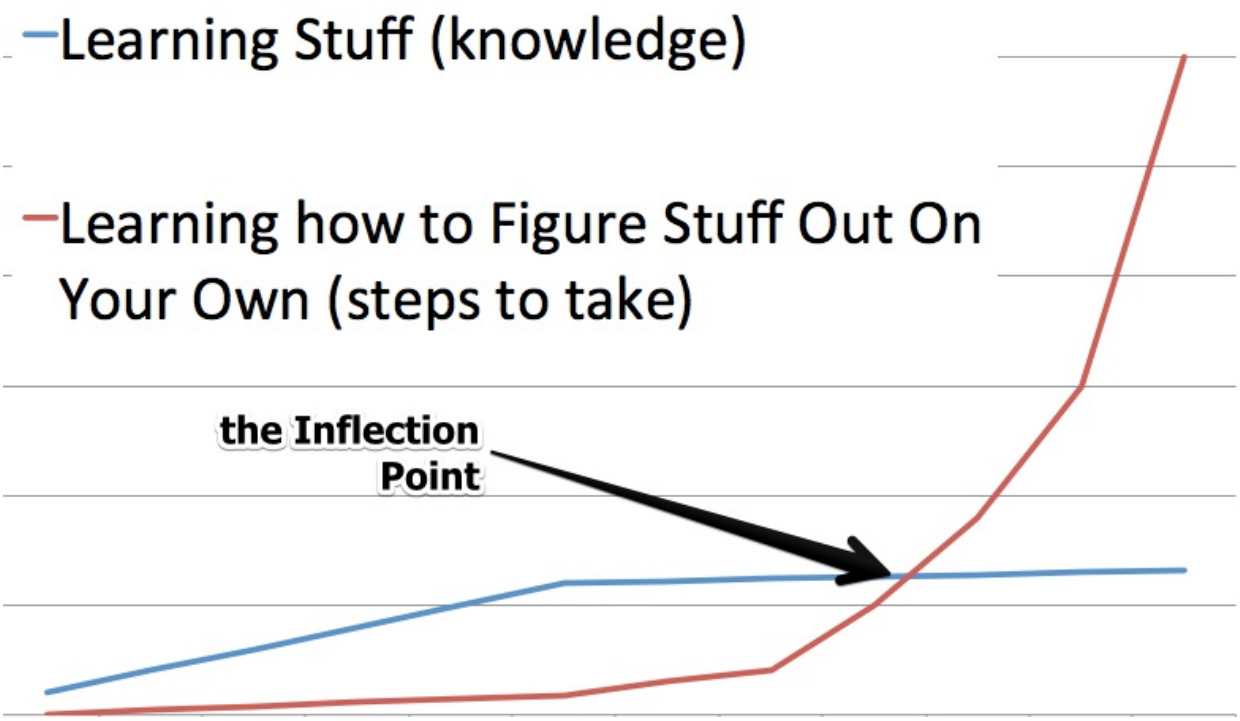
The Key To Accelerating Your Coding Skills

When you learn to code, there is a moment when everything begins to change. At Firehose, we like to call this the inflection point of coding. After this phase, the way you operate as a developer will be dramatically different. Building up to the inflection point is the process of becoming self-sufficient in programming, to the point where you no longer need any hand-holding. It can be a frustrating experience, but once it's behind you, it is incredibly empowering.



At Firehose, our goal isn't just to teach you Ruby, how to build web applications, or how to write tests. Although we do teach these skills and more, **our primary goal is to accelerate students past the inflection point so they gain the ability to solve any problem they encounter.** We believe that being able to problem solve on your own is an invaluable skill, and this method of teaching will take you much further than simply learning how to build a set of apps

The Tutorial Phase (3-8 weeks of serious coding)

When you start out learning to code, there's a lot of information that you don't know yet. This information is called domain-specific knowledge. Examples include: knowing how to write a loop in ruby or how to extract something from a database using Ruby on Rails. Domain-specific knowledge encompasses the protocols unique to a certain programming environment.

