[JSR] – Improving your JavaScript fundamentals (The Teenage Phase)

1 message

Zell Liew <zell@zellwk.com>
To: Anibal <anibalsantosgo@gmail.com>
Fri, Mar 8, 2019 at 1:12 PM

Hey Anibal,

You’re in the Teenage Phase if you already know how to build stuff from scratch. You’re somewhat confident that you’ll be able to build anything DOM related. At this point, your code is still messy and unorganized. You’re probably not too happy about it.

Your focus for the Teenage Phase

Your focus for this phase is to build even more things. Build things your boss ask you to. Build things your friends and relatives want. Build things that are fun for yourself.

Keep building. The more you build, the more experience you accumulate. As you build, learn to incorporate these four things:

1. Object-oriented Programming (OOP)
2. Functional Programming ideas (FP)
3. Asynchronous JavaScript (AJAX)
4. JavaScript Best Practices

Let’s go through each of them in more detail.

Object-oriented Programming

Both OOP and FP are popular programming styles in JavaScript. In order to get good with JavaScript, you need to know both. There is no need to dive too deep into either programming styles right now, but you need to know the basics of them both.

OOP is a style of programming that revolves around objects. At this stage, strive to learn these concepts for OOP:

1. `this` in JavaScript
2. JavaScript prototypes
3. The Module and Factory patterns for creating objects

Functional Programming

FP is a style of programming that revolves around actions you perform with functions. In FP, you manipulate data and pass them around through functions. It is completely different from OOP.

For FP, strive to learn these concepts:

1. Reduce side effects
2. Write pure functions
3. Write immutable code

Forget about currying and partial application (you’d see them as you research FP) for now. Trying to learn these two principles would surely confuse you at this stage. You can learn them later when you’re better.

Asynchronous JavaScript

JavaScript is single-threaded (it can only do one thing at a time). The key to JavaScript is to be comfortable with asynchronous JavaScript.

If you followed the roadmap so far, you would have already dipped your toes into AJAX with callbacks. The next step is to learn to use AJAX with the Fetch API and JavaScript Promises.

You may also want to learn to read API so you can use third-party (like Github's or Twitter's) API.

Here are some articles that will be useful:

1. Using Fetch
2. JavaScript Promises
3. Reading APIs (Article still under construction. I'll update you as soon as its ready!)

JavaScript Best Practices

Best practices are important. But they're the hard to learn, for good reasons.

First, there isn't a compendium of best practices lying around for JavaScript.

(I'm making one with Sitepoint now).

You often find best practices within articles and books. But the confusing thing is, best practices may conflict with each other.

That's because best practices are written by people. Different people have different opinions. You have to learn which ones to keep and which ones to throw.

Second you need to change the way you're coding now to adapt to best practices. You need to rewire your brain again.

I recommend you try to incorporate best practices by modifying your existing code. Make them better. In technical jargon, we call it refactoring.

Don't try to write code with best practices from scratch at this point. If you do so, you'll try to think logically and in best practices at the same time. Brains can't handle that. You'll end up in a mess.

Alright. So, best practices are hard. There's no one place where you can learn them. How do you learn then?

The best way is to read other peoples' code.

See how they're different from yours. Ask why. Understand why. Then, use them in your code.

Where to find other peoples' code? Your best bet is books and courses. As you read through examples, notice how they structure their code.

Your second best bet are plugins, modules and open-sourced code. They're harder to read than books and courses, but they show you how people code for real.

Don't try writing the perfect piece of code

You'll never reach it. Not now, at least.

Three months later, you'll look back and vomit at the shitty code you've wrote. Another three months later, you'll do the same. The cycle repeats.

Keep building new things. Once in a while, come back, refactor, then move on.

Note: Best practices change overtime as technology evolves. Some practices that are celebrated three years ago may be frowned upon now. The bulk of them did stay the same though. In the list of questions below, you'd find a list of best practices to know by heart.

Some questions to check your understanding

1. OOP
    1. How does `this` changes in different context? How many contexts are there?
    2. What is a prototype in JavaScript?
    3. How do you create objects in JavaScript?
    4. What is the module pattern? When do you use it?
    5. What is the factory pattern? When do you use it?
2. FP
    1. What is immutability?
    2. What array methods are immutable?

        3. How do you change JavaScript properties while not mutating the object?
        4. What is a pure function?
        5. How many kinds of actions should a function contain?
        6. What are side effects?
        7. How do you handle side effects when you write pure functions?
3. AJAX
    1. What are JavaScript promises?
    2. How do you chain promises?
    3. How do you catch errors when using promises?
    4. How do you use the Fetch API?
    5. What does CRUD stand for?
    6. How do you query Github’s API to get a list of your own repositories?
4. Best practices
    1. Why do you avoid global variables?
    2. Why use strict equality (===) instead of normal equality (==)?
    3. How do you use ternary operators to help you write terser code?
    4. What ES6 features help you write terser code?
    5. What is event bubbling and capturing?
    6. How do you delegate events?
    7. How do you remove event listeners? When should you remove them?

That’s it for the Teenage Phase. It’s tough, but not impossible to complete. It might take you a few months if you quick, and probably 1-2 years to grasp it all if you can’t spend time on learning JavaScript. Once again, don’t let this stop you. You can move on anytime.

You’ll learn more about the Adult Phase next. This is where it gets exciting for some of you.

Till then. Work on the Teenage Phase.

Stay awesome,
Zell
