WILL WILDER

Bulletproof Guide to Becoming a PAID Front-End Developer (No Experience Required)

How to Change Careers & Land Your First Job as a Front-End Developer

First edition

This book was professionally typeset on Reedsy.
Find out more at reedsy.com

Contents

III KNOWLEDGE

Preface

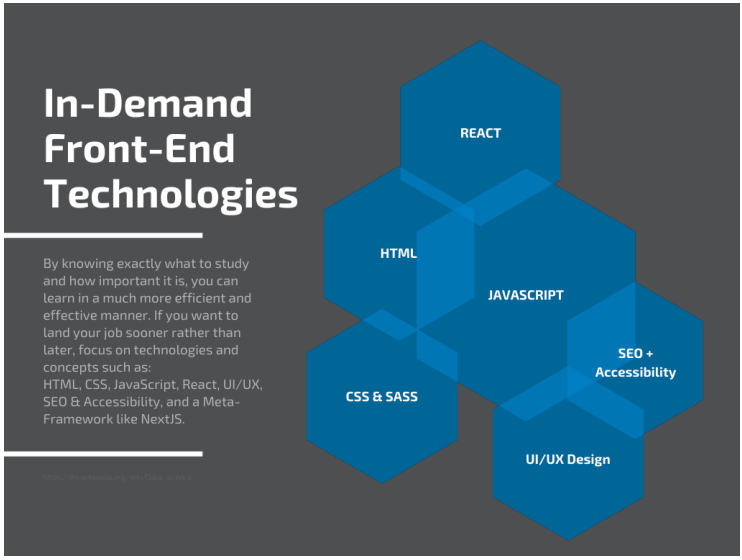
You're here because you want to learn how to get a job as a Front-End Developer.

You are already aware of the fact that learning how to code is one of the most valuable assets one can have in the world today, and you either:

- Want to **change careers and land your first job in tech** as a front-end dev.
- Have seen how **freelancing can be a lucrative way to earn cash and stay independent**.
- Already have some experience and want to **become a better developer** by ensuring you haven't missed anything on your learning journey.

If any of the above describes you, you'll *love* this guide.

It will help you **learn the RIGHT skills** to **get paid as a front-end developer**.



And guide you through interviewing techniques so you can land that first job with ease.

I have poured hundreds of hours into creating this valuable piece of content because, frankly, I needed it when first starting.

In this guide, you will find every strategy, mentality, and tidbit of information needed to learn front-end development and start getting paid for your expertise.

We will go over the "big picture" view of the journey ahead, all the way down to detailed real-world examples and step-by-step walkthroughs.

There is *nothing* like this guide out there, and **I am so excited to share it with you**!

Welcome Aboard

Front-End Developer Jobs are HOT. If you're looking for a career but don't have any relevant experience, **this is the perfect resource for you.**

In these pages and worksheets, you will find the exact strategies, mentalities, and knowledge you need to become a Front-End Developer in the most efficient and fun way possible.

By reading this guide in its entirety, you will find:

- A **complete, step-by-step system** for learning how to code effectively and get hiring managers talking about YOU!
- The **best-proven tactics and strategies** that established developers have used to get their first jobs and start getting paid for what they do.
- The **most prominent myths and mistakes** you should avoid when learning front-end development.

In the end, I'll also share with you a free 6-month learning checklist, an easy-to-implement study plan, and a secret bonus PDF that you can print out for reference as you continue your journey to **Becoming a PAID Front-End Developer**.

My Software Development Journey

I'm Will from Bulletproof-Code.com, and I teach career changers like you the most *effective* path to becoming a Front-End Developer so you can **do what you love, earn a great living, and change the world** (if you are into that).

I created this INSANELY detailed guide because changing careers and becoming a software developer has been one of the most significant endeavors that allowed me to change my life, enjoy what I do, and provide for those I love at a high level.

I could talk about Front-End Development ALL DAY! That's why it was so great to receive many comments over the past few months asking how I went from working 50-60 hours a week in a manual labor job to launching a lucrative career as a software developer.

Those comments, mixed with the fact that my journey of learning how to code was a nightmare, pushed me to put all my thoughts, experience, and resources into one incredibly detailed guide.

Making the Transition

When I was trying to transition from my exhausting warehouse position, making barely enough to provide for my family, I wished there was a step-by-step guide to becoming a front-end developer.

The sad fact was, there was no such resource.

Don't get me wrong; there are tons of articles, videos, and courses online vying for attention. And that was one of the problems. I had hundreds of bookmarks, no clear path, and a whole lot of learning ahead of me.

There was no one-stop solution. Instead, I would pick up a potential solution from one source only for its information to contradict the next one. Learning to code this way was exhausting. Yet, I learned a lot about what not to do through this process.

> *So, after landing my first job and reaping the o so sweet rewards of becoming a developer, I knew I wanted to give back. I started teaching.*

I taught a 6-month coding Bootcamp at the University of Oregon. From there, I went on back to the Mid-West and led eight separate 6-month cohorts on how to learn front-end development and land that first job.

All this to say, I know what I am talking about when it comes to front-end development and getting a job with no prior experience. **I have mentored and seen HUNDREDS of students just like yourself get jobs in tech** with far better excuses.

I saw students that were over 60 YEARS OLD get their first job! One that was *blind* - imagine that. Students that worked at

McDonald's and never graduated high school. One student could barely turn on a computer; it was entirely new for them.

Those examples are not there to brag, boast, or say that your excuses aren't valid. Instead, I want to let you know, if you follow this guide, stay patient, and put in the work, your first job is not only possible, it is *inevitable*.

What You Can Expect

You are probably wondering how hard it is to get a job as a developer with no experience. You might think that you are perhaps too old or too young. You may feel that failing math means you will fail at programming.

I am here to say, **without a shadow of a doubt, if you follow the plan in this guide, you can 100% get a job as a developer.**

I have seen it over a hundred times now; people from all walks of life wandering around with the same worries you have at this very moment, going on to succeed in landing a job as a developer with no prior experience. **It is life-changing, it is possible, and you can do it**!

As long as you stay focused on the goal, put in the practice daily, and receive relevant feedback regularly, landing your first front-end position is a cinch.

Comprehensive guides like this that provide practical tools, tips, and technologies will melt away most fears you have about whether or not YOU can become a paid software developer.

Information + Action = alleviated worries and doubts. (This guide helps you with both the Information and the Action.)

What You Will Learn

By following the advice on these pages, you will:

- Find out how to **become a PAID programmer in the quickest amount of time** so you can launch a lucrative tech career in the next 6-months
- Learn how to **implement an accessible yet profound study routine** backed by leading-edge science so you can succeed in coding and life
- Get access to **detailed plans, guides, and checklists** that will guide you step-by-step from wherever you are today to learning the most in-demand skills, applying for jobs, acing the interview, and repairing for your first developer position.

DOWNLOAD MY FREE RESOURCES NOW!

Does that sound like a plan?

Well, get ready because that is just the tip of the iceberg. There is a ton of actionable and valuable stuff content and materials ahead!

How to Use This Guide

Note: this is a reasonably lengthy guide with over 16,000 words, but each sentence in this guide has a definite purpose. As a result, this manual is beginner-friendly and easy to work through—no fluff.

I recommend that you read the eBook from start to finish and not skip ahead, as some chapters build on each other. If you want to take some extra time to check out the examples and resources I mention in the manual, that's fine, but make sure that doesn't kill your momentum.

If you need to read this on your Ipad, Kindle, Phone or prefer to have a PDF version of this guide, I got you covered, free of charge. Get the "Bulletproof Guide to Becoming a PAID Front-End Developer (No Experience Required)" PDF!

Once you finish reading all the way through, you'll know exactly what it takes to **become a Front-End Developer,** and you will understand all the crucial parts of the process so you can land that new job faster than you think! :)

Without further ado, let's dive in!

I

STRATEGY

1

Explore the Most Effective Path To Becoming a Programmer

Did you know it takes *10+ months* to land a real developer job for a typical job-seeker?

I don't know about you, but when I had had enough with my previous job, ten months felt like an eternity.

It took me even longer than ten months because I didn't have clarity on where I was going nor a plan of action that would land me in the tech world.

That's why **if you focus on the right goal and have a coherent system and steps to achieving that goal, and you have support and mentors guiding you along the way, you can break that ten into seven or even six months**.

I will not add to the false information and expectations floating around that you will learn how to code and get a $100,000 job in 12 weeks or 90 days. That is just absurd.

What I am offering is a fact-based and expert opinion on what to learn and how to learn to maximize your time so you can get that first job sooner by focusing on the right things at the right times.

NOTE: At the end of this eBook, you will find a free 180-day plan outlining EXACTLY what you need to learn and do each day to launch your programming career in style. Download Your Free 6-Month Plan Now.

What is the Quickest Route to Becoming a PAID Developer?

Because you are reading this guide, you are most likely aware that becoming a Front-End Developer is your best bet to landing your first job if you have no experience.

Yet, there are many paths you can take, so before we highlight the front end, let's outline the other routes you can take to become a programmer.

Some of the more popular content online guides newbies to learn "full-stack" development. Beyond that, there are copious blogs, videos, and courses for application development, back-end engineering, database management, and game dev.

I don't want to stop you from going after your dreams. If you have wanted to create a game and you have sufficient passion for pushing past rather significant obstacles, by all means, I support you.

But, from researching the facts and crunching the numbers, unless you have a CS Degree or are a black hoodie-wearing hacker, **students with no experience have the best chances of getting a Front-End Developer job over any other programming position HANDS DOWN**. *It's not even close.*

The barrier to entry is much much lower. The topics are simpler to grasp, and you can see your progress and gain feedback quicker (which is essential for learning anything). On top of all that, there are so many great and up-to-date resources at your disposal.

Most companies are willing to add someone with no experience to their team if they specialize in the Front-End first and foremost.

So what if you want to move towards a different route eventually? In my opinion, learning the Front-End first is still your best bet. Companies and teams are looking for someone who fits ONE main criterion.

> ***Have they successfully worked on a real development team in a production environment?***

So you see, learning front-end development can take care of that obstacle. You will learn it WAY quicker than the back-end or other programming fields. You will gain invaluable experience working on a team. **You will be getting PAID solid money** while transitioning responsibilities (which is 1,000,000 times easier than transitioning careers).

