Read this article and write down one thing from your life that exemplifies something Cecily talks about. Even if you're not a "coder," if you've ever tried to do something brand new that is creative and difficult, you can probably recall something that rings true for Cecily too.

Cecily CarverNov 22, 2013
https://medium.freecodecamp.com/things-i-wish-someone-had-told-me-when-i-was-learning-how-to-code-565fc9dcb329#.x2m6k3q8t

Things I Wish Someone Had Told Me When I Was Learning How to Code

And what I've learned from teaching others

Before you learn to code, think about what you want to code

Knowing how to code is mostly about building things, and the path is a lot clearer when you have a sense of the end goal. If your goal is "learn to code," without a clear idea of the kinds of programs you will write and how they will make your life better, you will probably find it a frustrating exercise.

I'm a little ashamed to admit that part of my motivation for studying computer science was that I wanted to prove I was smart, and I wanted to be able to get Smart Person jobs. I also liked thinking about math and theory (this book blew my mind at an impressionable age) and the program was a good fit. It wasn't enough to sustain me for long, though, until I found ways to connect technology to the things I really loved, like music and literature.

So, what do you want to code? Websites? Games? iPhone apps? A startup that makes you rich? Interactive art? Do you want to be able to impress your boss or automate a tedious task so you can spend more time looking at otter pictures? Perhaps you simply want to be more employable, add a buzzword to your resume, or fulfill the requirements of your educational program. All of these are worthy goals. Make sure you know which one is yours, and study accordingly.

There's nothing mystical about it

Coding is a skill like any other. Like language learning, there's grammar and vocabulary to acquire. Like math, there are processes to work through specific types of problems. Like all kinds of craftsmanship and art-making, there are techniques and tools and best practices that people have developed over time, specialized to different tasks, that you're free to use or modify or discard.

This guy (a very smart guy! Whose other writings I enjoy and frequently agree with!) posits that there is a bright line between people with the True Mind of a Programmer and everyone else, who are lacking the intellectual capacity needed to succeed in the field. That bright line consists, according to him, of pointers and recursion (there are primers here and here for the curious).

I learned about pointers and recursion in school, and when I understood them, it was a delightful jolt to my brain—the kind of intellectual pleasure that made me want to study computer science in the first place. But, outside of classroom exercises, the number of times I've had to be familiar with either concept to get things done has been relatively small. And when helping others learn, over and over again, I've watched people complete interesting and rewarding projects without knowing anything about either one.

There's no point in being intimidated or wondering if you're Smart Enough. Sure, the more complex and esoteric your task, the higher the level of mastery you will need to complete it. But this is true in absolutely every other field. Unless you're planning to make your living entirely by your code, chances are you don't have to be a recursion-understanding genius to make the thing you want to make.

It never works the first time

And probably won't the second or third time

When you first start learning to code, you'll very quickly run up against this particular experience: you think you've set up everything the way you're supposed to, you've checked and re-checked it, and it still. doesn't. work. You don't have a clue where to begin trying to fix it, and the error message (if you're lucky enough to have one at all) might as well say "fuck you." You might be tempted to give up at this point, thinking that you'll never figure it out, that you're not cut out for this. I had that feeling the first time I tried to write a program in C++, ran it, and got only the words "segmentation fault" for my trouble.

But this experience is so common for programmers of all skill levels that it says absolutely nothing about your intelligence, tech-savviness, or suitability for the coding life. It will happen to you as a beginner, but it will also happen to you as an experienced programmer. The main difference will be in how you respond to it.

I've found that a big difference between new coders and experienced coders is faith: faith that things are going wrong for a logical and discoverable reason, faith that problems are fixable, faith that there is a way to accomplish the goal. The path from "not working" to "working" might not be obvious, but with patience you can usually find it.

Someone will always tell you you're doing it wrong

Braces should go on the next line. Braces should go on the same line. Use tabs to indent. But tabs are evil. You should use stored procedures, but actually you shouldn't use them. You should always comment your code. But good code doesn't need comments.

