A Practical Guide to Web Development

Saad Sheikh

June 16, 2021

Contents

Preface

This document is broken down into several chapters for your convenience. Furthermore, each chapter or section might have certain pointers to help you distinguish their importance for your career path. FE, BE, FS refer to front-end, back-end, and full stack, respectively.

An italicized FE/BE/FS refers to a recommended but optional skill for those paths. For example, "JavaScript - FE, BE, FS" means that as a Front-end developer, you should definitely focus on JS. As a Back-end or Full Stack Dev, you might not need to focus on it as much. This holds for any other italicized text.

Notes or important messages are written in such a box.

0.0.1: Example Code (c)

```c
int main(){
    printf("Example code is written in blocks such as
    ↪ these");
    return 0;
}

main();
```

Resources

0.1 Learn to Code

A. Harvard's CS50 Course

B. Hack Reactor - "The most advanced coding bootcamp. Think like a software engineer."

C. Geeks for Geeks - Highly useful for not only learning to code various languages, but also for acing interview questions.

D. Code Academy - A free coding website with courses for plenty of languages.

E. Free Code Camp - A completely free non-profit code camp that has several large scale projects that help you fill your portfolio.

F. You Don't Know JS - Kyle Simpson's phenomenal book on learning and mastering JS.

G. Dev Docs a great place to find documentation for all sorts of things. Compiled into one website for your convenience.

0.2 Coding Interview Prep

A. Interview Cake

B. Hired in Tech

0.3 Advice and Application

A. A blog post about post boot camp job search

B. A list of companies that are easy to apply to.

C. A reddit post outlining steps for preparing for a tech interview. Useful for junior developers and people who haven't had an interview in a while

D. Front End Developer Interview Questions

E. Leetcode - A resource for algorithm and data structure questions

F. Cracking the Coding Interview Book - The go-to book for technical interview prep. Coderust 2.0 is an alternative resource if you're avoiding CTCI.

Chapter 1

Getting Started

1.1 Before You Begin Learning

It's important to figure out what you want to do and why you want to do it. The following should help figure that out.

Front-end

Front-end refers to front-facing website or application UI. Common technologies for this position include:

- HTML, CSS, Sass, JavaScript, JS Frameworks, TypeScript, DOM(Document Object Model), HTTP, etc.
- Front-end framework like React or Vue.

Depending on where you work, you'll probably need to know some design.

Back-end

A back-end developer usually works server-side, with databases creating REST APIs, web services, etc. Usually need some kind of knowledge about setting up and working with servers.

- Languages: JS, Python, C#, PHP, & more
- Databases: Relational, NoSQL

Full Stack

Both front-end and back-end technologies.

Goals

- Why do you want to be a web-developer?
- Do you want for a large corpo? Smaller firm?
- Freelance? Run your own business? Start-up?
- Build your own apps? Software as a Services?
- Content Creation
- Code as a hobby

There are lots of routes you can take when you're learning to code so figure out what really drives you because the decisions you make really depend on that.

Chapter 2

The Building Blocks

2.1 HTML and CSS - FE, FS, BE

- HTML 5 Page Structure & Semantic Tags

  Header section, try not to use divs, etc.

- Basic CSS Styling

  Colors, fonts, padding and margins, etc

- Positioning

  Absolute, relative, fixed, etc.

- Alignment (Flexbox & CSS Grid)

  At least learn flexbox, if not both.

- Transitions and Animation

  Very important for Front-end devs. A lot of UI's have some kind of animation involved.

- Responsive Design and Media Queries

  Everything you build should look good on all screens.

2.2 Sass - FE, FS

Sass is a CSS preprocessor at gives you more functionality in your styling. Sass offers things like variables, mixins, functions, nesting, etc. While Sass is optional, it is a useful tool. After learning CSS, it is easy to pick-up.

2.3 CSS Frameworks - FE, FS (Choose one)

These UI frameworks offer pre-made classes to create elements on the fly. Menus, lists, alerts, etc. It's important to understand CSS before diving into a framework. Popular frameworks include:

- Tailwind CSS - low-level utility classes. Which means you can create your own cards. On the downside, your HTML has a ton of CSS classes. On the up, it's highly customizable and you don't really need to write any CSS.
- Bootstrap - CSS classes that give you alerts, cards, etc. A popular framework.
- Materialize - Based on Material Design.
- Bulma - Modular & lightweight.

2.4 UI Design - FE

At a larger company, someone else is probably going to be doing the design work. But in general, it's important to know the basics of design. How large margins should be, what types of fonts to use, what sizes, etc.

- Color and Contrast - Make sure text is readable.
- White Space - Spacing between elements.
- Scale - Sizing relative to other elements.
- Visual Hierarchy - Arrange in order of importance.
- Typography - Text typefaces, sizing, etc.

2.5 JavaScript - FE, BE, FS

JS is extremely important for front-end/full stack web developers. It's the language of the client side. If you plan on doing back-end, just learning the basics of JS should be plenty.

- Basics - Variables, arrays, functions, loops, etc.
- DOM and Styling - Selecting and manipulating elements.
- Array Methods - foreach, map, filter, reduce, etc.
- JSON - JavaScript Object Notation
- HTTP Requests - Fetch API - GET, POST, PUT, DELETE

Chapter 3

Deployment

3.1 Basic Front-end Deployment - FE, BE, FS

You should be able to do a basic website or frontend app deployment.

- Static Hosting - Netlify, GitHub Pages, Heroku

  Netlify uses Git so all you have to do is push to your repository and you have continuous deployment using Netflify.

