How to Progress Rapidly as a Developer

Every craft has a set of foundational skills. Here's what's needed to speed up learning to code

Szymon Adamiak
May 6 · 5 min read ★

Learning to code is like running to the top. Photo by Mauro Paillex on Unsplash

You can learn the basics of programming in no time. Countless resources will get you going. The hard part is to take a step from simple conditionals and loops to real-life applications. Fortunately, you can develop a few simple skills to help you along the way.

. . .

Become a Google Master

You'll spend lots of time searching for information, and you need to do it effectively. Googling is a transferrable skill, so if you researched in depth any subject, you'd be fine. But make sure you put on quotes to search for the whole phrase, know how to set a date range, and use a hyphen to exclude words from search results.

You also need to master programming-specific Google skills. Not all information sources are created equal. Pay attention to websites and people that helped you solve a problem. Over time you'll build a robust database of reliable resources.

Technical terms matter

Remember relevant technical terms. It will prove a lot more beneficial than you think. Programmers have a particular language to communicate fast and accurately. To find the information, you need to understand that language. For example, it's a lot easier to discover why you don't have access to the variable if you know the term *scope*. Note every technical name and define it, if possible, using your own words.

Besides, when you know proper terms, you don't need to read so much. Skim the internet. Search for the keywords in the text instead of reading everything.

. . .

Focus When You Make a Mistake

When coding, most of the time, you're on autopilot. But to progress as a developer, you need to be mindful of the code. Use errors as reminders to focus.

Mistakes are the perfect opportunities to learn new things. A new error or unexpected bug may vastly deepen your code understanding. When encountered, take a deep breath and analyze what happened. After you solve the issue — take notes. You'll stumble on this same error again. Without the notes, it’s almost sure you’ll forget how to fix it.

Try not to take mistakes personally. Treat them as friends who help you develop your skills. If you need more information about a productive attitude towards errors, I wrote an article about that.

You'll improve next time. Photo by Jonas Denil on Unsplash

. . .

Don't Waste Your Time

When you've mastered coding fundamentals, the learning curve becomes steeper. New concepts are significantly harder to grasp. You need to learn effectively in order to progress.

Assume you've watched a great tutorial. You coded along and built an app with your teacher. But two days later you don't remember much. When you try to use new knowledge, you get stuck. You feel the urge to watch that great tutorial again.

Don't do it.

Time is a precious resource. You shouldn't use it for rewatching or rereading.

Here are three simple techniques that help you study effectively.

Active recall

Active recall is a process in which you try to retrieve information from memory, instead of rereading it. A typical example is the use of flashcards.

You can use active recall anytime when learning to code. Next time when you're watching or reading a tutorial, don't code. Focus on following a teacher. Then try to recreate code from memory. Only get back to the lesson if you can't reproduce some part.

Spaced repetition

You'll understand new concepts much better if you learn in smaller chunks spread over time than if you cram in one sitting. Create a repetition system with the breaks increasing each time.

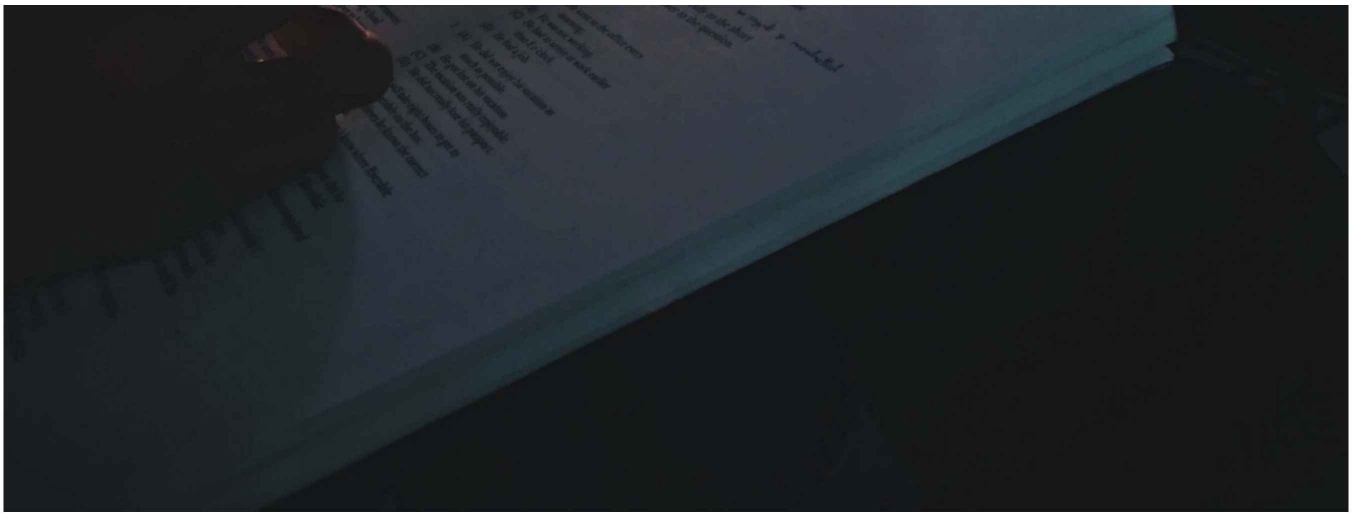
For example, learn about closures for thirty minutes. Check out what the closure is and when to use it, see some code samples, take notes, or make flashcards. Tomorrow, try to recall information about closure from memory. If you succeed, try again after three days. Each time lengthen a delay between repetitions. If you fail — stick to this same delay.

Experiment with different schedules to find the one that suits you the best.

Tests

Remember how you used to learn at school? You may have studied over all the semester, but you worked hardest and learned the most before the exam. Use that to your advantage.

Create a simple schedule. For example, devote two hours to testing once a month. You can easily find tests online or create them by yourself. The exam doesn't need to have a traditional form. It's even better if you test yourself by making an app using the knowledge you want to solidify.



You don’t need to go back to school to take a test. Photo by Yustinus Tjiuwanda on Unsplash

. . .

Watch Out for Procrastination

There’s one way to learn to code: coding. But the world conspires to distract you — new articles, tutorials, courses fight for your attention. And the teacher in the last video had a great font in his IDE; you want to get it.

It’s all background noise. Don’t google for fonts or themes — yours are okay. Don’t watch another tutorial on this same topic. I know it’s easy, but it’s a waste of time. Your time is limited, use it wisely.

Build more side projects using the knowledge from tutorials. Don’t abandon the project after the first setback. Developers who take a break from the project don’t get back. So when you get stuck during an exciting project, ask for help. The programmers’ community is vast; lots of people want to help you.

. . .

Don’t Overthink — Try to Understand

Almost everything you’re trying to do right now has been done before. For your next project, you don’t need to create authentication or routing on your own. Numerous formidable and tested libraries are waiting for you.

Choosing the right tools is a part of your job. You need to become an expert in finding libraries and frameworks to help you. As developers, we're standing on the shoulders of giants — leverage that.

But there's a caveat. To pick your tools correctly, you need to strive to understand them. In your spare time, analyze the code of the libraries you're using. You can even do a fun project of trying to recreate them. Many libraries are not that complex at their core. Think about how you would implement them.

. . .

Thanks for reading! Good luck!

Thanks to Zack Shapiro.

Programming Education Learning To Code Python JavaScript