And you will have a comprehensive understanding of the Front-End that you can bring to your next endeavor. Knowing both sides of an application will put you ahead and give an edge to others who started and siloed in one aspect of the development process.

Slow and Steady.

Bootcamp vs. Self-Taught vs. CompSci Degree

Now that we know our goal, we have to figure out what vehicle we will use to get there. The three main options are going to a Bootcamp, attending a University, or learning on your own.

To better rank these as objectively as possible, we will place these options on a matrix of five critical areas.

The five categories we will walk through are as follows:

1. **EXPENSES**: *How much does it cost?*
2. **EFFICIENCY**: *How quickly can you learn and land a job?*
3. **EASE**: *How difficult is this path?*
4. **ENCOURAGEMENT**: *What support do you have?*
5. **EDGE**: *How Employable are you (over others) by taking this route?*

Using these as metrics will help us determine the pros and cons of each option. But, in the end, the decision is up to you.

EXPENSES

1. **Self-Taught**: The only expenses are the courses you choose to buy.
2. Bootcamp: These can vary widely. There are free Bootcamps, some around $2000, but the majority have prices around the $10k mark. (Word of Caution: I would advise against any income-sharing Bootcamps that are $0 upfront, but they take a portion of each paycheck for a given amount of months or years).
3. *CS Degree*: Colleges and Universities will take a considerable chunk out of your bank account. No secrets here.

EFFICIENCY

1. **Bootcamp**: The curriculum is meticulously designed and iterated upon to help you learn to code and get a job in a given timeframe.
2. Self-Taught: This will take longer than a Bootcamp because, while you can choose a curriculum, finding the most optimal one will be like a needle in a haystack.
3. *CS Degree*: This can take 4+ years to complete.

EASE

1. **Bootcamp**: Once again, while Bootcamps are very intense, they are shorter and more streamlined for this particular goal than the other two options.
2. CS Degree: The path will take four years. You may disagree, but it is much easier to tackle a shorter challenge, even if it is more difficult for the duration.

3. *Self-Taught*: Being a self-taught developer is almost more complicated than the other two on every front. From motivation to staying focused, designing a curriculum, discipline, and getting job offers.

ENCOURAGEMENT

1. **Bootcamp**: If you choose the right Bootcamp, there will be dedicated coaches, support groups, and a community purposely designed to keep you motivated and in the program.
2. CS Degree: You could argue that a University has more support than a coding Bootcamp. I have it as number two because the support is not solely focused on learning front-end development and getting a job pronto.
3. *Self-Taught*: You will have to rely on YouTube or forums. It is challenging to code day in and day out with the doubt and worries hanging over your head. But, alas, it can be done.

EDGE

1. **CS Degree**: There is no way around this one. Resume AI, Receptionists, Hiring Managers will all (subconsciously or consciously) rank those who endured four years of computer science more competent and perceived as more valuable both now and in the future.
2. Bootcamp: In the next few years, Bootcamps will take the number one spot for this. Colleges' authority and prestige are slowly fading, and the importance and prevalence of Employer Projects significantly increase the value of

Bootcamp graduates.

3. *Self-Taught*: There are rare cases where companies admire that you "pulled yourself up by the bootstraps" and showed that you have perseverance. But in most occurrences, not having a certification or record of sticking with a program can be a rather significant hurdle.

	BOOTCAMP	SELF-TAUGHT	CS DEGREE
EXPENSES	2	①	3
EFFICIENCY	①	2	3
EASE	①	3	2
ENCOURAGEMENT	①	3	2
EDGE	2	3	①

NOTE: While Bootcamp did seem to be the best route to take to get your first job as a front-end developer, that doesn't mean all Bootcamps are created equal. There are, without doubt, countless scams & money grabs posing as quality developer Bootcamps. You must use that programmer's discernment to find the one right for you. Generally, anything less than three months is almost definitely overpromising results. And, if you choose a Bootcamp costing over $10k, ensure they have proven results verified by outside sources.

2

Bulletproof 6 Month Plan to Learning Front-End Development

By following this **Bulletproof 6-Month plan**, you will learn Front-End Development from scratch and land your first job in under six months.

Six months is a ballpark estimate, and no results are guaranteed. But, if you complete everything in this guide and continue building projects and applying to jobs, it shouldn't be too long before everything falls into place.


Front-End Developer
6-MONTH PLAN
MONTH 1
Set up, Structure & Learn Web Fundamentals
Download and set up all tools and accounts. Dive into the web and begin learning HTML & CSS.
MONTH 2
Responsive Design Styling & SEO + Accessibility Primers
Learn CSS, Sass, mobile-friendly websites, and SEO + accessibility techniques.
MONTH 3
Functionality and Building Applications
Jump into the JavaScript world and begin to build more and more complex applications.
MONTH 4
JavaScript & CSS Framework Fundamentals
Start learning a JS Framework such as React. Then, pick a popular CSS Framework like TailwindCSS & begin creating a capstone project.
MONTH 5
Application and Digital Presence Preparation
Design an alluring resume, create an application strategy and tracking system, and practice JS Algorithms.
MONTH 6
Execute Your Plan & Continue Building Skill
Keep building projects, practicing technical and behavioral interviewing abilities, and sending out your resume.
BULLETPROOF CODE

At the end of this eBook is a **FREE Downloadable PDF Checklist** containing the 6-Month Plan for Learning Front-End Development.

MONTH 1 - Structure

GOAL #1: Setup Development Environment & Implement Version Control

- Install NodeJS, GitBash, Chrome & Visual Studio Code
- Create a GitHub Account
- Download VS Code Extensions & Learn Common Shortcuts
- Practice Basic Terminal Commands
- BONUS: Create a WP or HashNode Blog to Document Your Journey

GOAL #2: Comprehend How the World-Wide-Web Works & Browser Dev Tools

- Study the Internet Basics & How the Web Works
- Understand the Full Development Cycle
- Be Able to Articulate What A Front-End Developers is Responsible For Doing
- Experiment with the Chrome Developer Tools

GOAL #3: Build Websites Using Semantic HTML

- Define HTML and Why it is Used
- Create Your First HTML File
- Examine HTML Elements & Attributes
- Implement Lists, Links, Forms, Images & Tables
- Grasp Semantic HTML and Why it is Important

GOAL #4: Style Applications Using the Fundamentals of CSS

- Grasp How CSS Works & Explain its "Cascading" Effects
- Examine CSS Syntax & Basic Selectors
- Comprehend the Box Model
- Practice Simple Positioning Techniques
- Research Typography, Color, and Background Styling Methods

MONTH 2 GOALS - Style

GOAL #1: Learn Advanced CSS Concepts & Sass

- Compare CSS Naming Conventions & Understand the Idea Behind BEM
- Grasp Relative Sizing Units (rem, em, vh, vw, etc.)
- Learn CSS Pseudo-Selectors & Pseudo-Elements
- Practice Using Transitions & Animations
- Learn, Implement & Apply SASS

GOAL #2: Build Mobile-Friendly Responsive Websites

- Realize the Benefits of Mobile-First Design
- Practice Media-Queries
- Master CSS Layout By Utilizing Flexbox
- Investigate CSS Grid for More Advanced Layout Control

GOAL #3: Optimize Websites for SEO & Accessibility

- Learn Significant Meta Tags
- Understand JavaScript Minification Strategies
- Utilize Advanced Image Optimization Techniques
- Comprehend Google Lighthouse Metrics
- Setup Analytics on Websites
- Investigate Adding Sitemaps, Manifests & Favicons
- Study & Implement Vital Accessibility Features

Supplementary Content: Learn Advanced Animations OR UI/UX w/ Figma

Milestone Project - Create Your Portfolio Website

MONTH 3 GOALS - Functionality

GOAL #1: Aquire JavaScript Fundamentals

- Dive into the JavaScript Syntax Documentation
- Practice Writing Loops & Conditional Statements
- Understand How Functions Work
- Compare Object-Oriented Programming vs. Functional Programming

GOAL #2: Improve Knowledge of JavaScript (Data Structures, Built-In Methods & More)

- Comprehend JavaScript Data Types & Their Use-Cases
- Master the Essential JS Built-In Array Methods
- Investigate Different JS Design Patterns
- Create a Comprehensive Mental Model of How JavaScript Works
- Explore Asynchronous Coding & Why it is Important
- Adventure Into the World of API's & HTTP Requests

GOAL #3: Build Functional Applications Using JavaScript

- Create a Contact Form
- Code an Image Gallery
- Build a Functional JS Clock

Supplementary Content: Typescript

Milestone Project - Content Gallery w/ Filterable Results OR Note-Taking App

MONTH 4 GOALS - Frameworks

GOAL #1: Grasp the Core Concepts of React

- Learn JSX and React Rendering
- Comprehend Components & Their Lifecycle
- Practice Passing Props
- Learn About Lists & Keys
- Analyze Built-In React Hooks

GOAL #2: Dive Deeper into React & Application Design

- Study the Differing Approaches to React Forms
- Compare Composition vs. Inheritance
- Learn to "Think in React"
- Master Using React Router
- Compare State Management Solutions (Context, Local State, Redux, etc.)
- Create Your Own React Hooks

GOAL #3: Investigate NextJS & TailwindCSS

- Compare SSR vs. CSR
- Understand When You Would NextJS
- Read NextJS Documentation
- Practice the NextJS Fundamentals
- Learn about TailwindCSS and See if it is Right For You

- Build a Blog Website Using NextJS & TailwindCSS

Supplementary Content: Software Testing or PWA's

Milestone Project - E-commerce App or Course Platform or Job Board App

MONTH 5 GOALS - Preparation

GOAL #1: Design an Alluring Resume

- Learn What Information is Essential On a Resume
- Analyze Ways You Can Showcase Your Abilities (Even w/ No Relevant Experience)
- Create or Purchase a Resume Template
- Build Out Your Resume & Receive Feedback

GOAL #2: Create an Application Strategy

- Compare Application Strategies (Mass Mail vs. Personal Approach)
- Realize What Metrics to Track
- Create an Application Plan & Tracking System

GOAL #3: Practice JavaScript Algorithms (Job-Ready JavaScript)

- Study the Most Common Interview Questions
- Learn Algorithms & DataStructure Fundamentals
- Read or Watch JavaScript Algorithm Solutions
- Practice JavaScript Coding Challenges

Milestone Project - Application Tracking Software or Interview Preparation App/Blog

MONTH 6 GOALS - Execution

GOAL #1: Continue Building Projects, Practicing Algorithms & Networking

- Code Every Day
- Practice Algorithms in JavaScript Every Day
- Continue Networking & Reaching Out on LinkedIn Every Day
- Join Meetups, Talk with Others, and Build Relationships

GOAL #2: Drill Down Behavioral Interview Delivery

- Look Up Most Common Behavioral Interview Questions
- Create Responses that Showcase Your Unique Personality & Abilities
- Practice Your Answers Infront of Others

GOAL #3: Ace the Coding Interviews

- Master Relaxation Techniques During Interviews
- Learn Subtle Tricks You Can Employ During Interviews
- Enroll In An Interview Course or Program
- Write Follow-Up Messages For Each Interview

GOAL #4: Prepare for Your First Front-End Developer Job

- Review Your GitHub Knowledge
- Study Their Tech Stack
- Celebrate Your Amazing Journey

Milestone Project - Capstone Project of your choosing

3

Easy-to-Implement Study Strategy Proven to Enhance Learning, Memory & Recall

I want you to succeed. And I know you will.