The first step to becoming a self-sufficient developer is learning how to do specific tasks. Once you master certain tasks, the broad strokes of how the pieces fit together will start to become apparent. Over time, you'll begin to recognize patterns and eventually, the things that initially seemed confusing and foreign will become second nature.

For students starting out, the most important skill to acquire is attention to detail.

Paying close attention to detail is important when going through materials like documentation or tutorials. Even the most minor typos and misspellings will result in error messages or bugs. Seeing error messages is a frustrating experience at first, but it's a crucial step in the learning process. Dealing with error messages and problems in this phase teaches you one of the most important skills of programming within a safe environment: being detail-oriented.

Debugging error messages is incredibly important. The fact of the matter is, error messages are just a part of programming: they are seen by inexperienced and very experienced developers alike. The only difference is, the more experience you have dealing with error messages, the less time you'll need to spend trying to fix them. Here's why:

- Over time, you will learn **how to read error messages** and extract the relevant details of the problem quickly. The first time you see an error message, it will take you a while to decode what it actually means. But after you've seen hundreds of error messages (and you will see hundreds!), you will be able to pinpoint the problem's location and the relevant details you need in order to fix it.

- You should **learn from each error message that you resolve.** Don't just fix the error and be done with it; understand what is wrong with the code you're fixing. By learning from each of your errors, the next time you make the same mistake, you'll be able to fix it much faster.

- Initially, you will probably ask for help on each error message you see. Over time, you'll learn to ask for help less frequently by double-checking your code and **conducting smart Google searches.**

In the tutorial phase, you will follow instruction. At first, you'll find it challenging to follow instructions and error messages will happen frequently. Over time, you'll develop the skill to debug errors and pay better attention to small details, and you'll be able to make progress much quicker. As you're wrapping up the tutorial phase, you'll notice you're able to write code at a much more rapid pace.

At this point, some people feel confident– like they're ready to ditch the training wheels and start building things without structured guidance– and will happily dive into the deep end. Other students will reach for more tutorials, trying to obtain more domain-specific knowledge in search of a "full understanding." Unfortunately, tutorials will only take you so far, and true confidence isn't found in tutorials or guides. True confidence comes from struggling through a problem you have no idea how to solve, and discovering a solution on your own.

The dirty little secret of programming is…

You will never know everything you need to know to solve all your problems. Going into the journey, you probably envisioned yourself eventually learning everything you need to learn, and then being squared away. This moment will never happen.

Programming is a life-long learning experience. Experienced software engineers seek to find solutions to problems they haven't solved yet because it gives them the opportunity to learn more. If you find yourself waiting for the moment when you finally feel like you know everything there is to know about coding, know this: the day you're waiting for will never come. And that is a wonderful thing.

You will be ready to jump into the next phase of your journey when:

- You've seen enough error messages that **they no longer startle you.** Instead, you know how to decipher what they mean and where to look for the problems in your code.

- **You're a pro at Googling for solutions.** When you're working to add a feature or see a confusing error message, you know what to search for to find the information you need.

- You're able to **reference code you've written in other parts of your application and follow patterns** within them, rather than always seeking out step-by-step instructions.

The Inflection Point (2-4 weeks with the right mentality)

The inflection point stage is one of the most frustrating stages of learning to code, but in many ways, it's the only stage that matters. It's the point when you phase out of using tutorials and begin solving problems for which no one has lined up a solution for you.

At some points, you will feel like you aren't ready to tackle this phase and like you want to return to building something with an outline of exactly what to do. Don't fall prey to this mentality. The reason you'll feel frustrated is:

During the inflection phase, you will be coding 10-20 times SLOWER than in the previous phase.

You may start questioning yourself and wondering if you are actually capable of becoming a programmer. Feelings of insecurity and doubt are common in this stage.