- CPanel Hosting - InMotion, Hostgator, Bluehost The good thing about CPanel hosting is that there is a lot included by default like e-mail hosting, SQL databases, STP accounts, etc. A VPS(Virtual Private Server) is suggested for CPanel and not shared hosting.

Methods of Deploying: There are various ways of getting your files onto your server.

- Git - Continous deployment by pushing to a repo
- FTP/ SFTP - File Transfer Protocol/Secure File Transfer Protocol (Slow)
- SSH - Secure Shell (Terminal)

There are ways of hosting full stack applications - AWS, Digital Ocean, etc.

Some other things you will run into during a basic deployment:

- Domain Names - Namecheap, Google Domains, Enom

  If you use something like Netlify, you can have your main domain name point to Netlify and then setup your e-mail hosting somewhere like Namecheap.

- Email Hosting - Namecheap, Zoho Mail, CPanel.
- Let's Encrypt, Cloudflare, Namecheap

Basically, you want to know how to host a website, connect a domain name, setup a SSL. At this point, you are basically a Foundational Frontend Developer.

3.2 Basics Conclusion- FE, FS

At this point, you should be able to do the following:

- Setup a productive development environment.
- Write HTML, CSS, and JS.
- Use Sass and CSS framework (optional).
- Create responsive layouts.
- Build websites with some dynamic functionality and work with the DOM.
- Connect to 3rd party APIs with Fetch and understand basic HTTP.
- Use Git with GitHub or some other Git repo (Bitbucket, etc).
- Deploy and manage a website or small web app.

If you can consider these skills conquered, you are essentially a Foundational Front-end Developer.

3.3 What Now?

At this point, you can take a couple different routes. You can sharpen your JS skills, start doing algorithms or more advanced JS. If you want to stick to front-end, move to a front-end framework like React. Or you can move to back-end.

Chapter 4

Front-end Superstar

4.1 Front-End Framework - FE, FS (Choose one)

As a front-end developer, you will most likely need to be familiar with a popular front-end JS framework. These allow you to create Single Page Web Apps (SPWAs). The page loads from the pre-built JS from the framework that is then inserted into a single page. Frameworks allow you to have intricate user interfaces that are possible with vanilla JS but harder and more time-consuming.

Working with a framework and a team makes it much easier because you have standards that devs can follow instead of customized JS that you created that no one else understands. You can use any framework on the front-end with any language on the server. i.e. React can be used with AI and back-end can be created in Python/Java/C#.

To pick what to learn, see what is being used by jobs in your area or where you want to work.

- React - Most popular in the industry. Library to which you can add packages to make it function as a framework.
- Vue - A bit easier to learn than React and has a great community.
- Angular - Angular is the hardest to learn of all. It uses TypeScript and is backed by Google. Angular is a full-fledged framework.
- Svelte - Not branded as a framework but as a compiler. But, it allows you to write JS in a clean and effective way just like any other framework. Use for personal projects, not professional.

Each of these framework has it's own ecosystem and part of that system is state management. State management is basically the state of your application - is the menu open or closed? data displayed? State can be local to a specific element or can be global to share across your entire app or UI. Global state managers can help manage global state as it can get messy. These are the following state managers used by different front-end frameworks.

- React - Context API, Redux, MobX

  Context API is built into React and is great for smaller/medium sized apps. If you have different types of data your app holds, you might want to look into Redux. Redux is the most popular 3rd-party state manager for React.

- Vue - Vuex

- Angular - Shared Service, NgRx

  Angular is a larger framework and is more full-featured. It has built-in services to spread state/data across your components.

- Svelte - Context API

4.2 TypeScript - FE, BE, FS

TypeScript is a super-set of JS and is popular on it's own as well as being paired with a FE framework. It's not something you have to learn right away but is pretty popular in the industry. Since JS is dynamically typed, which means that you don't have to define our variables or functions as strings, numbers, booleans, etc.

TS basically is JS but has static types. It helps you find errors before they occur and is really robust. However, it is something else you have to learn and more code to type. Seems like one of those things that you prefer not to do until you start using it and can't type JS any other way. Just like JS, TS can be used with other front-end frameworks and on the back-end if you're using Node.js.

4.3 Testing

Testing is a great thing to have in your toolkit. It helps prevent problems before they happen. This goes for all languages.

- Unit Tests - Individual units like functions or classes.
- Integration Test - Modules tested as a group.
- End-to-End Tests - Test workflow from start to finish.

There are testing frameworks available for different languages. For example, Jest and Mocha for JS and PyTest and Robot for Python.

- Brings a "strict" type system to JS.
- Makes your code more robust and less prone to errors.
- Object-oriented-programming (classes, interfaces, generics, modules).
- Great for larger projects.

> The rest of the front-end section consists of technologies that are trending and becoming more popular. You don't need to focus on these until you get to the point where you're comfortable with a front-end framework. This is because these technologies are part of a framework ecosystem.

4.4 Server Side Rendering **(Choose one)**

Normally, you build a SPWA which runs on the client, browser loads the JS bundle and everything happens on the browser. In this case, if you look at the source code in your browser, you see the JS files that load your application. This method results in web crawlers being unable to see your content. There are ways around this but, using Server side rendering solves this problem by default. Furthermore, it's possible to run a front-end framework on the server. There are advantages to this - better SEO, easy routing, better performance, etc.

- Next.js - React

- Nuxt.js - Vue
- Angular Universal - Angular
- Sapper - Svelte

4.5 Static Site Generators **(Choose one)**

SSG's generate your website pages at build-time as opposed to real-time, making them super fast and secure. This way, there's no generation of HTML or calls to a database, it just responds with the file itself. This makes your website very fast and easy to host. Static site doesn't mean you can't have dynamic functionality or work with data.