To aid in that process, I have cultivated the most relevant and recent peer-reviewed Neuroscience research (done by people much more intelligent than I).

There are thousands of seemingly good pieces of advice on the web, but after reviewing the studies, most of them turn out to be wives-tales or over-hyped sales pitches. So drop your misconceptions, and let's dive into what is working in this day and age.

NOTE: At the end of this eBook, you can download a free printable version of this schedule (along with many other checklists and resources).

Setting the Stage for Study Success

Optimizing Your Environment

One of the most incredible things you can do for your well-being and productivity is creating and maintaining an organized and distraction-free workplace.

Too many times to count, my students have heeded this advice reluctantly, only to come back and say this was **THE MOST IMPORTANT change they made that boosted their output & motivation to code.**

To find what "distraction-free" means to you, a bit of experimentation is required. However, the general guideline is clean, not too loud, and everything is easy to find.

You can tell when your space is clean and organized when you sit down for a study session, and there is nothing around that is vying for your attention. You don't have to put things away and "set up" before getting started. Instead, your workspace should be *streamlined*.

You don't need to take this to extremes, as another tip for improving your learning is changing your scenery and studying in new locations every so often.

The mechanisms behind why changing learning locations works have to do with the way our brain reacts to novelty. When we encounter a new situation, our brain responds by increasing neuroplasticity and information processing power.

The key is to feel "*in the zone*" when coding or learning. Your house-mates should know that this is your time and respect your study space.

Planning Your Study Vision

> *"A goal without a plan is just a wish."*

To implement a study routine, we must make a plan. We can do a few things to ensure our success at this stage.

The first is implementing a scheduling technique called "time blocking" or "time boxing." This method entails creating chunks of time set on a calendar outlining what type of work you will be doing and when.

Time-blocking differs significantly from the typical "todo" list we all know and. . .love?

Time-blocking has been an essential instrument in my success as a programmer, impacting every aspect of my life.

If creating and scheduling every hour of your day is too intense, a more straightforward program is to wake up an hour earlier and use that time solely for studying. This approach has the unique benefit of being simple and easy to implement.

Plus, everyone will most likely be asleep, giving you excellent focus time away from the typical everyday distractions.

The last system you should test and see if it is right for you is another tool I use every day. At the end of the workday, I take 10 minutes to review what I learned and what went well.

Then, after seeing what I accomplished, I plan and write down my top 3-5 priorities I will get done tomorrow.

I put that sticky note on my work desk or laptop so when I wake up, and I can jump right into the most impactful and meaningful work that will move me most towards my goals.

Planning your study vision helps you be clear on WHAT to do, HOW to do it, WHEN you will do it, and WHY you are doing it. All parts are essential to making a habit stick for the long haul.

Types of Learning You Should Incorporate

There are many methods to choose from for learning how to code. The most valuable mediums you will find are:

- Reading documentation
- Reviewing others code
- Building Projects

I understand that it will feel impossible to create projects on your own at the beginning. But, unfortunately, thinking this way leads many to watch tutorial after tutorial, not realizing they aren't learning very much if anything at all.

While watching videos seems productive, it is probably not the best use of your time. That being said, if you do choose to watch videos or courses, you MUST implement what they are teaching and tinker with the code, making sure you understand WHY everything is working.

The key here is to channel your inner curiosity. Don't just copy line for line without downloading the reason behind the coding decisions.

The reason I, along with so many other professional programmers, hammer in the advice that you will learn best by building projects is, well, because it is true.

Building projects that are a tad out of your comfort zone has a myriad of unique benefits. The most prominent one revolving around learning has to be its baked-in spaced repetition.

When you learn something, going over it 100 times in the same day is way less effective than going over that same piece of info ten times over five days.

By building projects, you will use and reuse the same methods and techniques, further solidifying them in your brain in a way watching videos won't do.

Okay, maybe I was a little too harsh on videos and courses. They are fine (in moderation). But don't say I didn't warn you.

During the Deep Work

Length, Duration & Intensity of Learning Session

Tons of studies on studying are now weighing in on the productivity conversation. Most of them point to a simple statement: "Study less and with greater intensity."

Now, if you aren't studying at all, you should probably study more. But for those of us trying to learn coding by studying 8 hours a day, perhaps we should take a step back and see if spending all that time is truly necessary or if the return on investment falls off at some point.

Indeed it seems so. After 4, maybe 5 hours of what Cal Newport calls "Deep Work," our output and learning ability *drastically* drop. So it is wise to stop after we become tired of either mind or body—moral of the story: study *less* and with greater intensity.

While we are studying, there are two trains of thought worth considering dealing with the duration of learning bouts.

First, a productivity method being tossed all around the internet, called the Pomodoro Technique, is worth investigating.

The Pomodoro Technique entails working (or coding in our case) for 25 minutes at a time, taking a 5-minute break after, and repeating that for a set number of cycles. There are endless variations on this that you can choose from on the internet.

While this method is trendy, and some swear by it, I don't believe

it is optimal for coding. The reason being is that it is tough to get anywhere in programming in 25 minutes.

By trying the Pomodoro Technique on many occasions, I found that the timer went off just as I got into the zone. So, by all means, try it, but I think there is a better alternative that science is backing up heavily.

The second and more evidence-based approach to learning has to do with a naturally occurring phenomenon in our body called an "Ultradian Rhythm."

Each one of us has an internal clock running 24 hours a day. The Ultradian Rhythm is a biological happening where we oscillate between alertness and sleepiness in 90-minute cycles.

The goal here for our purposes is to use this fact and study programming in 90-minute increments. Following our Ultradian Cycles allows us to get into the Deep Work we need and optimize our time and learning.

There is much more to say on this topic, so I highly suggest doing some of your research because I can't cover it without distracting from the overall message:

> ***"Study intensely for 90-minute with no distractions for maximum learning and health"***.

NOTE: If you are in the beginning stages of your journey, the Pomodoro Technique might work for you. But as you progress and

start creating more complex applications, the studies suggest you should transition to larger blocks of focused, undistracted learning and work.

Energy and Learning

While learning is possible at any moment in the day, not all points in the day are created equal.

Your levels of alertness (or autonomic arousal) play an essential role in focus, and therefore learning and retention.

Show up to a study session too alert, and you run the risk of getting distracted by every noise or movement in your vicinity. On the other hand, show up too drowsy, and you're bound to have a lackluster learning period.

As a gross oversimplification, spend one-minute breathing in more than breathing out if you are tired. Likewise, if you are hyper-alert, spend one to two minutes lengthening the exhale portion of the breath.

Some other tips for regulating your energy level include drinking or eliminating caffeine, performing exercises, and taking a nap.

In general, you will be most alert and ready for an intense learning or work session 30 minutes to an hour after waking up or right after a workout session.

Teach and You Will Learn (Feynman Technique)

The Feynman Technique. Google it. There is so much information regarding the famous Scientist Richard Feynman who outlined the four key steps to learn anything.

1. Choose a concept you want to learn about
2. Explain it to a 12-year-old
3. Reflect, Refine, and Simplify
4. Organize and Review

The main idea is to simplify what you have learned, refine the message, and teach it to someone. This method solidifies your knowledge and improves your comprehension of the topic.

Reframing Errors and Challenge

As crazy as the following will sound, it proved to be the "ah-ha" moment around learning needed to rapidly accelerate my programming growth.

It may even sound self-evident at first but stay with me.

In programming (and life), you will face many errors and challenges you have no idea how to solve. When this happens, you will become frustrated. I don't care how calm a cucumber you are; it is biology.

When we face a challenge that we don't know how to solve, certain hormones release, creating this feeling of agitation.

Here is the cool part. This reaction to stress is inevitable, normal, and most importantly, the **precursor and propagator of learning**.

WE LEARN BY FACING CHALLENGES.

The frustration is a sign that our brain listens to and secretes the necessary chemicals to remember what we are facing and how we solve the problem.

If you think about it, this phenomenon makes complete sense. We are, deep down, problem-solving creatures. In the wild, when we faced a challenge that threatened our survival, we needed to learn how to solve that issue and remember the solution for the next time.

And this process never went away. So unless you are a child and have neuroplasticity blasting through the roof, you need to face challenges to tell the brain to pay attention.

So the next time you are facing a problem, you can relax and remind yourself that this feeling means you are learning!

Practical Routines for Staying Self-Disciplined

Rest & Recover

Okay, great. We have learned so much by now, and perhaps you think that if you implement all these tactics and techniques, we can study all day without ever tiring. Wrong.

While the studying part is essential, **the real learning and consolidation happen during rest**. This fact is as valid for muscles as for the mind and memories.

Getting adequate rest is an absolute MUST when it comes to learning anything. So don't stress if you have one bad night's sleep, but try and keep a consistent routine that allows 7-8 hours of quality rest each night.

