

Lately, I've been learning Python. It's something I've wanted to explore for a while, and I've set a goal for myself: by the end of next year, I want to be hireable as a data engineer. Not because I plan to switch careers, but because I believe this skillset will become increasingly important in the future of software engineering.

What I've realized during this process is just how overwhelming it can be to learn something entirely new.

It's not just about picking up the technical skills; it's about navigating a sea of advice, opinions and conflicting information.

I have a whole new perspective on what many of you go through when starting to learn coding.

It's hard to know where to start, who to listen to, and how to make real progress. Let me share what I've learned about finding good advice and reverse-engineering your path to success as a software developer.

Part 1: How to Find Good Advice and Reverse-Engineer Your Path

The Problem with Too Much Advice

I went to the University of YouTube, Reddit and X to find out what data engineers do, what to learn and what to build.

Some said to focus on Python. A few said R. Another said Rust.

I either need to be an expert in linear algebra or barely know it. Data engineering is either the future of software or is dying and I should avoid it at all costs.

Sound familiar?

One person says you need to master a dozen frameworks. Another says it doesn't matter what you learn.

The truth is a lot of people giving advice don't actually know what they're talking about. Some have limited experience or only worked in a specific niche, and their advice may not apply to your situation.

How to Vet Your Sources

Before you take someone's advice, ask yourself a few questions:

What is their experience?

Have they worked at companies similar to the ones you want to work at?

Have they built the kind of projects you want to build?

Do they have breadth and depth?

Someone who's worked across multiple industries or roles might have more practical insights than someone who's only worked at Google or a single company.

What's their track record?

Have they helped others achieve what you're trying to achieve?

For the data curriculum I'm introducing at my online coding school, Parsity, I looked for someone who had both practical experience and a history of teaching. This person helped me map out a clear curriculum based on real-world skills.

Reverse-Engineering Your Path

The best way to learn anything complex – whether it's coding or data engineering – is to start with the end goal in mind.

What kind of job do you want?

What projects do you want to be able to build?

Here's how I approach this:

Find a North Star Project: Choose a project that represents the skills you need to master. For software developers, this might be a full-stack app with authentication, a database, and a responsive frontend.

Break It Down: Identify the skills and technologies required to build that project.

Learn in Layers: Start with the fundamentals, like JavaScript and basic databases, before moving on to more advanced topics like frameworks and authentication.

Validate Your Plan: Run your roadmap by someone who's done it before.

Now, let's do the same for becoming a hire-able software developer.

Step 1: Just Start with JavaScript

Why JavaScript? It's the most versatile and widely used programming language in the world. Whether you're working on the frontend, backend, or even mobile apps, JavaScript will come into play.

Here's how to approach it:

Start at [Codecademy.com](https://www.freecodecamp.org) – it's the easiest place to begin.

Learn Arrays and Objects: Learn how to iterate through arrays, and access object properties. Most data on the web is in this format.

Work with APIs: Practice fetching and displaying data from public APIs.

Build a Basic App: Create a simple form that submits data and displays a response. For example, a "fake" social media app where a user can post a comment.

Step 2: Learn React and Next.js

Once you've got the basics of JavaScript down, it's time to move on to frameworks. React is the industry standard for building user interfaces, and Next.js adds backend functionality which allows you to go full-stack faster (basically you can create the website and handle storing data in the cloud).

Here's what to do:

Start by re-creating your JavaScript app using React.

Learn enough Next.js to handle backend tasks like APIs to handle form submissions. Their docs are solid, you don't need another tutorial for this.

Build something small but functional, like a to-do list or a blog with dynamic routes (multiple pages).

Build in some form validation so a user can't post a blog or a todo without certain fields being filled.

Step 3: Get Comfortable with Databases

Every app needs a way to store and manage data. That's where databases come in.

Start with SQL – it's used almost everywhere. Learn how to:

Write SELECT statements to query data.

Perform JOINS to combine data from multiple tables.

Use aggregation functions like COUNT and SUM.

If you're interested in NoSQL, MongoDB is a good alternative, but SQL should be your priority. It's super easy to use with tools like NeonDB and Prisma or Drizzle.

I wrote a guide on getting started with SQL here: <https://www.parsity.io/blog/a-hands-on-guide-to-learning-sql>

Step 4: Add Authentication

Authentication is one of the most challenging parts of building an app, but it's essential.

Here's my advice:

Start by building a simple login system yourself. Learn how sessions work and

how to manage whether a user is logged in or not.

Once you understand the basics, switch to a professional solution like Auth0 or Clerk for real-world projects. You don't want to be responsible for handling sensitive data.

Step 5: Deploy Your App

When it's time to show off your work, you'll need to deploy it online. Start with Vercel, which is perfect for Next.js apps, but don't stop there. Learn how to deploy on platforms like AWS to understand more about cloud infrastructure.

Step 6: Build Something Meaningful

Forget a portfolio. Build something that:

Solves a real problem.

Is full-stack and secure.

Is accessible online for anyone to view.

A mentee at Parsity built an inventory tracker for his family's business. It wasn't just a portfolio piece; it was a tool they used in their daily operations. Projects like this stand out because they show you can deliver value.

Step 7: Gain Experience

If you can't land a paid role right away, offer to build something for free.

Look for entrepreneurs or small businesses that need help. This gives you real-world experience and something to showcase on your résumé.

There are Reddit communities of entrepreneurs and local businesses that can use your help. Your local church, your uncle's plumbing business and your kids' school could all use a better website.

Why This Works

When I started, I had no idea what I was doing. I made mistakes, got stuck, and spent way too much time Googling solutions, reading random books and taking the advice of every employed coder as gospel.

By focusing on real projects and building practical skills through some free work, I turned myself into a hire-able developer.

If you follow this roadmap, you'll not only learn to code – you'll learn how to solve problems, build valuable tools, and showcase your abilities in a way that gets you hired.