- Gatsby - React based.
- Gridsome - Vue based.
- 11ty - JS alternative to Jekyll.
- Jekyll - Ruby based.
- Hugo - Go based.

Just because a SSG is based on a language doesn't mean you need to know that language to use it. Other languages have their own SSGs as well.

4.6 Headless CMS

Back-end only content management system that is commonly used with static site generators.

You or your client can log into an area and create content for your site without touching any code. Wordpress and Drupal inlude interfaces for a front-facing website and a part where you login and add data. A headless CMS works in a similar fashion but doesn't have that front-facing webpage. It's used strictly for the data part and you can add whatever you want for the front-end. A popular option is to use Gatsby(static site generator) and connect that to a headless CMS.

There are some headless CMSs that are hosted elsewhere and have premium packages. Some charge you once you go over a quota but can be generous and allow you to host a personal blog free-of-charge.

- Strapi

  Open-source, node.js headless CMS that you host yourself. No subscription, free, and easy to use.

- Sanity.io
- Contentful
- Prismic
- Wordpress

  Wordpress can also be used as a headless CMS because it has a REST API built into it.

4.7 The Jamstack - JavaScript, APIs and Markup

The past few sections and technologies can be fit into the Jamstack. The Jamstack is a web architecture with high performance, security, and scalability at a low cost with a great dev experience. These aren't new concepts but what is new is the way we use them.

Unlike the MERN stack (MongoDB, Express, React, Node.js), the Jamstack has a plethora of technologies to choose from including:

- Static Sites
- Markdown
- Serverless

  Serverless is a big trend which means you don't have to manage your own server. Your cloud provider runs the server and allocates machine resources on-demand.

- Headless CMS for Content
- Hosting with services like Netlify, AWS (also offers serverless architectures).

4.8 Front-end Superstar Conclusion - FE, FS

- Build apps and interfaces with a frontend framework
- Work with compononent and global state
- Connect to backend JSON data and integrate into your apps
- Write and test clean and efficient code
- Use TypeScript to write more robust code
- Server side rendering
- Static site generators/Jamstack

Chapter 5

Back-end/Full Stack

5.1 Server Side Language - BE, FS (Choose one)

The back-end/server focuses on data, modeling, and HTTP requests/responses. A server side language is needed for back-end/full stack development.

- Node.js (JavaScript)

  If you already know a good amount of JavaScript, Node.js is a pretty good option. However, it really comes down to what you want to do and what you're interested in.

- Deno (JavaScript)

- Python

  Python is a great language to learn in general. Good for AI and machine learning but also used in web dev.

- C#

  Check out what companies in your area are using. If they're using C# you might want to learn it.

- GoLang

  Another language created by Google. Similar to C but has garbage collection and memory safety which makes it easier to learn than C. Often used to program large-scale servers. Often used in the big business enterprise world.

- Ruby

  If you choose Ruby, you're probably going to be using Ruby on Rails. RoR is great for rAPId development and has lots of integrated tools. Ruby's not as relevant as it once was but before you set it aside for good, check out jobs in your area.

- PHP

  There's still lots of websites that use PHP. If you're freelancing or creating small-scale web apps, PHP can be a really good option. It uses Wordpress.

- Java

  Probably the most relevant in the big business world.

- Kotlin

  Kotlin is being used in a lot of places that Java used to be used like Android web development.

5.2 Server Side Framework - BE, FS **(Choose one)**

A framework is usually used in back-end web development. Before you jump into a framework, you should learn a good amount of a language - fundamentals, ecosystem, package managers, etc, and then move on to a framework. Most of the time, you're going to be using a framework and not building everything from scratch.

- Node - Express, Koa, Nest, Loopback

  Express is the most popular. Pretty minimalistic. Allows you to handle and customize the HTTP request/response cycle pretty easily, create middleware, add routing endpoints, etc.

  Koa is very similar to Express. Nest.js is a larger, more fully-featured framework for Node that uses TypeScript by default.

  Loopback allows you to easily create REST APIs with a GUI tool

- Python - Django, Flask

  Django is very full-featured and high level. Admin area built-in. Very specific ways to do things but get a lot of bang for your buck. Flask is

low-level and minimalistic which gives you a lot of freedom to do things. Similar to Express.

- PHP - Laravel, Symfony, Slim

  Laravel is pretty elegant and a good framework. Laravel is actually based off of Symfony. Symfony is a little harder to learn than Laravel.

- C# - ASP.NET

  ASP.NET is very powerful and uses the .NET framework. Used a lot in the industry.

- Java - Spring MVC

- Ruby - Ruby on Rails, Sinatra

  Ruby on Rails is another MVC framework. Has lots of tools to allow you to build quickly.

- Kotlin - Javalin, KTor

5.3 Database - BE, FS **(Choose one)**

Back-end/Fullstack devs work with databases and ORM/ODMS.

A huge part of working on the back-end of a web application or creating APIs for services is working with data and that usually includes working with a database of some kind.

There are many types of databases, the most popular are relational databases or SQL (structured query language) databases and NoSQL databases which also have a subset of database types. For example, MongoDB is a document database.

- PostgresSQL Relational database. Popular with PHP but great with lots of languages.

- MongoDB

  Document database. Popular with JS stacks like MERN (MongoDB, Express, React, Node.js). Very JS/JSON like, flexible, and scalable.

- MySQL

- MS SQL Server
- Firebase

  An entire platform. Data and file storage, authentication, easy API. A combination of something like React and Firebase might not be great for giant, scalable applications but might be a good solution for small, personal projects or small business solutions.