If you are learning and doing lots of deep work bouts, naps are also a great way to stay mentally sharp and improve memory and performance.

Aside from sleeping well, when you face a particularly challenging problem and spend longer than 30 minutes on a solution, the best move here is to stop what you are doing and take a walk.

The walking part is just as important as the taking a break part. Some exciting and astounding things happen during a walk that can be hard to get from other activities. I won't bore you with the details. Just know to take a walk (and it can't be on the treadmill — our body knows the difference).

Creating a Coding Routine (That You'll Want to Stick To

Now that we have learned and understood all the facets of creating a coding routine that works wonders, it is time to put it all together and create a system that works for you.

Some notes here are to:

- Be realistic and **think long term**
- Set an alarm or use another cue to **remember your commitment to code**
- **Reward yourself** (think a small piece of chocolate) after completing your assigned work
- Please **keep it simple**
- Ease into it. Once again, **consistency is king**

At the end of this eBook is a downloadable PDF example plan you can use and try out in your daily life to learn how to code in the most effective way possible.

II

MENTALITY

https://github.com/dreamingechoes/awesome-mental-health

4

10 Unique Tips for Effectively Learning How to Code

You can employ many practices, techniques, and mentalities in your life and coding to help you achieve your goals more effectively.

Learning these can be one of the most valuable meta-skills you can master to skyrocket your results. Applying these to your learning journey will be a pivotal step to creating the life you desire and deserve.

So, here are ten not-so-common but *highly* practical tips (I wish I knew when starting) that will help you learn to code effectively.

Tip #1 - Have a Long Term Approach to Your Learning & Career

Whenever taking on an arduous endeavor, it is critical not to burn up all your fuel right off the bat. Instead, let your interest in computers simmer and slowly build your passion for programming.

By doing this, you will come to appreciate and enjoy what you do while simultaneously avoiding burning out. And trust me, there will be a time in the future for you to ignite your fire and put a lot of effort into a desirable result. But let things happen naturally and gradually.

Also, programming is like any other skill; it takes loads of patience plus deliberate practice to become skilled and proficient. There are no shortcuts.

All that being said, don't be so hard on yourself, especially in the beginning. Learning to code will be challenging - take care of yourself and create a reminder every day to stay healthy and happy as a top priority.

Take it one day at a time.

Tip #2 - Create a Distraction-Free Zone for Learning

Every time new studies come out on productivity and learning, almost without fail, the result shows how much our environment affects our ability to concentrate.

Yes, there are some of us who "thrive in chaos" and seem to have an uncanny ability to focus in loud and busy environments. But, unfortunately, these people are the exception and not the rule.

Even so, just because you *can* thrive in a busy environment doesn't mean it is optimal for you to do so.

Most of the quality research around this topic have shown that a clean, comfortable, and organized environment improved ALL participants' ability to get more productive work done with less energy and less stress.

Eliminating visual and audio distractions is essential. However, to go one step beyond, turn off all your notifications and devices when in deep study/coding mode and have only one thing on your desk.

Some people even go as far as having a friend hide their phone or lock it in the car for the duration of their coding session. Whatever works for you!

On a side note, there are apps such as "Forest" or "Freedom" to keep you focused and distraction-free.

All this to say, creating a distraction-free zone where you can study is a guaranteed way to improve the effectiveness of your learning on your journey to **Become a PAID Front-End Developer.**

Tip #3 - Start Writing a Programming Blog

Starting a programming blog has so many benefits it is amazing how little this discussion is found in the coding space.

The first thing that stops most beginners from doing this is thinking they need to be an "expert" to write a blog. The fact is, everyone has something to teach, and your style might make a concept "click" by the way you explain it. In addition, learners love listening to people one or two steps ahead because it makes everything more relatable.

Another roadblock most beginners have is a fear that more experienced developers will criticize them or no one will read it. Both are valid, but neither should stop you because the potential upsides are enormous and *vastly* outweigh the potential costs.

The benefits of starting a coding blog are as follows but are not limited to:

1. Greatly **enhancing your learning** of fundamental and advanced concepts
2. Creating a **valuable repository of information** you can refer back to when needed
3. **Positioning yourself as a professional** in the field (which makes you stand out)
4. **Receiving opportunities** that otherwise would be veiled
5. Make some **extra cash on the side** to fuel your journey
6. **Help others** with problems they may be facing

Your blog website doesn't have to be the fanciest thing in the world. Create a simple WordPress site or use a service such as Hashnode. The point is to write.

Tip #4 - Learn How to Read Documentation Instead of Worrying about Memorizing Information

> *"Never memorize what you can look up in books" – Einstein*

The thing about software development is that the landscape is constantly changing.

If you try and follow the trends, learning every language and framework while neglecting the overarching concepts, you will always be playing catch up.

The fact is, you will forget a lot of things. **Mastering the fundamentals by building projects will help you retain the most amount of information for the least amount of effort in the most enjoyable way**!

The way you build projects is by reading documentation, not memorization.

By focusing on building projects, you will reap the rewards of inherent spaced repetition. You will also learn what skills and techniques you are repeatedly using and use that to hone in on your weak points.

So, if you are not practicing by creating projects that make you review and reuse what you have already learned, nothing is going into your long-term memory. You will forget what you have studied.

The solution is to read more documentation and better implement what you read from reputable sources because your memory is fickle and limited while words are not.

Tip #5 - Commit to Coding 3 Hours Every Day for 100 Days

I repeatedly saw from those who successfully changed careers and are now accomplished Front-End Developers is that they set up and followed a **solid learning routine.**

We have most likely all heard by now of the principle Malcolm Gladwell has popularized called the "*10,000-hour rule*". Similar to that rule, it seems for learning how to code effectively, 300 hours of dedicated and focused practice is the magic number for you to be & feel ready to land that first job.

The secret is out: **Consistency is KING!**

When you begin learning, your brain creates neural pathways to make it easier to retrieve that information next time. According to recent Neuroscience studies, neural pathways are more reliable by increasing the frequency rather than intensity.

What does this mean for you?

Well, it points to the fact that 1 hour a day for five days a week is greatly more valuable for learning than 3 hours in one day. *And even better is coding 3 hours a day, every day! (Remember to enjoy what you do.)*

This knowledge goes under the motto: "Work smarter, not harder."

So split your coding sessions into one or two 90-minute chunks and stay consistent. It is similar to working out - you need to do enough to be challenged, but not so much so that you are too sore to workout the next day.

To help with the motivational aspect, get a calendar and put a nice big ✕ or ✓ every day you successfully code for the time you choose to do every day.

NOTE: Remember that it is more about how deep you can focus in this time than the amount of time you spend. (30 minutes of quality work > 2 hours of distracted work.)

Tip #6 - Don't Fall for New & Shiny Objects

> *Don't follow the hype train, follow the results.*

This tip is simple. Don't constantly change your learning to fit in the new language or framework. It isn't going to save you. You will not gain superpowers.

Stick with reliable and proven solutions and frameworks

because the important part is learning how to learn - not memorizing a specific way of doing something, as we discussed earlier.

As hard as it can be, it comes down to choosing ONE path that is right for you and sticking with it until the goal is complete.

Life is not going away anytime soon that I know of, so take your time and complete what you start. **Knowing and proving you are someone who finishes what they begin is more valuable as a life skill and to employers than being on the "bleeding-edge."**

Also, never be the guy or gal willing to die on a programming language hill. And don't mind the "gatekeepers." You are not any less of a programmer because you are not using the hottest new tool.

Fighting over that kind of stuff is silly and childish. It won't get you anywhere, and we are trying to get somewhere by golly!

Tip #7- Keep Track of All Your Wins

I will keep this short and sweet.

You are awesome!

Let yourself feel that fact and remind yourself of it often.

Imposter syndrome is no joke, so to protect against this foul

beast, keep track of all your "little" wins and post it somewhere you often see, such as a bathroom mirror or behind your computer.

Every day is a success if you open up your laptop or turn on your computer and put forth the effort to do something challenging in the direction of your dreams and goals. Don't take that for granted. Celebrate often how far you have come, as learning to program is no small feat!

NOTE: It is best to keep quiet about it around others and learn for yourself... don't go boasting around town. Share more significant wins with those you love.

Tip #8 - Learn the Same Concept from Multiple Sources & Different Mediums

Not all tutorials are created equal.

That single piece of advice would've cut my learning time in half. But, alas, I hope you heed this message and understand the consequences of following the wrong mentors or tutorials.

Take some time to explore different teaching styles, sources, and ways of information transportation. You will have to find what works for you on this one as there is no right or wrong way.

There are many mediums to choose from for learning to code. Some of the best options to use and include in your routine are:

- Reading other peoples GitHub (look for professionals)
- Studying a comprehensive guide or blog post
- Implementing a project idea from a YouTube video after watching
- Purchasing and walking through a book page-by-page
- Talking with an experienced developer
- Explaining a concept to a complete beginner
- Practice your skills by playing programming games

Tip #9 - Prioritize a Good Nights Sleep & Maintaining Healthy Habits

Many aspiring programmers have a skewed perception of what makes a good developer. These (false) ideas come from the portrayal of coders on the internet, TV, and movies.

I am sure some successful programmers indeed do fit this stereotype, but it is by no means the majority and not something to "strive" towards.

From experience, the software developers that have truly stood out and lasted the test of time almost always prioritized their health and well-being over their job.

Aim for 7-8 hours of quality, undistracted sleep. If you can't get this, don't stress out. These are not hard and fast rules, and I understand sometimes life can be crazy! Being a new parent myself has tested these habits. So do your best!

Try to get 20-30 minutes of walking OUTSIDE IN NATURE instead of the treadmill if at all possible.

Exercising for 30+ minutes 3 times a week has also been shown to have a whole host of positive benefits (and for not just the body but also the mind and emotions).

This eBook isn't a guide on eating, moving, and being healthy. However, it was essential to put a primer here to realize where you are and start taking baby steps towards a better and sustainable future.

Tip #10 - Be Social & Realize that Networking is Essential (and Inevitable)

We get it. You are an *introvert* and don't need anybody to succeed. Or you don't want to play the social game; that is why you want to be a developer.