Despite the fact that you'll feel like you're learning and accomplishing things at a much slower rate, in reality, you are achieving the things that matter the most. While your domain-specific knowledge is screeching to a putter, everything you're learning will be about procedural knowledge.

Procedural knowledge is the ability to teach yourself what you don't know along the way. When you need to implement a new feature, what type of Google search should you do? At this point in time, you'll feel like you're "in the dark" when it comes to many of the things you want to accomplish. Learning how to find the light on your own is critical because you can never know everything there is to know, so you need to be able to teach yourself how to solve the problem at hand.

Most people do not realize that in order to learn to code, you need to learn both domain-specific and procedural knowledge.

For the rest of your life, go outside your limits every single day

Some software engineers stay inside their comfort zone once they find their footing. These types of programmers are known as maintenance programmers– not something you should strive to be. Instead, you should strive to go outside your limits every single day. The most common reason programmers quit their jobs is because "it's not challenging anymore since I've solved all the interesting problems."

Rather than trying to pull coding projects into your comfort zone, you should be seeking out problems that are outside your current skill set. This is the only way to build on and expand your skills.

In the words of a Firehose student upon passing his inflection point:

> I still feel like I'm in the deep end! I'm just getting more comfortable knowing that's where I've got to be!

In web development, there are actually two inflection points that will come together.

The web development inflection point is the point when you become capable of building any database-driven application that you want. This means being able to build a web application with many pages that stores and retrieves information from a simple database. Web developers call this: "mastering CRUD." At this phase, you should also be able to integrate with any 3rd party library (a ruby gem for example) simply by following the documentation provided on GitHub or a blog post.

The algorithm and data structures inflection point is a less superficial inflection point, but it is actually more important. Someone who has conquered this point will have mastered the programming language they're working in, in addition to mastering the fundamentals of programming and having a depth of knowledge for solving complex coding challenges.

People who have conquered the algorithm and data structures inflection point will be able to:

- Write sorting algorithms
- Implement and reverse linked lists
- Understand and write programs leveraging stacks, queues, and trees
- Write computer programs using recursive or iterative solutions

In short, once you pass this inflection point, you will have mastered data manipulation and will understand the performance implications of your code decisions. Traditional computer science degrees focus exclusively on getting students past the algorithm and data structures inflection point. Many universities teach this with programming languages that are generally not used in the industry, like Scheme, Racket, or LISP.

In most technical interviews, the interviewer will assume you've passed the web development inflection point, given that's easier to do, and focus their questions on evaluating your skill in algorithms and data structures. These questions will generally focus on the topics we mentioned above: sorting algorithms, reversing linked lists, and using stacks, queues, and trees.

Once a developer has passed both the web development inflection point and the algorithm and data structures inflection point, they hold the keys to the kingdom.

These developers will be able to solve challenges that intersect the two: complex algorithms that need to be built in the context of advanced web applications. This is at the heart of what professional web developers do every single day.

Consequences of the Inflection Point

The biggest consequence of the inflection point will sound a bit counterintuitive when you first hear it. Take a deep breath in: **When learning to code, domain-specific knowledge doesn't matter in the grand scheme of things.**

Yup. I'm not joking– it really doesn't matter that much at all. Once you pass the inflection point, these concepts will fluidly translate with just a week or two of tutorials, maybe even days!

What ultimately really matters is:

- You have a **solid grasp on a web development framework**
- You have a **solid grasp on writing algorithmically complex code** in any programming language

Hiring managers want developers with solid web development and algorithm skills.

While I was working at PayPal, my team hired a Senior Rails Developer who had no experience in Rails– he had been coding a lot in Python, LISP, and Perl. Within a couple of days, he was already making a big impact. And within weeks: a huge impact. He quickly rose to become the technical team lead and was one of the best hiring decisions I've ever been involved in.

Don't sweat the stack. Many people will say things like, "AngularJS is hot these days," "JavaScript is on the rise," or "the latest fad is…" My response to that is: "so?" When you're learning how to program, your singular goal should be to find the inflection point and annihilate it. Once you do, learning that new, sexy fad won't be a difficult task at all.