- Elasticsearch

  Search engine based database. Provides full text search with an http interface and schema free JSON documents.

Object relational mappers/ object data mappers give you an abstract layer for your application to interact with your database. A lot of times, you don't actually have to write SQL queries, you just have an API to interact with your database.

- Mongoose

  Popular with Node.js and MongoDB

- Sequelize

  Also used for PostgresSQL and should be usable with multiple with different SQL languages.

- SQLAlchemy

  A ORM for python.

- Doctrine
- Eloquent

  Used by Laravel.

Obviously, there are a lot of databases. Just do your research on what's available for your language/framework.

5.4 GraphQL

GraphQL is basically a data query language for APIs. In many cases we create and consume data using restful APIs where send a specific HTTP request to a specific endpoint to get a bunch of data. GraphQL is used in a similar fashion but, instead of having all these different endpoints that return all this different data (even stuff you don't need), GraphQL provides a single endpoint and you can make queries based on the data that you need, so you don't have to get everything.

For example, if it's an API of user information and all you need is the first and last name, you can make a request or a query to get just the first and last name. The queries that you send to a GraphQL server are formatted similar to JSON which is the format of the response. The query looks like the response. If you already know JavaScript and JSON, making and receiving queries is really easy.

- Send a query (similar to JSON) to your API and get exactly what you need.
- Setup a GraphQL server and query using a client like Apollo (can be implemented with Node.js and other languages).
- More targeted than a REST API and saves a bunch of trips to the server.
- Easily use with React and other frameworks.

5.5 Socket.io and Real-Time Technologies

Real-time applications are becoming more popular. Socket.io allows real-time, bidirectional communication.

In a lot of cases you have a client-side and you send HTTP requests to the server where the server is Express, Django, Laravel, etc, and you have to make a separate request for each task.

Using websockets (Socket.io) you have a constant bi-directional communication through events. There's all sorts of apps and services that you can create using websockets.

- Instant messaging and chat.

- Real-time analytics.
- Document collaboration.
- Binary streaming.
- Much more...

5.6 WordPress Development

WordPress is still used, especially in the small business world. If your goal is to work at a large corporation, you can probably skip on WordPress. If you want to do freelancing for the mom and pop shop down the road, WordPress may be a pretty good choice.

- Setup websites quickly.
- Give your clients complete control.
- Tons of plugins to add functionality.
- Create custom themes and plugins.
- Wordpress can be used as a headless CMS since it has it's own REST API.

5.7 Deployment, Servers, and DevOps - BE, FS

Deploying apps to production, monitoring, security, containerization/virtualization and more.

As you become a more advanced developer, you're probably going to learn about DevOps, deployment strategies, and hosting platforms. Hosting a full stack app or an API is a bit more difficult than a front-end app or static website. You need to deal with servers, configurations, file storage, databases, and more.

- Hosting platforms - Heroku, Digital Ocea, AWS, Azure

  Heroku is pretty good for small apps and personal projects. Digital Ocean is useful larger apps built to scale. Digital Ocean is a cloud hosting company and it gives you full control over everything including the operating system you want to install. It takes a bit more knowledge than something like Heroku which is basically a platform as a service. Where Heroku is setup and ready for you to use, Digital Ocean is an empty computer that is ready for you to use. AWS is a very popular option for large scale options. Azure is a pretty popular Microsoft platform.

- Web Servers - NGINX, Apache

  You need some kind of webserver to serve your back-end code. NGINX and Apache are really popular. NGINX is easier to configure and works well with Node.js and PHP. However, if you use a platform such as Heroku, you probably don't have to do much configuration.

- Containers - Docker/Kubernetes, Vagrant

  If you're doing DevOps, you definitely want to familiarize yourself with Docker. Gives you a way to run your applications in a virtual environment called a container. Useful for teams so your app runs in the same environment regardless of its physical location.

  Kubernetes is often used with Docker. It's a system for automating the management of containers. It's extremely popular in the DevOps world.

- Image/Video - Cloudinary, S3

  For image and video hosting you can use a local method or you can choose to upload to something like Cloudinary or Amazon S3. Cloudinary has all types of media APIs that let you manipulate and optimize your media.

- CI/CD - Jenkins, Travis CI, Circle CI

  Continuous integration and continuous deployment is also something you'll need if you go into DevOps. Jenkins and Travis CI are popular when it comes to facilitating and automating continuous automation, testing, and deployment. As a web developer, you most likely won't have to come close to mastering these technologies. It's more on the DevOps side of things but it's good to get your feet wet.

5.8 Conclusion: Full Stack Developer - BE, FS

- Comfortable with both building front-end UIs and servers.
- Know a server-side language/technology (framework).
- Can work with and structure databases, work with ORMs/ODMs.
- Understand HTTP and Create RESTful APIs.
- Can successfully deploy full stack projects.
- Very comfortable with the terminal - navigating your system, using GIT, working with certain CLIs, network commands, etc.

Chapter 6

Extras

6.1 Mobile Development (Choose one)

More and more web developers are getting into mobile development with web-related technologies.

- Flutter/Dart

  Open-source, user-interface, software developer kit (SDK). Uses the Dart Language. If you know JS, Dart is pretty easy to pick up.

- React Native

  If you already know React, might want to get into React Native. React Native is a framework for building native mobile apps using React.

- Ionic

  Javascript Based.

- Xamarin

  Allows you to build mobile apps with C#.

- Kotlin

  Allows you to build native Android apps.