Perhaps you resonate with neither of those, but no matter where you are with your comfortability talking and networking with other people, it doesn't detract from its importance in this process.

You don't have to become a social butterfly overnight but instead, open that cocoon just a smidgen each day.

Here are some ways you can network without being overly weird or going out of your comfort zone too far:

- First, try asking colleagues or coworkers more questions or utilize online forums.
- Find a coding mentor that you speak with regularly.
- Join a community of learners like yourself.
- Create a Twitter account (or maybe not) — better yet, a LinkedIn profile and connect with one new person who has a tech job a day.

Life and learning to code is not a popularity contest. We do this because it opens the door to landing a new job, finding freelance clients, making new friends, helping others learn, and being a part of the community you want to join.

Networking doesn't have to take more than 1 or 2 hours of your time each week. Start small and continue making connections and building relationships.

One of my favorite sayings on this topic is:

> *"Dig your well before you are thirsty".*

Be yourself, stay humble, and over time your network will expand. Building relationships is an area where you truly get what you give. So put in the effort, add value to some people's lives and carry on your way.

Overview

1. **Have a Long Term Approach to Your Learning & Career**
2. **Create a Distraction-Free Zone for Learning**
3. **Start Writing a Programming Blog**
4. **Learn How to Read Documentation Instead of Worrying about Memorizing Information**
5. **Commit to Coding 3 Hours Every Day for 100 Days**
6. **Don't Fall for New & Shiny Objects**
7. **Keep Track of All Your Wins**
8. **Learn the Same Concept from Multiple Sources & Different Mediums**
9. **Prioritize a Good Nights Sleep & Maintaining Healthy Habits**
10. **Be Social & Realize that Networking is Essential (and Inevitable)**

5

The REAL Reason Most Self-Taught Developers Fail

There is no way around it; programming is hard.

Learning to code is difficult for everyone, not just self-taught developers. So, in this chapter, we will dive into the most common reasons why most self-taught developers "fail." Yet, these reasons apply to anyone who doesn't succeed with what they set out to do. (Believe it or not, many Computer Science students quit too!)

These reasons outlined here are not to scare you or strike fear into your heart. Instead, their purpose is to spark reflective thought on the coding path you are either taking or planning to take so you can avoid and overcome these challenges and succeed.

REASON #1: Lack of Clarity

How can you get where you want to go if you don't know where you are headed?

And even if you have a general idea of what you want to be going towards, how would you know the best way to get there if you have never been?

Many self-taught developers know they like computers to some degree, or maybe saw that fancy paycheck and a promise they can do it in 3 months. But, no matter the primal mover, because they are a novice in a vast field, they have no idea of the full scope of potential paths and possibilities available to them.

This lack of clarity can paralyze or distract endlessly. And most of the time, it does just that.

Beyond being unaware of the roads they can take, most self-taught would-be-devs are also oblivious to the pitfalls the majority of similar souls have fallen into before. Some common ones are *tutorial hell* or watching way too many theory videos and not enough building.

Another peril that a lack of clarity creates is a potential bad career fit. We get it, the tech space is cool and hip and all, but it might not be for you. However, if you have read this far, I doubt this is the case. **You have enough interest to succeed and land your first job as a PAID developer.**

The last destroyer of dreams having no clarity brings is most

tutorials online don't share the WHY you are learning what you are, leaving a massive gap in knowledge and, therefore, future ability.

Mix these all together, and you have a recipe for learners to set unrealistic goals, not realizing that **learning to code is a marathon, not a sprint**.

REASON #2: Lack of Support

Now that we have defined how lack of clarity can kill your goal of becoming a paid software developer, we must discuss how having a lack of support can do the same.

Support separates into two categories. The first category is support from those above us who have accomplished what we wish to achieve. The second type is those that are comrades walking this path alongside ourselves. We need both if we want to succeed in this grand endeavor.

Having a solid support system provides a strong network for landing a job. In addition, being an active member of a community has positive emotional benefits; **it dramatically improves our motivation by way of buy-in, making the whole process much more enjoyable.**

With a support system, when the honeymoon phase wears off, your mentors anticipate this, and you can trust what they say and advance to the next level. Unfortunately, this exact scenario for those without a support system results in them quitting at

precisely this moment.

The sad truth is that many of us deal with copious amounts of doubt in our minds every day and don't have the essential life support necessary to succeed at challenging tasks. Plus, you can almost trick yourself into believing you are braver and more passionate by taking the journey alone.

This "needing" to be a self-taught programmer is a huge hindrance. You are not less of a software developer for taking a quality Bootcamp or getting a Computer-Science degree.

Choosing to utilize expert opinions and guidance is the *more intelligent* move. You know where you want to go, have identified the best route, and are taking the steps forward with no mind to how you or others will perceive it. THAT is passion.

REASON #3: Lack of Structure

It may not seem like it in this post, but **I want you to achieve your goal**. I spend hundreds of hours creating this guide so you can succeed. And you will if you follow the steps outlined.

So, I hope all you self-taught programmers are heeding the messages in these words. There are solutions to these problems, and I believe you can teach yourself to code if you solve the previous two reasons most aspiring developers fail and implement a solid routine and plan.

Without a set structure to follow, life will get in the way. It is

much harder to fight the world's schedule every day and "find time" to program. Your learning time must be evident and protected at all costs.

Having a study and coding routine is one aspect of the structure. Another factor is setting monthly goals and deadlines instead of daily ones.

Daily coding should be a routine, a system of practice. Monthly goals should be a target and the next big step that will move you forward in the correct direction. Monthly goals set a deadline.

As much as I hate them, deadlines work. They work so much that they are the reason you are reading this when you are. Having a deadline is an essential programmer's productivity tool.

Besides having a routine, clearly defined goals, and a deadline to reach them by, we also need to think about our overarching plan.

While schools worked for some, they never explained how to independently tackle a significant learning challenge.

The skill of breaking down large amounts of information into a solid curriculum is a daunting and challenging task, even for a professional developer. Being a self-taught developer means you would be learning two things at once!

There is no shortage of information on front-end development. It would be a miracle if you could parse all that data before starting and turn it into an effective study plan. It can be done,

but why reinvent the wheel?

Luckily for you, at the end of this eBook, you can download the 6-Month plan and the easy-to-implement study routine and **succeed in your goal of becoming a paid front-end developer.**

Last but not least, I want to talk about one of the biggest pitfalls I see many aspiring coders fall into often. This career-destroyer is "tutorial hell."

Tutorial Hell is a term to describe when a beginner has perceived incompetence in their ability to code, which creates a negative feedback loop of endlessly watching or reading new tutorials, not getting anywhere in the process.

The negative spiral is quite nebulous. Nevertheless, here's an inner-view of how it all works in the form of a story.

Little old Timmy just got done watching Mr. Robot. He has decided he wants to become a programmer. So Timmy jumps on the computer and starts watching a tutorial on how to create his first website.

"This is so awesome," Timmy says. So he continues watching video after video to learn everything there is to know about software development.

But, because he is consuming more than he is creating, his mind lays the groundwork for imposter syndrome.

Timmy has seen tutorials on HTML, CSS, JavaScript, React,

Python, Ruby, and on and on, all without building the projects he watches.

By attempting to learn in this way, it creates a large gap between what Timmy knows is possible and what he can do. Because Timmy knows he should be able to create a new full-stack application as he watched the videos and cannot, his brain thinks he is incompetent.

This feeling of incompetence registers in little Timmy's brain as a desire to watch more videos to "learn" what he is missing.

Do we see the loop in action here?

Take tutorials beyond current implementation ability => Feeling of inadequacy => Watch more tutorials to fill the gap => Greater feeling of inadequacy => . . .

The actual process of learning how to code isn't sexy. And having the expectation you can stay up late into the night over and over and magically become a programmer is a gross falsehood.

The truth is much less appealing, and it goes like this:

1. After getting a great night's rest to recharge, you try to start your morning on a positive note.
2. You sit down at your computer and learn one topic you decided on the night before.
3. It was frustrating, and you didn't get as far as you like, but alas, you have to go to work.
4. You go to work and come home exhausted.

5. There are 30 minutes before dinner, and you use that time to try and build a project using the information learned this morning.
6. After the day is done, you write what you have learned and create tomorrow's tasks.
7. Night comes quickly, and despite not progressing as far as previously hoped, you go to bed and get your essential 7-8 hours of sleep.

I will repeat it, *becoming a programmer isn't sexy*. It is a lot of hard work, but boy, is it worth it! **Changing careers can and will change your life for the better.**

All of this to say that you can be a self-taught programmer if you spend the time to **get clear on what you want**, the best way to get what you want (within reason), **find a community that will support you emotionally, and technically.** Then you **create a solid schedule and routine to code consistently.**

After doing all those, your odds of becoming a programmer skyrocket. The only last thing you need to do is commit to a path and not give up. Easier said than done, but with your perseverance and this guide, you should be a developer in no time! ;)

6

What My Most Successful Students Do That Puts Them Ahead of the Rest (7 Little Habits That Make a HUGE Difference)

Becoming a star student and employing these effective tactics will improve your learning rate ten times and help you get your dream job sooner!

While you may not have respected all your teachers at school, when continuing your education beyond the childhood years, it is essential to be the best student you can be to get the most out of the lessons and, therefore, your life.

So let's see what my top students do that puts them ahead of the rest. *(Hint: we have already spoken about most of these).*

SUCCESSFUL HABIT #1: They Eliminated Distractions & Setup a Clear Coding Schedule (With Deadlines)

Distractions are the most deadly focus and motivation killers.

The students who heeded this warning and took time to set up their coding environment to improve their learning experience from the beginning had a considerable headstart that compounded into remarkable results.

These students prioritized deep learning sessions to get the most out of the 24 hours we all have. **They did not stay up later or**

work longer hours; they used their time in the most effective manner because they also realized the importance of healthy mental and emotional well-being.

In a way, learning to code or taking on any great challenge does not only teach you valuable skills you will use for the rest of your life. It also teaches you how to be a better person that can get more done with less stress. So it's the best of both worlds if you think about it.

