are you a JavaScript master?') === 'true'  
console.log(isJsMaster && 'proficient coder')

**2. Error handling**

This took a while for me. It does not matter if you are working on frontend or backend, the first year or so, you will probably default to console.log or maybe console.error for ‘handling’ errors. To write good applications, you definitely have to change that and replace your lazy logs with nicely handled errors. You may want to check out how to build your own Error constructor and how to catch them correctly, as well as showing the user what the actual problem is.

UPDATE: Check out the [article](https://medium.com/@gisderdube_55369/the-definite-guide-to-handling-errors-gracefully-in-javascript-58424d9c60e6) about how to handle Errors gracefully!

**3. Data Models**

Similar to moving through your application continuously, you have to decide where to group specific information chunks and where to keep them separate. This does not only apply to building database models, but also function parameters and objects or variables. Example:

const calcShape = (width, height, depth, color, angle) => {...}const calcShape = ({width, height, depth, color, angle}) => {...}

**4. Asynchronity**

This is a very important aspect of JavaScript, Either you are fetching data from the backend or you are processing requests asynchronously in the backend itself. In pretty much all usecases, you will encounter asynchronity and its caveats. If you have no idea what that is, you will probably get a weird error, which you will try to fix for a couple of hours. If you know what it is, but you don’t really know what to do about it, you will end up in callback-hell. The better approach is to use promises and/or async await in your apps.

**5. DOM Manipulation**

This is an interesting topic. Normally it is somewhat left out in the day today life as a developer. Maybe you learned jQuery and never felt the need to pick up some native DOM manipulation skills, maybe you are just using a frontend framework, where there is rarely a need for custom DOM manipulation. However, I think this is a crucial part of understanding JavaScript, at least in the frontend. Knowing how the DOM works and how to access elements gives you a deep understanding of how websites work. In addition, there **will** be the point where you have to do some custom DOM manipulation, even when you use modern frontend frameworks, and you definitely do not want to put jQuery in your package.json just to access an element.

Photo by [LinkedIn Sales Navigator](https://unsplash.com/photos/0QvTyp0gH3A?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText) on [Unsplash](https://unsplash.com/search/photos/professional?utm_source=unsplash&utm_medium=referral&utm_content=creditCopyText" \t "_blank)

**6. Node.js / Express**

Even as a frontend developer, you should know the basics of node.js. Ideally, you would also know how to spin up a simple express server and add some routes or change existing ones. JavaScript is great for writing scripts to help you automate a lot of tasks. Therefore, knowing how to read files, work with filepaths or buffers gives you a good toolset to build anything.

**7. Functional Approach**

There is an everlasting debate about functional vs. object-oriented programming. You probably can achieve the same thing with both of the approaches. In JavaScript, it is even easier, you have both of the approaches available. Libraries like [lodash](https://lodash.com/" \t "_blank) give you a really nice collection of tools for building applications with a functional approach. Nowadays, it is not even necessary to use external libraries any more. A lot of the most important functions have been implemented in the official JavaScript specification. You definitely should know how to use map `reduce` filter `forEach` and `find`.

**8. Object Oriented Approach**

Similar to the functional approach, you also have to get familiar with object oriented JavaScript, if you want to master it. I neglected that part for a long time in my career and just worked my way through with a workaround, but sometimes it is definitely better to use objects/classes and instances to implement specific functionality. Classes are widely used in React, MobX or custom constructors.

**9. Frontend Framework**

The big three are React.js, Angular and Vue.js. If you are looking for a job nowadays, you will almost always have one of those listed as a prerequisite. Even if they change quite quickly, it is important to grasp the general concept of those to understand how applications work. Also, it is just easier to write apps that way. If you haven’t decided which train you want to jump on, my suggestions is React.js. I have been working with it for the last couple of years and did not regret my decision.

**10. Bundling / Transpilation**

Unfortunately, this is a big part of web development. On the one hand I should not say unfortunate, because it is great to be able to write code with all the newest features. On the other hand, the reason why I’m saying that is that we always have to keep in mind that there’s older browsers around that may not support these features, therefore we have to transpile our code into something else that the old browsers understand. If you work with node.js, you will probably have less exposure to transpiling your code. The de-facto standard for transpilation is [babel.js](https://babeljs.io/), so get familiar with it. As for bundling your code and tying everything together, you have a couple of options. [Webpack](https://webpack.js.org/) was the dominant player for a long time. Some time ago, [parcel](https://parceljs.org/) popped up out of nowhere and is now my preferred solution, since it is so performant and easy to configure, although not perfect.

**BONUS: Regular Expressions**

This is not specific to JavaScript, but incredibly helpful in a lot of use cases. Just as confusing as well. Getting to know the syntax of Regular Expressions definitely takes some time and remembering all of the different options is impossible.

**Update: Testing**

 pointed out, testing is a really important part of software development, JavaScript is no exception. When writing code, you (hopefully) test it before you push the feature, even if it might be manual. A better approach is using automated tests, different test types are unit testing, end-to-end testing, load testing, security tests or frontend-tests (e.g. is a component mounted or not). There are a lot of different test environments, enzyme, jasmine, mocha, chai, etc. My favourite solution at the moment is [ava.js](https://github.com/avajs/ava), so go check it out if you did not work with automated tests so far.