- Swift

  Allows you to build native iOS apps.

6.2 Progressive Web Apps (PWA)

Regular web apps built with HTML, CSS, and JS but run and feel like a native mobile app. Not just a responsive website but with a completely native feel as far as experience, layout, and functionality regardless of the device.

- Built for all screen sizes.
- Offline content/Service Workers

  Offline content managed by service workers in the browser. Look into service workers if you want to get into PWAs.
- HTTPS
- Native experience (fast, engaging, splash screens, installable, etc).

6.3 Desktop Apps with Web Technologies

There are different web-technologies that can be used to create desktop apps.

- Electron

  The most popular. You can use JS to create desktop apps. Desktop apps built with Electron include Slack, Atom, VSCode.
- NW.js
- Python and Tkinter

6.4 AI/Machine Learning

A bit beyond the scope of web dev but are special situations where they can be beneficial to learn. Machine learning can be especially useful for Python developers.

- Automation and Tools.
- Machine Learning APIs
- Understand user behavior/engagement/analytics
- Create code - GPT-3

6.5 Web Assembly

Efficient, low-level bytecode for the web. It's an "improvement" to JS, not a replacement. Meaning it can do much more powerful things in specific areas.

- Create extremely powerful web apps (games, video/image editing, etc)
- Can use languages such as C++ and Rust to compile to WASM.
- AssemblyScript is a variant of TypeScript and it makes it easy to compile to WASM without learning a new language.

6.6 Algorithms

Algorithms may not seem productive however, they help your logic and critical thinking skills in ways you can't imagine.

- Beginner algorithm questions: FizzBuzz, string reversals, array chunking, palindromes, anagrams, max character, etc.
- Popular challenge websites: Codewars, Project Euler, Coderbyte.

6.7 Data structures (Primitive and non-primitive)

Organizing and managing data efficiently so that we can perform specific operations efficiently.

- Popular data structures: Array, linked list, queue, stack, tree, graph, hash table.

6.8 Software Design Patterns

As you become a more advanced developer and write more intricate code, you might want to look into more specific software design patterns which are general reusable solutions to commonly occurring problems. Design patters are language agnostic, just like data structures and algorithms.

- Singleton

- Facade
- Bridge/adapter
- Strategy
- Observer

Chapter 7

What Now

- Create a learning plan based on what you want to do.
- Learn the fundamentals and necessary technologies.
- Watch tutorials/courses/etc but be sure to create your own personal projects based on what you learn.
- Create a portfolio.
- Start applying for jobs or finding clients.