SUCCESSFUL HABIT #2: They Realized the Importance of Teamwork

Great coders (and students) understand that they are not working in a silo.

You don't have to be a charismatic leader or a social butterfly to realize how important other people are to your success.

Take a moment to see where you are at this stage in your life. Who are all the people that have helped you along the way? It could be specific humans that have given their time, energy, or money to aid in your well-being. Or it could be someone you never met that created a great product or piece of content that changed your life for the better.

There is no way to do this exercise, and honestly believe this journey can be done alone. **Star students recognize this fact early on and make it a point to create relationships built on trust and value.**

They listen to their classmates and help out when they can do so. They work well together, and this characteristic will take them far in their coding career and beyond.

SUCCESSFUL HABIT #3: They Attending Every Class & Office Hours Ready With Questions

A teacher can instantly tell whether or not a student is engaged. An easy way to identify what students are there to succeed is by asking: are they showing up consistently, and what are the quality of their questions?

Those two factors alone account for the majority of being a great student who will extract the most value out of a course or curriculum.

My best students put their learning ahead of most other aspects of their life. They rarely, if ever, missed a day, and when they showed up, they had already researched the basics so they could come into class with more fine-tuned questions.

I want to make it clear that no questions are wrong or dumb. When we talk about quality questions, it has more to do with engagement than intelligence. As a general rule of thumb, if a quick Google search can easily find the answer, it most likely isn't a great question.

Another aspect seldom talked about is curiosity. Yet, curiosity is essential to learning and is a tremendous motivator. When a student is curious, it means they care. When they care, their

brain holds on to the information better, and they seek out solutions to their problems rather than running away.

Be curious, and results will follow.

SUCCESSFUL HABIT #4: They Commit Their Code to GitHub & Welcome Feedback

I get it. That feeling is universal. Feedback is a scary beast, and if you haven't had the best experiences with it in the past, it can be one of the most significant obstacles preventing a life of learning.

But, we cannot let that stop us. Overcoming this fear of failure, looking stupid, and receiving feedback is another essential habit my excellent students possessed.

> *Like you and I, star students still have that feeling of being afraid to ask for help or have their work reviewed, but they have a more profound motivator that overrides that emotion.*

See, when you want something more (to start a career in tech), and you know the path to get there (utilizing a mentor and getting feedback), the fear is no match.

Use this knowledge and ask for feedback the next time you create something you put your focus and energy into, and it will become

easier to do so in the future!

The second portion of the successful habit deals with a specific tool, *GitHub*. (GitHub is a web-based version-control and collaboration platform for software developers.)

Publishing your code to the web has a few positive effects. One of them is that you now have a place where all your work is stored to find out how you solved a problem when facing it in the future.

Another great benefit is seeing how others solved their problems and reviewing enterprise-level code written by some of the most brilliant senior developers.

To continue listing the positives, **committing your code shows employers that you have been active and learning** most days**.** In addition, they can see the type and level of your projects and mark your enthusiasm by how often you are pushing up and building something.

Plus, having a place for you and employers to see your code serves twofold. For one, it is an excellent ally in your job hunting, and the second is it pushes you to code more often and serves as an accountability tool.

It is no secret now that successful students push their code to GitHub often and welcome feedback on their work (even when they don't want to).

SUCCESSFUL HABIT #5: Focus on Learning the Fundamental Concepts

If you want to be a successful student and land that first job as a PAID developer, you need to focus on mastering the fundamentals.

All the great trainers in every arena, from boxing to coding, hammer in the point that **those who focus on the fundamentals will surpass those following the latest trends.**

Yeah, you won't be able to pick up a new hot date by letting them know you have another intimate night researching JavaScript fundamentals (although you never know).

But, by doing what others will not and continuously creating a more comprehensive understanding of software development and your particular niche, you are guaranteed to succeed over time.

On a practical note, my students who went on to get jobs had a plan like this:

1. Ask a mentor or follow a curriculum to see what they should be learning next.
2. They find the top 3-5 videos or articles on the topic and consume them.
3. They then build or practice using the knowledge they learned, asking questions along the way.
4. From here, they utilize the power of teaching and pass this information along to others either in-person or through

an online medium such as a blog.

5. After a few days, they revisit the concept, check their notes, and assess their understanding. If they need more practice, they repeat the loop; else, they continue learning the next concept.

SUCCESSFUL HABIT #6: They Worked on a Passion Project

Don't let the word "passion" intimidate you. For our purposes, we are not worried about whether or not you have a deep burning devotion for David Bowie's Labyrinth and want to build something to commemorate him and that aw— movie.

By *passion project*, we mean, is there something you are interested in outside of coding? If yes, which is inevitable, what is one thing you can create that revolves around that topic or theme?

Everyone has a passion project. Yes, even *you*. I don't care if you despise everything in the world and don't enjoy doing anything; you can make a website about depression.

Okay, so now that you know there is no escape, that a passion project lives within you and must come out, we will discuss why it is essential and beneficial to work on this outside of your regular learning.

The main point of having a coding project that is interesting to you is that it increases your motivation to code. **The more you**

like the topic or, the more valuable you perceive the outcome of the endeavor, the more chemicals your body will release to get you to stop being lazy and get on the computer to code.

And the more you code, the better you become. What a great feedback loop!

Besides higher motivation and getting better at the skill you wish to develop, it will also serve as an excellent centerpiece for talking during coding interviews, at a meetup, or even with friends. Additionally, having something you are proud and excited about will, once again, increase your motivation and drive to code and learn more. It's a win-win.

As a bonus to those already amazing benefits, having a passion project allows you to gain practice building an application similar to the ones you will create for the company you choose in the future. **You will gain real-world experience and learn at an accelerated rate.**

You will inherently go over each concept multiple times by building your project. You will learn how to Google more advanced queries. And you will be more willing to work through errors instead of giving up and moving on to the next project.

Most importantly, you will learn to solve problems on your own and will basically be a developer at this point.

Feeling like a developer will make you want to do more "devel-opery" things and be around more developer people. All good things for you when you want to start applying and get that first

job you so desire and deserve!

SUCCESSFUL HABIT #7: Stay Informed on Tech News By Listening to Trusted Experts

The final habit all-star students perform, whether I have mentored them one-on-one or taught them at a coding Bootcamp, is they stay informed on tech news and information.

Once again, ensure you aren't using this as a procrastination tool. Instead, when you are most alert, spend that time coding or doing deliberate learning.

But, when you have done your bidding, take time to enter the software development arena. **Listen to trusted experts on topics that intrigue you. Keep the curiosity alive.**

Here are some of the ways you can stay informed on essential development conversations:

- RSS to top blogs
- Subscribe to coding YouTube creators
- Attend meetups or seminars
- Join a live or online front-end developer group
- Enter a Slack or Discord channel for aspiring developers

The goal here is to become a member of the greater community. By doing this, you will learn specific terminology, meet potentially valuable people, feel more like a developer, and have a feeling of being included.

NOTE: If you want to join a great community, you can subscribe to the blog or enter the Bulletproof Slack group for learners just like you! If my neighborhood isn't suitable for you, there are plenty of others to choose from on the web! :)

III

KNOWLEDGE

7

How to Make a Few Simple Changes to Boost Your Focus & Productivity (as a Programmer)

A massive amount of articles and videos on the web are focused on creating habits and increasing productivity.

After reading the most notable books on the topic and narrowing down the best blogs and videos, I have created a simplified list of tips and techniques you can practice TODAY that will skyrocket your overall productivity in life and programming.

We will split the list into four distinct categories:

1. **Time-Management Techniques**: How to work on the right things.
2. **Motivation Methods**: How to stay inspired to learn every day.
3. **Focusing Strategies**: How to get the most out of each session.

4. **Efficiency Procedures**: How to get more done in less time.

Each section will provide tools and tactics to improve your overall productivity.

Let's start with some time-tested & proven time management techniques.

Time Management Techniques

> "Every minute you spend in planning saves 10 minutes in execution; this gives you a 1,000 percent Return on Energy!"— Brian Tracy

Determine High Impact Tasks

This approach reminds us to cut out all the fat in our lives. When we determine our High Impact Tasks, we can eliminate the noise and delegate the tasks out of our "genius" zone.

Reevaluate all the actions you perform daily as a programmer and honestly assess what has been working and making you feel busy without moving you ahead.

Some low-impact tasks could include learning non-essential languages and how much time you are on social media instead of building projects. It could also be what types of projects you are working on currently; *(if you are trying to get a job) do they best show your abilities, and would a hiring manager be pleased to see them on your GitHub?*

Whatever you deem as low impact should be eliminated or delegated. All medium impact tasks should only take 10% to 20% of your overall work time, while high impact work will occupy a minimum of 80% of your routine.

Plan Your Priorities & Learn to Say "No"

Getting more done has an inverse relationship with taking on more responsibility. The less we are responsible for, the more we can hone in and perform that task at a world-class level.

To implement this knowledge, **go into work knowing what 3-5 tasks you can do each day that will disproportionally move you forward more than anything else.** Then, execute those tasks and don't worry about the rest.

If you take this advice sincerely and delegate all those smaller "to-do's," your output will surpass ten developers. *(Okay, perhaps not, but it will significantly improve your impact by at least two-fold).*

Long and short — say no to more things and put your head down laser focusing on the highest impact tasks every day and watch your productivity rise exponentially.

Batch and Burn

These two simple productivity methods go as follows.

Batch: Assignments of a similar nature should be done together in one session. Batching works by avoiding the pitfalls of task-switching (or multitasking) - which drastically reduces our ability to focus and get stuff done as a byproduct.

When we read and answer all our emails one afternoon a week instead of multiple times before lunch, we will spend 1 hour a week instead of 45 minutes each day doing the same amount of work. More simply, *read and answer all emails once a week and save a huge amount of time.*

Some everyday tasks you can and should batch as a programmer or aspiring developer include coding, planning, sending applications, networking or social media, and anything else that you can think of that you need to do daily or weekly.

Burn: This is also known as the "two-minute rule." The general premise is anything you can do in under two minutes, do it now.

This tactic works well in every situation other during a deep working or learning session when you must eliminate any distractions at all costs.

Motivation Methods

When we talk about motivation, we speak about more than feeling inspired and fired up to work.