There are almost always many different approaches to a particular problem, with no single "right way." A lot of programmers get very good at advocating for their preferred way, but that doesn't mean it's the One True Path. Going head-to-head with people telling me I was Wrong, and trying to figure out if they were right, was one of the more stressful aspects of my early career.

If you're coding in a team with other people, someone will almost certainly take issue with something that you're doing. Sometimes they'll be absolutely correct, and it's always worth investigating to see whether you are, in fact, Doing It Wrong. But sometimes they will be full of shit, or re-enacting an ancient and meaningless dispute where it would be best to just follow a style guide and forget about it.

On the other hand, if you're the kind of person who enjoys ancient but meaningless disputes (grammar nerds, I'm looking at you), you've come to the right place.

Someone will always tell you you're not a real coder

HTML isn't real coding. If you don't use vi, you're not really serious. Real programmers know C. Real coders don't do Windows. Some people will never be able to learn it. You shouldn't learn to code. You're not a computer programmer (but I am).

"Coding" means a lot of different things to a lot of different people, and it looks different now from how it used to. And, funnily enough, the tools and packages and frameworks that make it faster and easier for newcomers or even trained developers to build things are most likely to be tarred with the "not for REAL coders" brush. (See: "Return of the Real Programmer")

Behind all this is the fear that if "anyone" can call themselves a programmer, the title will become meaningless. But I think that this gatekeeping is destructive.

Use the tools that make it easiest to build the things you want to build. If that means your game was made in Stencyl or GameMaker rather than written from scratch, that's fine. If your first foray into coding is HTML or Excel macros, that's fine. Work with something you feel you can stick with.

As you get more comfortable, you'll naturally start to find those tools limiting rather than helpful and look for more powerful ones. But most of the time, few people will ever even look at your code or even ask what you used—It's what you make with it that counts.

Worrying about "geek cred" will slowly kill you

See above. I used to worry a lot, especially in school, about whether I was identifying myself as "not a real geek" (and therefore less worthy of inclusion in tech communities) through my clothing, my presentation, my choice of reading material and even my software customization choices. It was a terrible waste of energy and I became a lot more functional after I made the decision to let it all go.

You need to internalize this: your ability to get good at coding has *nothing* to do with how well you fit into the various geek subcultures. This goes double if you know deep down that you'll never quite fit. The energy you spend proving yourself should be going into making things instead. And, if you're an indisputable geek with cred leaking from your eye sockets, keep this in mind for when you're evaluating someone else's cred level. It may not mean what you think it does.

Sticking with it is more important than the method

There's no shortage of articles about the "right" or "best" way to learn how to code, and there are lots of potential approaches. You can learn the concepts from a book or by completing interactive exercises or by debugging things that others have written. And, of course, there are lots of languages you might choose as your first to learn, with advocates for each.

A common complaint with "teach yourself to code" programs and workshops is that you'll breeze happily through the beginner material and then hit a steep curve where things get more difficult very quickly. You know how to print some lines of text on a page but have no idea where to start working on a "real," useful project. You might feel like you were just following directions without really understanding, and blame the learning materials.

When you get to this stage, most of the tutorials and online resources available to you are much less useful because they assume you're already an experienced and comfortable programmer. The difficulty is further compounded by the fact that "you don't know what you don't know." Even trying to figure out what to learn next is a puzzle in itself.

You'll hit this wall no matter what "learn to code" program you follow, and the only way to get past it is to persevere. This means you keep trying new things, learning more information, and figuring out, piece by piece, how to build your project. You're a lot more likely to find success in the end if you have a clear idea of why you're learning to code in the first place.

If you keep putting bricks on top of each other, it might take a long time but eventually you'll have a wall. This is where that faith I mentioned earlier comes in handy. If you believe that with time and patience you can figure the whole coding thing out, in time you almost certainly will.