7.1 Learning Plan

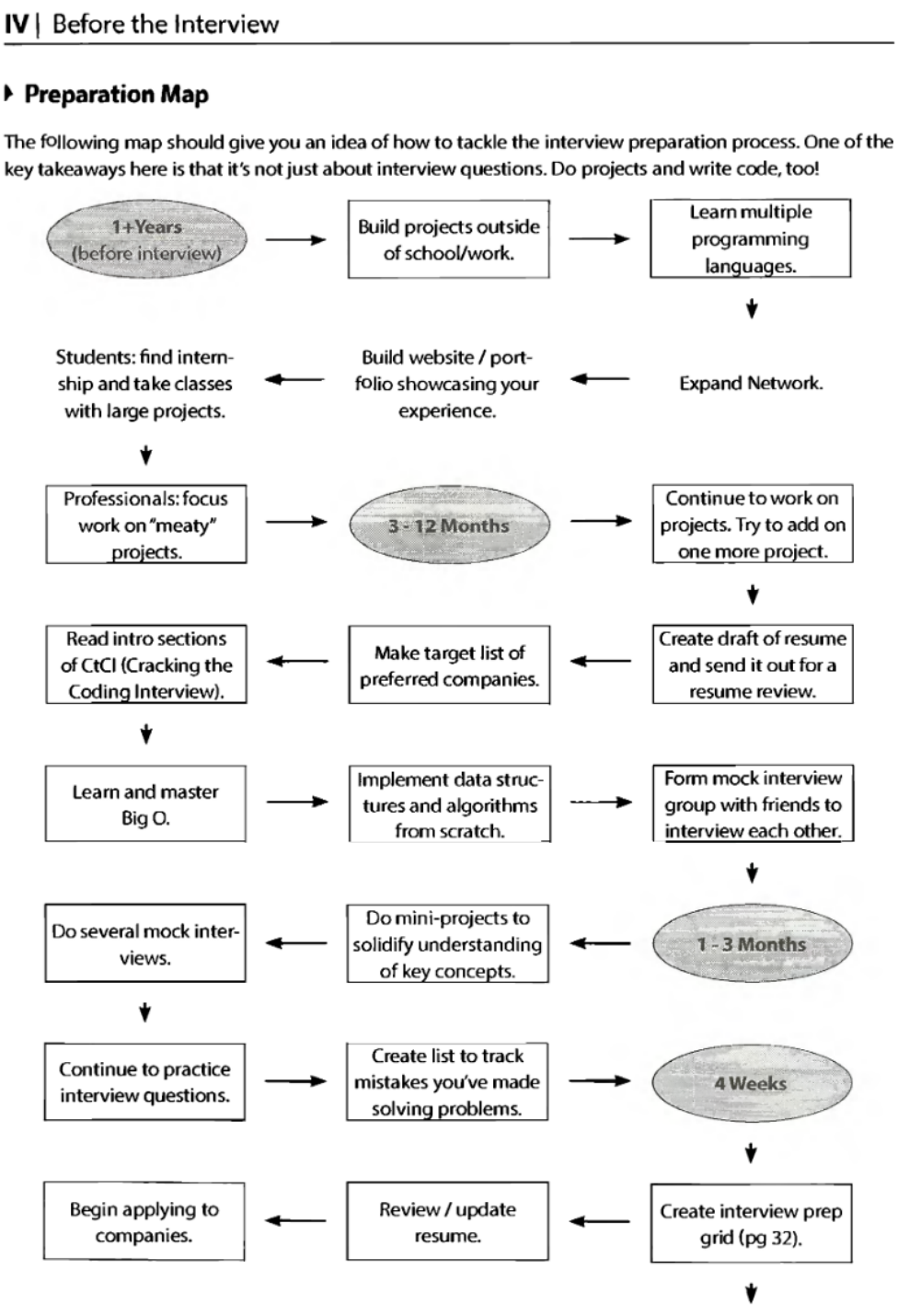

Figure 7.1: Roadmap Part 1

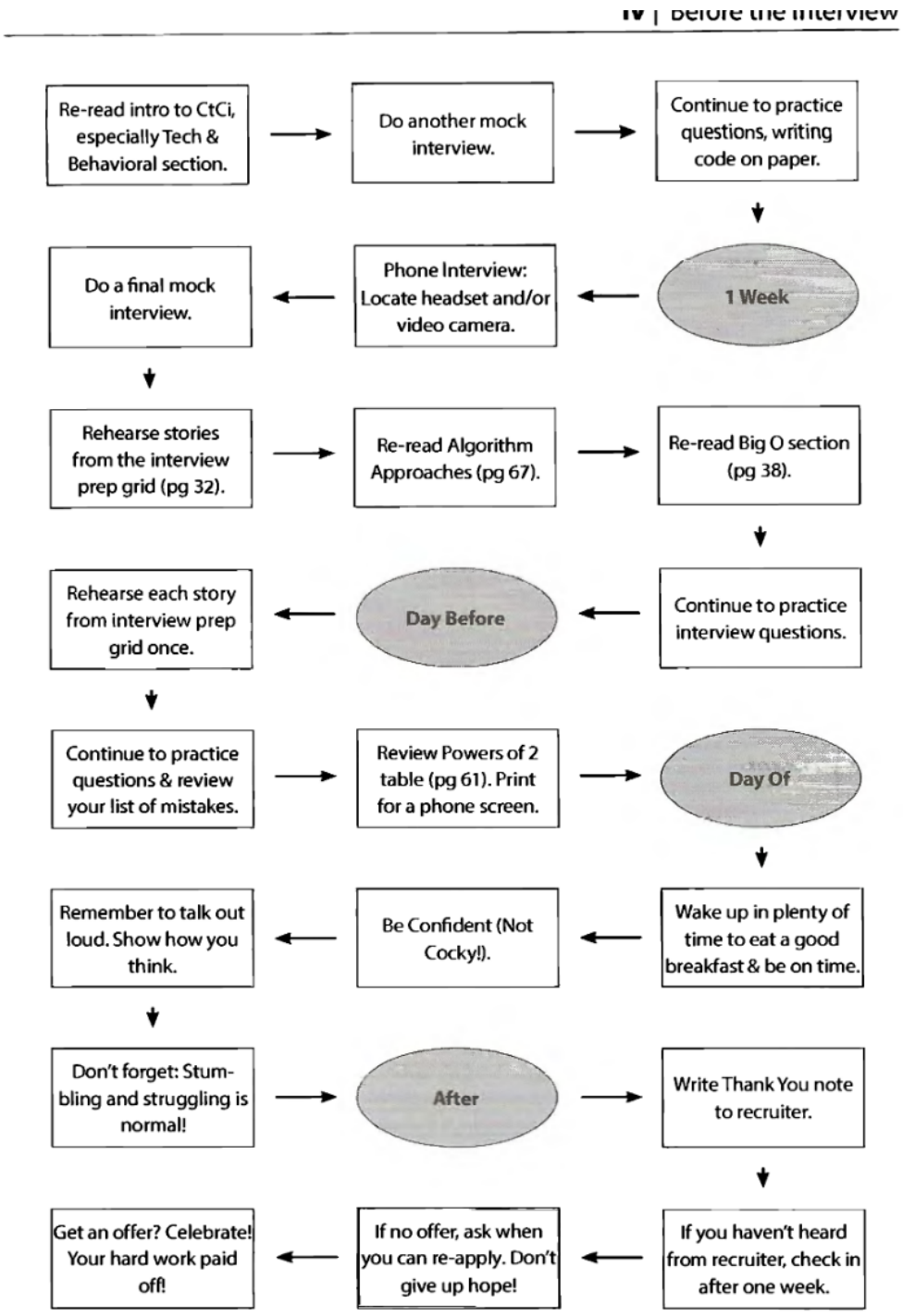

Figure 7.2: Roadmap Part 2

A. Learn the building blocks.

   - Code academy courses: Responsive Web Design, JS Algorithms and Data Structures, front-end development libraries.

B. Be a front-end superstar.

C. Follow Crack the Coding Interview roadmap.

7.2 Example Job Postings

Front-End

Qualifications and Skills

- Collaborate with back end, product and design teams
- Own responsibility for overall front end architecture
- Communicating tradeoffs with stakeholders, executives, sales/marketing.
- While 100% remote role, availability for meetings U.S. Pacific Time.

Responsibilities

- Minimum of 4 years of software architecture and development experience with at least 1-2 years of lead or principal responsibilities in a data-centric enterprise platform.
- Expert experience with Javascript, GraphQL, React, and React Hooks
- Substantial startup experience and/or a track record of success in a fast-paced environment.
- You make smart architectural decisions, build efficient systems, and have know-how to test and monitor the platforms.
- A track record of taking measured risks and providing robust backup and rollback plans.
- A track record of moving fast, while proactively managing technical debt and continuous refactoring.