You don't need any tips or tricks in this distinctly driven state. But, not every day will you feel this way.

The fact is, some days will be downright difficult. So the motivation explained here has to do with maintaining a sustainable system geared towards a specific aim.

Find Your Why

When you have a strong reason that reminds you often of why you are doing something, it can serve as one of the most powerful motivators in our lives.

The WHY has to be honest, tangible, and something you believe you can achieve.

Don't worry about sounding vain or childish with your desires, as they will change over time. For example, it is okay to change careers and get your front-end developer job because it pays well, just as long as that isn't the only reason.

For me, it was so I could provide for my fiance (now wife) and family and give them the life they deserve. I also had a passion and affinity for creating websites, which helped! ;)

Don't Banish Breaks

It is easy to believe that you need to "work work work" to be successful in today's day and age. Unfortunately, message after message bombards us confirming this workaholic mentality, but I have two things to say.

The first is that this is entirely untrue and not necessary.

The second is the semi-cliche advice of "turn your work into play, and you'll never work a day."

You don't need to sacrifice your well-being to become a programmer. Of course, in the beginning, if you have another job and a family, it can be hectic, but that won't last forever, I promise.

Take it day by day and keep putting in the reps, and your life will transform before you know it!

Eat the Frog

"Eating the frog" means that you tackle the biggest and most important tasks first thing in your workday.

This system works wonders for some, but not others. For example, if you come into your work session with your eyes barely opened and you need three coffees before you are "functional," this may not be the advice for you.

As for anyone else, I advise you to try it out and see how great you will feel after completing the most critical work right off the bat. It makes everything else you do feel like a bonus and the day usually goes so much smoother!

This method reminds me of High school exams. When the big scary test was at the beginning of the day, it felt great to get it out of the way. But, if it was during the last period, most people turned into a mess with all the worry about how it will go.

Process Over Perfection

Process over perfection. Write it on a post-it note, put it on your wall, get this tattooed to your forearm. Whatever you have to do for this message to sink into your system.

I know we all are "perfectionists" at heart, but it won't cut it as a programmer. 99% of so-called *perfectionism* is a fancy way of saying you are procrastinating and fear criticism.

In tech, you will have to ship code that isn't ready to meet some client deadline. It is best to get used to this fact now and realize that your code will never be perfect.

So what can we do? Instead of focusing on the end result all the time, try and master your process. Code for a set amount of time on a set number of features and do that consistently. Put all your energy and focus into creating the best work you can in the given timeframe.

If you are going to try and perfect anything, make it your process of sitting down and coding consistently because that is something you can control.

Launch Yourself Into Action

This technique seems simple and too easy to work. Yet there has been a book written about it with tons of raving fans. Launching yourself into action or "blast-off" is a technique whereby when you feel resistance to get up or do a task, you count down from 5 to 0. When you reach 0, you get up and do what you were procrastinating.

Before you write this one off, I recommend you try it out sincerely. You might be amazed at how something so simple can get you out of your head and into action.

One of the most exciting research to come out of productivity science that backs this up is the discussion on the motivation chemical *dopamine*.

Pop-Sci has coined dopamine as the "reward drug/chemical", but that is misleading. Dopamine is primarily released when we *anticipate* a reward rather than once received.

What this means for us is that if we start working (perhaps by using the "blast-off" technique), dopamine will begin to drip into our system, creating what we feel as "motivation."

Our shared knowledge has it all backward. For example, we think

we should feel motivated to begin work, but it goes more like this: start working => dopamine releases => we feel motivated to continue working.

Knowing that there will be resistance when we start performing a demanding task, but that it will get better if we push through the initial discomfort, can be one of the most valuable pieces of knowledge in your arsenal!

Take Advantage of Cues and Rewards

If you are interested in learning more about "cues and rewards" and habits in general, I can't recommend enough the book "Atomic Habits" by James Clear. Absolutely phenomenal book.

For our purposes, I will keep this topic short and suggest that after you perform a cycle of your work, reward yourself with a small square of chocolate or your favorite song. Then, every now and again, skip giving yourself the reward to reap maximum benefits.

Focusing Strategies

> *"Manage your energy, not your time." - James Clear*

When you realize the outsized effect great focus has in creating, building, and learning, you can begin to work less than the rest while completing higher quality work and making more impact.

We have talked a lot about the importance of focus already, so I will just run through some basic practices and things you can look further into if interested.

Multitasking Catastrophe

Less is indeed more when it comes to performing at your best. So do one thing at a time and don't switch between coding and email and Slack, as that will wear you down at an alarming rate.

Caffeine from the Coffee Bean

It is no secret that caffeine increases your alertness and energy. As long as you are not too hyped-up, use this tool to improve your mental agility for completing complex tasks you already know how to do well.

Meditation & Mindfulness

> *"Mindfulness is about focusing attention on the present moment, and practicing mindfulness has been shown to rewire the brain so that attention is stronger in everyday life" - Neuropsychologist Kim Willment*

Take this one or leave it. Millions of people love meditation, and there are well over a thousand quality peer-reviewed studies about the positive benefits of meditation, one being increased focus.

I will not try and convert you, and there is enough about meditation and mindfulness on the web already.

Healthy Lifestyle

Another topic I need to mention because it is essential (and overdone) is creating a healthy lifestyle.

Here is what you should be focusing on that provides the greatest return on investment:

- Sleep
- Nutrition
- Exercise
- Nature
- Walks
- Relationships
- Hydration
- Journaling

Once again, I am not here to tell you how you shouldn't eat pizza, and becoming a programmer requires you to live the life of a Zen monk.

Please employ one of these habits at a time, improve one thing about it, and gradually watch your life change dramatically. As a society, we underestimate how much lifestyle affects learning and success.

Leave the phone on the Airplane

While in deep work mode or learning something new, turn your phone off, put it in another room, and don't check it until you have completed what you have decided to do.

Avoid Excess Alcohol

This guide certainly is not a secret AA meeting. This tip is for those who wish to heed the message.

No judgment is being passed on your life choices here. But, for the sake of being throughout, science has confirmed the adverse effects drinking alcohol has on sleep and next-day performance. And for all you day drinkers and night sippers, it doesn't look promising.

(Perhaps consider taking this learning journey as a chance to discover a new side of yourself that doesn't rely on substances as much! :)

Efficiency Procedures

The efficiency procedures listed below are similar to the focus strategies but are directed more at the bigger picture.

Create Shortcuts & Snippets

In your text editor of choice, create snippets for common chunks of code or functionality you make often. Saving one second 10,000 times will give you a few hours back on your life. These will stack up and provide increasing dividends as the clock goes on ticking.

Also, automating the repetitive tasks in your life can work wonders while simultaneously allowing you to work on your programming chops!

Never Underestimate the Utility of a Whiteboard

I never thought this would be a hill I would die on, but alas, here we are.

After getting a whiteboard, I was able to get my ideas into the world and work with them in a way that, simply put, greatly enhanced my productivity and enjoyment.

You can use a whiteboard to design mock-ups of websites, write your daily priorities, and it can be a place you explain new concepts you are learning to others. The possibilities are . . . vast.

Focus on the Fundamentals

When you spend the majority of your time learning the new framework or tool, you are (in my opinion) wasting your time.

We all need to learn more and improve, but will memorizing the syntax of another language really improve your programming skills?

Once you have a solid tech stack, the more optimal thing to do is **focus on software development fundamentals rather than the specific implementations of those concepts.**

There are tons of great resources online to choose from in this arena!

Measure Your Results

Once you have a solid routine in place, you can start tracking your sessions and determine what factors create great learning/-coding sessions and which negatively affect your performance.

You can also use measurements to track how your application strategy is working by calculating the percentage of resumes getting sent out that get responses and how many of those lead to an interview.

There is so much you can do with tracking, but it is best to keep it simple in the beginning and focus on the coding!

Rubber-Ducking

Another humorous one here that works wonders. "Rubber-ducking" is purchasing a little rubber duck (yes, the ones you put in the bathtub) and placing it on your desk. Then, when you get stuck on a problem (or just need a friend), you explain to it what the error is in as much detail as possible.

Talking to a rubber duck does two things. First of all, it gets you out of your head, and secondly, it makes you look totally insane. No, the second thing is it allows you to ensure that you understand the error before tying willy-nilly to solve it.

Next time you have a nasty bug, talk it out with an amiable rubber duckie.

Utilize the Power of Deadlines

We talked about this earlier, but it bears repeating. **Deadlines work**. They will allow you to better estimate project times in the future (which is a valuable skill), and they will also help you get this project done in a more timely fashion.

Consistency is King

If you haven't downloaded this critical piece of information already, now is the time to let it seep in. Consistency will be your number 1 asset in your coding journey. Protect this quality at all costs!

When you code every day, you no longer have to spend 30 minutes remembering where you left off the previous day, and this fact can add 3+ highly productive hours of work each week.

Productivity Conclusion

So, while this primer isn't an exceptionally detailed guide on these protocols and practices, and it certainly isn't exhaustive, it should give you practical ideas you can implement today.

The primary goal was to initiate you into some productivity practices you can test and implement if you so wish.

It bears to say that productivity isn't an *end. It* is a *means*. So please don't spend your life over-optimizing everything; it will drive you and everyone around you nuts.

This section shouldn't feel overwhelming. Try choosing one or two ideas and implementing them this week. Then, if they go well, you can always come back and integrate the ones you feel would work best with your personality and specific need.

8

Essential Front-End Interview Preparation Materials (Technical & Behavioral)

There are tons and tons of lists on lists of program preparation materials. Here I have gathered, refined, condensed, and curated the creme of the crop when it comes to practical advice and techniques to acing the interview.

If you use a resource that is not listed, it does not mean it is an insufficient resource. Use your judgment and find what works best for you.

Yet, this list below contains the absolute **BEST tools, courses, videos, articles, and GitHub repositories** I, and many of my successful students, have used to land high-paying developer positions.

Courses

AlgoExpert

AlgoExpert is the ultimate resource to prepare for coding interviews. Everything you need in one streamlined platform.

(This is not an affiliate link. I have used AlgoExpert and thoroughly enjoy this platform. It was one of the main reasons for landing my most lucrative position.)