Become self-reliant. Having the ability to learn new coding skills without structured guidance means you no longer need to wait for anyone to help you out. This means that for the majority of what you need to learn, you can simply search the internet and read the various material on what you need to know.

This doesn't mean you immediately "know" everything, but just that everything is now "figure-out-able," so in essence, you are unstoppable.

The Skills You Will Develop During the Inflection Point

As a software developer, the best reference material is similar code that you have already written. When you fully understand the code you've written, you don't need to commit all the details to memory. This means that **the first question you should ask yourself when building a new feature is: "Have I built something similar before?"** If the answer is yes, revisit the code and walk through the code line-by-line in your head. Re-explain to yourself what it's doing and ask yourself, "could I use the same approach now?"

Videos suck at explaining domain-specific details because they take so darned long to watch. Say you want to integrate with the Google Maps API. Once you've experienced doing so once, it can take less than a minute to open the code up in GitHub, copy the code, and paste it into a new project. Videos, on the other hand, can often take 10-30 minutes to re-watch.

Strategies for Passing the Inflection Point as Efficiently as Possible

Because passing the inflection point is the most important part of learning to code, you should set yourself up to make the process as smooth as possible. This means you should start preparing while you're in the tutorial phase and maintain the right mindset during this period of time.

During the tutorial phase, take breaks from going over structured material and give yourself challenge problems along the way.

- For every handful of lessons, **try to do something that is outside the scope of the tutorial** you're following. If the tutorials you're going through provide "challenges" or "self-directed" pieces, do all of them. Solving unguided challenges will give you the important experience of doing things without structured guidance.

- **Try to use tutorials as little as possible.** At Firehose, we often walk students through how to integrate certain gems or do things using the provided documentation. Rather than simply following the instructions explained in tutorials that are geared towards people who are just starting out, many students will follow the documentation and use the tutorials as a back-up. **Note that documentation will treat you like a developer who has passed the inflection point.** Getting comfortable reading and following documentation on GitHub will give you a leg up when you're on your own.

- Focus on the essentials and use repetition. Learn how to do common things like spinning-up an application from scratch, pushing a new app to GitHub and Heroku, and building a database migration early on.

Pushing through the inflection point can be challenging. Here are some pointers to get you through it:

- Understand that **this is a difficult process and go easy on yourself.** Also, set realistic expectations. You can't compare your "superman"-level speed of going through tutorials to your "snail"-speed of learning things on your own. Keep in mind that you're learning plenty, but at this phase, you're learning a brand new skill of figuring new things out on your own.

- If you're struggling with self-confidence, know that what you're feeling is completely normal. Keep working. If you continue to struggle, try talking to someone who has recently passed the inflection point. They will be able to relate to the position you're in and will assure you that what you're experiencing is only temporary. **Work consistently, but don't overwork yourself.** At this phase of the game, know that you can only be productive for around 6 hours a day at the most. Working in an exhausted state will only prolong the time you spend building up to the inflection point.

The best way to gain confidence at this stage is to power through any doubts you have. Your emotions may start to feel like a roller coaster. At times, you'll feel like you're on fire, but after 15 hours of struggling on the same problem, it's very common to feel the polar opposite.

It can be frustrating to have no idea if something will take you 5 minutes or 5 hours, but every time you power through and successfully implement a new feature, the rush of confidence will be everything you need. After solving a handful of hard problems without any help, you'll be addicted to the feeling of building things outside your comfort zone.

How to know when you've passed the inflection point

The final stage of the inflection point process is acceptance. Acceptance that software development is a process of continuous learning. Acceptance that the feeling that you've successfully learned everything just means you should start thinking about solving more complicated problems.

Have you experienced the inflection point yet? Share this post and start a conversation with your friends– you would be surprised how many people have reached and surpassed this moment.

文章链接