Back-End

Qualifications and Skills

- Minimum of 5 years of software architecture and development experience with at least 1-2 years of lead or principal responsibilities in a data-centric enterprise platform.
- Expert experience with Python OR Go
- Significant experience in search, API data services, and performance optimization
- Substantial startup experience and/or a track record of success in a fast-paced environment.
- You make smart architectural decisions, build efficient systems, and have know-how to test and monitor the platforms.
- A track record of taking measured risks and providing robust backup and rollback plans.
- A track record of moving fast, while proactively managing technical debt and continuous refactoring.
- Bonus for experience with GCP, BigQuery, Redis, jhub/Jupyter Notebook, web scraping tools and Kubernetes

Responsibilities

- Responsible for overall architecture including API design
- Search algorithms and performance, e.g. elastic search, entity & document search.
- Data schemas and persistence, data importers.
- Machine learning processing pipeline.
- Operations: Kubernetes, Monitoring & Alerting, GCP Resources
- Dev tools: Github actions, Makefiles, Release process/scripts

- Communicating tradeoffs with stakeholders, executives, sales/marketing.
- While 100% remote role, availability for meetings U.S. Pacific Time.

Full Stack

Essential Duties and Responsibilities

- Lead and participate in requirements gathering discussions and document application specifications.
- Architect, develop, test, deploy, maintain full-stack customer-facing cloud-based applications (front-end and back-end).
- Develop and manage well-functioning databases and applications.
- Deliver superior performance for customers ensuring all applications are responsive and fast.
- Ensure and manage multiple integrations between O&O applications and 3rd party SaaS and application platforms.
- Write clear, and thorough technical documentation for all developed applications.
- Test, troubleshoot, debug and upgrade application software and code as needed.
- Manage individual project priorities, deadlines, and deliverables in a flexible work environment.
- Participate with other technologists and developers to develop forward-thinking strategy, plans, and software throughout Company.
- Work closely with the marketing and research and development to ensure application user-experience meets customer needs.

Education and Experience

- Bachelor's Degree in Computer Science, or equivalent work experience
- At least 5 years of programming experience through work or personal projects with a strong working knowledge of:
  - AWS Lambdas, Dynamo, Beanstalk, ElastiCache, ElasticSearch, CloudFront etc.
  - Relational/Non-Relational DB experience
  - Node.js
  - React.js
  - Python
  - Django
  - Flask
  - JavaScript
  - User-authentication

Chapter 8

Languages

8.1 HTML

Input Labels

It is considered best practice to set a for attribute on the label element, with a value that matches the value of the id attribute of the input element. This allows assistive technologies to create a linked relationship between the label and the related input element. For example:

8.1.1: Labels and inputs (html)

```
<input id="indoor" type="radio" name="indoor-outdoor">
<label for="indoor">Indoor</label>
```

Alternatively, we can nest input elements within label tags.

8.1.2: Alternative Code (html)

```
<label for="indoor">
    <input id="indoor" type="radio"
    ↪ name="indoor-outdoor">Indoor
</label>
```

All related radio buttons should have the same name attribute to create a radio button group. By creating a radio group, selecting any single radio button will automatically deselect the other buttons within the same group

ensuring only one answer is provided by the user.

8.1.3: Label Groups (html)

```
<label for="indoor">
    <input id="indoor" type="radio"
    ↪ name="indoor-outdoor"> Indoor
</label>
<label for="outdoor">
    <input id="outdoor" type="radio"
    ↪ name="indoor-outdoor"> Outdoor
</label>
```

Here, you have two radio inputs. When the user submits the form with the indoor option selected, the form data will include the line: indoor-outdoor=indoor. This is from the name and value attributes of the "indoor" input.

If you omit the value attribute, the submitted form data uses the default value, which is on.

8.1.4: Input Values (html)

```
<label for="indoor">
  <input id="indoor" value="indoor" type="radio"
  ↪ name="indoor-outdoor">Indoor
</label>
<label for="outdoor">
  <input id="outdoor" value="outdoor" type="radio"
  ↪ name="indoor-outdoor">Outdoor
</label>
```

Then any markup with the content of the page (what displays for a user) would go into the body tag.

Metadata elements, such as link, meta, title, and style, typically go inside the head element.

Here's an example of a page's layout:

8.1.5: Page Layout (html)

```
<!DOCTYPE html>
<html>
  <head>
    <meta />
  </head>
  <body>
    <div>
    </div>
  </body>
</html>
```

8.2 CSS

Inline CSS

Here's how you would set your h2 element's text color to blue:

8.2.6: Inline CSS (html)

```
<h2 style="color: blue;">CatPhotoApp</h2>
```

It is a good practice to end inline style declarations with a ; .

Global CSS

8.2.7: Changing global styles (html)

```
<style>
  h2 {
    color: red;
  }
</style>
```

Note that it's important to have both opening and closing curly braces ({ and }) around each element's style rule(s). You also need to make sure that

your element's style definition is between the opening and closing style tags. Finally, be sure to add a semicolon to the end of each of your element's style rules.

CSS Classes

8.2.8: CSS Clss Declarations (html)

```
<style>
  .blue-text {
    color: blue;
  }
</style>
```

You can see that we've created a CSS class called blue-text within the <style> tag. You can apply a class to an HTML element like this: <h2 class="blue-text">CatPhotoApp</h2>. Note that in your CSS style element, class names start with a period. In your HTML elements' class attribute, the class name does not include the period.