GitHub Repositories

Tech Interview Handbook

The Tech Interview Handbook is an excellent GitHub Repo of curated interview preparation materials for busy engineers.

Front-End Developer Interview Questions

This repository contains a list of helpful front-end related questions you can use to interview potential candidates, test yourself or completely ignore.

Coding Interview University

For a more comprehensive guide, check out A complete computer science study plan to become a software engineer.

Front-End Interview Questions

This Invaluable GitHub Repository is here to help the Front End community rock the interview.

Articles

Codecademy

Codecademy is a superb blog resource with an article detailing 7 CSS questions you will see on an interview and potential answers you can give and a similar post on 7 HTML questions you will see on an interview.

Indeed

Being a job board and all, Indeed.com helps out its job-seekers by posting helpful tutorials like 65 Junior Developer Interview Questions (With Example Answers).

Eric Elliott

The prominent software developer author, Eric Elliott, describes the 10 Interview Questions Every JavaScript Developer Should Know.

Videos

Algorithms and Data Structures Tutorial - Full Course for Beginners

In this 5-hour video on the basics of algorithms and data structures, you will learn about algorithms and data structures, two of the fundamental topics in computer science.

Q&A Posts

Ammon Bartram AMA - (Triplebyte)

Ammon Bartram, co-founder of Triplebyte, an innovated engineer job board tool, posted on Reddit a Q&A all about programming interviews with tons of great questions and responses.

Reddit Question to Interviewers

In the post titled "At the end of a job interview they always ask, "Would you like to ask any questions?" What question should the candidate ask?" Redditors gathered together and gave some

great answers to the age-old question!

Tools

Front-End Mentor

Front-End Mentor is how you can improve your skills and solve real-world HTML, CSS, and JavaScript challenges. In addition, this fantastic tool will prepare you for take-home interviews and overall front-end knowledge.

LeetCode

At some point in their career, every programmer has used Leetcode or a similar tool to practice coding interview questions at some point. It is the best platform to help you enhance your skills, expand your knowledge and prepare for technical interviews.

NOTE: There are almost infinite resources out there aiming to make your developer interview run smoother. This guide is not an exhaustive list, but it is comprehensive enough to be all you need to crack any coding interview.

9

12 MUST-KNOW Programming Terms to Ace Your Next Interview (+3 Bonus Questions)

While there are a couple of hundred new words you will learn on your journey to programming, some of them are more important than others.

This chapter will focus on the **12 most important words (and top 3 comparisons) you will need to know to be ready for the technical interview.**

12 Front-End Developer Terms

SEMANTIC HTML

Semantic HTML uses description HTML element tags to describe the meaning of the content to both the browser and the developer.

Examples of semantic elements include the ‘<form>‘ tag or the ‘<article>‘ label.

CACHE

Computer Cache is a prearranged storage location that collects data temporarily to make it easier to retrieve data, allowing websites, browsers, and apps to run faster.

MEMOIZATION

Memoization is a programming technique used to enhance application performance by caching the return value of a function call. Then, the cached value will be returned the next time you call that (expensive) function.

RECURSION

Recursion is a method used to solve certain complex problems whereby a function calls itself until the return result reaches the base case requirement.

SEO

SEO stands for Search-Engine Optimization. This term used to represent how to rank first on search engines like google, but now encompasses a whole range of activities developers and

marketers use to create websites or posts that solve a problem better than anyone else.

DOM

the **DOM** or Document Object Model is a JavaScript object representing the HTML on the page. A developer can use or "manipulate" the DOM to create, read, update, or delete elements or content on the page.

JAVASCRIPT DATA TYPES

Data types are a classification that describes the type of value a variable holds. For example, we can use six primitive (and immutable) data types when defining a variable in JavaScript.

String: Any variable that holds any characters inside quotation marks.

```
const myString = "This is a string";
```

Number: Any variable that holds a number integer without quotation marks.

```
const myNumber = 44;
```

Boolean: Any variable that holds either a true or false value

```
const myBoolean = true;
```

Symbol: Any variable set to a value not equal to any other value.

```
const mySymbol = Symbol("A");
```

undefined: Any variable that is declared but has no value assigned to it.

```
const myUndefined;
```

null: Any variable purposefully set to have no value.

```
const myNull = null;
```

RESPONSIVE DESIGN

Responsive Design refers to creating aesthetically pleasing websites on all screen sizes, from a large desktop to a small handheld device.

Because the prevalence of phone use for web surfing is increasing, so is the demand to create and have a responsively designed

website or application.

ASYNCHRONOUS PROGRAMMING

Asynchronous Programming is a term used to define code running outside the call stack.

The antithesis, synchronous code, is the default way the computer runs, one thing at a time in a FIFO order.

But, if a function relies on some data that will not be completed right away, instead of making the whole system wait to execute other tasks, we use asynchronous code and create a "Promise" to run the async code completes.

API

An **API** is a more straightforward way to say Application Programming Interface. An API is a set of methods and protocols for programs to talk with each other.

Example: IMDB has a movie API that allows other developers working on their projects to access their API and get data for all the movies in the IMDB Database.

IMMUTABLE

Immutable means that once the constructor for an object or variable has been executed, that variable can no longer be altered.

Immutability is helpful because it allows you to pass references of objects or variables around and through functions without worrying that someone or something will change the content.

RESTful Architecture

REST, yet another acronym, this time for Representational State Transfer, is a term used to describe an architectural style for creating and accessing API's. REST isn't a tool or technology but rather a philosophy of guiding principles and constraints to develop functionality around HTTP requests.

BONUS COMPARISONS

These are three prevalent interview questions I couldn't leave you without knowing! Study and understand these concepts, and you will be more valuable than the majority of your so-called competition.

INHERITANCE VS COMPOSITION

Inheritance: when the child or subclass is a specialized version of the parent class that "inherits" all of the parent's properties.

eg. Vehicle Class (grandparent) => Car Class (parent) => Toyata Class (child)

Composition: when a class uses other classes to implement its functionality.

eg. Car Class => Wheel Class + Transmition Class + Engine Class + . . .

Inheritance vs. Composition: Inheritance creates the "tightest coupling" of classes which means whenever you change a parent class, it can create many unwanted side effects on all subclasses. Composition, on the other hand, is becoming more and more standard as it allows more flexibility and the ability to change runtime behavior.

OBJECT-ORIENTED PROGRAMMING (OOP) VS FUNCTIONAL PROGRAMMING

Object-Oriented Programming: paradigm primarily consisting of *objects* used to represent things and hold data.

Functional Programming: paradigm primarily consisting of (pure) *functions* for the purpose of maintaining immutability.

OOP vs. Functional Programming: There are pros and cons to both of these programming paradigms. The differences come down to mainly the structure of the code, the use of immutability or not, and the focus on variables & functions or objects & methods.

Object-Oriented languages are better for projects with more *things* than *operations*, and Functional Programming is better for projects with more *operations* than things.

These days, the general thinking is that functional programming is more suitable for the front-end while OOP should govern the back-end. But, at the end of the day, it comes down to preference and use-case (as almost everything in programming eventually does).

SERVER-SIDE RENDERING (SSR) VS CLIENT-SIDE RENDERING (CSR)

Client-Side Rendering: A method which web pages render on the client's browser. When the request for a particular URL is received on the server, it sends back a skeleton page and the JavaScript required for the application to function. The user or client's browser then executes the JavaScript to update the HTML and what is on the screen.

Server-Side Rendering: A method where web pages render on the server instead of the browser. When a request for a particular URL is received on the server, it parses and creates the "finished" page and sends it to the user.

CSR vs. SSR: The primary difference between CSR and SSR is the rendering location of the webpage. CSR works well if your site has lots of interactions or needs a constant data stream. On the other hand, use SSR when SEO is a high priority, you want faster initial loading, and there aren't too many user interactions.

So there you go! With these terms in your arsenal, you are prepared to go into that interview feeling confident and ready for whatever comes your way.

BONUS: Your FREE Learning Checklists & Blueprints

A Quick Favor (Before You Go)

The **Bulletproof Guide to Becoming a PAID Front-End Developer (No Experience Required)** is a highly detailed manual to landing your first job. If you actually read through it, I have to give you props.

If you LOVED it and have some friends who might benefit from it, then e-mail it to them or share it on Facebook. Here's a message that you can copy-paste:

> *"Hey NAME,*
>
> *I've found this fantastic guide on the internet that will help you learn the exact strategies, mentalities, and knowledge you need to become a Front-End Developer and [get paid to code].*
>
> *I think you'll love it because it has a lot of specific stories, checklists, examples, and systems in there that you can*

> *put into action right away!*
>
> *Here's the link*
>
> *Enjoy!*
> *"YOUR NAME"*

Your 6-Month Plan & Study Routine That Practically Guarantees You Will Get a Front-End Developer Job

But just reading the guide won't be enough. If you don't take action, you will stay stuck in the same place and with a feeling of disappointment and regret.

Imagine all of the opportunities that could open up to you if you learned to code and landed your first paid developer position.

How much faster would you live your dream life and buy your dream home? **How would it make you feel if you had a valuable, in-demand skill and recruiters messaged you on the daily with high-paying and lucrative positions?**

Think about it.

Speaking from experience, putting a comprehensive guide like this into action can be difficult. Manuals like this can feel overwhelming at points, and it can be hard to know where to start. And, even if you do begin, it can take up too much time, and not everything applies to your unique situation.

For this reason, I have put together a few **quick-and-easy implementation checklists** you can print out or use to overcome the obstacles that otherwise would prevent you from taking action and becoming the person you wish to be deep down.

Here is what's included in Bonus Cheatsheets and Content:

- A **step-by-step blueprint** that you can use to plan your learning for the first 6-months of this amazing journey.
- **100+ Tools & Technologies** that will help you program more effectively and enhance your coding skills so you can stop having excuses like, "I don't know where to start," "I don't know what tools I need," and "I don't have enough time."
- **Your Detailed Study Strategy for putting this guide into action for** any aspiring developers, future freelancers, or paid professional programmers.

You can find the download link here for all your amazing bonus worksheets.

Thanks for reading,
- Will