Font Degradation

For example, if you wanted an element to use the Helvetica font, but degrade to the sans-serif font when Helvetica isn't available, you will specify it as follows:

8.2.9: Font Degradation (css)

```
p {
  font-family: Helvetica, sans-serif;
}
```

Generic font family names are not case-sensitive. Also, they do not need quotes because they are CSS keywords.

id Attribute

In addition to classes, each HTML element can also have an id attribute.

There are several benefits to using id attributes: You can use an id to style a single element and later you'll learn that you can use them to select and modify specific elements with JavaScript.

id attributes should be unique. Browsers won't enforce this, but it is a widely agreed upon best practice. So please don't give more than one element the same id attribute.

Here's an example of how you give a h2 element the id of cat-photo-app:

8.2.10: Setting an id Attribute (html)

```
<h2 id="cat-photo-app">
```

Styles Using id

One cool thing about id attributes is that, like classes, you can style them using CSS.

However, an id is not reusable and should only be applied to one element. An id also has a higher specificity (importance) than a class so if both are applied to the same element and have conflicting styles, the styles of the id will be applied.

Here's an example of how you can take your element with the id attribute of cat-photo-element and give it the background color of green. In your style element:

8.2.11: id Attribute Style (css)

```
#cat-photo-element {
  background-color: green;
}
```

Alignment, margin, and Padding

Three important properties control the space that surrounds each HTML element: padding, border, and margin.

An element's padding controls the amount of space between the element's content and its border.

An element's margin controls the amount of space between an element's border and surrounding elements.

If you set an element's margin to a negative value, the element will grow larger.

Clockwise Notation for Padding

Instead of specifying an element's padding-top, padding-right, padding-bottom, and padding-left properties individually, you can specify them all in one line, like this:

```
padding: 10px 20px 10px 20px;
```

These four values work like a clock: top, right, bottom, left, and will produce the exact same result as using the side-specific padding instructions.

Measurement Units

The two main types of length units are absolute and relative. Absolute units tie to physical units of length. For example, in and mm refer to inches and millimeters, respectively. Absolute length units approximate the actual measurement on a screen, but there are some differences depending on a screen's resolution.

Relative units, such as em or rem, are relative to another length value. For example, em is based on the size of an element's font. If you use it to set the font-size property itself, it's relative to the parent's font-size.

This article written by Bert Bos, a member of the W3C board, goes more in-depth about absolute and relative units in CSS. According to Bert, the uses of each unit differs by medium.

	Recommended	Occasional Use	Not Recommended
Screen	em,px, %	ex	pt, cm, mm, in, pc
Print	em, cm, mm, in, pt, pc, %	px, ex	

Table 8.1: Recommended Usage of Units

Absolute units (cm, mm, in, pt, and pc) should be avoided in uses other than print. The reason being that an inch on a desktop screen looks very small when compared to an inch on a mobile phone. In this case, using a relative unit (em, ex, px, %) is better.

The magic unit of CSS, the px, is a often a good unit to use, especially if the style requires alignment of text to images, or simply because anything that is 1px wide or a multiple of 1px is guaranteed to look sharp.

But for font sizes it is even better to use em. The idea is (1) to not set the font size of the BODY element (in HTML), but use the default size of the device, because that is a size that the reader can comfortably read; and (2) express font sizes of other elements in em: H1 font-size: 2.5em to make the H1 $2\frac{1}{2}$ times as big as the normal, body font.

The only place where you could use pt (or cm or in) for setting a font size is in style sheets for print, if you need to be sure the printed font is exactly a certain size. But even there using the default font size is usually better.

The px unit thus shields you from having to know the resolution of the device. Whether the output is 96dpi, 100dpi, 220dpi or 1800dpi, a length expressed as a whole number of px always looks good and very similar across all devices. But what if you do want to know the resolution of the device, e.g., to know if it is safe to use a 0.5px line? The answer is to check the resolution via Media Queries. Explaining Media Queries is out of scope for this article, but here is a small example:

8.2.12: Media Query Example (css)

```css
div.mybox { border: 2px solid }
@media (min-resolution: 2dppx) {
  /* Media with 2 or more dots per px */
  div.mybox { border: 1.5px solid }
}
```

Overriding CSS

A class overrides the body's style properties. Class B declared after class A overrides the overlapping effects of class A regardless of order of use in html. For example:

8.2.13: Overriding CSS (html)

```
<style>
  body {
    background-color: black;
    font-family: monospace;
    color: green;
  }
  .pink-text {
    color: pink;
  }
  .blue-text {
    color: blue;
  }
  #orange-text {
    color: orange;
  }
</style>

<h1 class="pink-text blue-text" id="orange-text">Hello
↪ World!</h1>
```

Because .blue-text was declared in <style></style> after .pink-text, it takes priority. However, the id will take precedence over the class and override it's effects. Furthermore, inline styles override all stye declarations. Lastly, the most powerful overrides is that of !important. Adding this to any CSS declaration ensures that that style is implemented.

Variables

Create a custom CSS Variable

To create a CSS variable, you just need to give it a name with two hyphens in front of it and assign it a value like this:

8.2.14: Setting a Variable (css)

```
--penguin-skin: gray;
```

This will create a variable named –penguin-skin and assign it the value

of gray. Now you can use that variable elsewhere in your CSS to change the value of other elements to gray.

Fallback Values for Variables

Despite the following code having a typo, the browser displays a black color for the background because of the fallback.

8.2.15: Fallbacks (css)

```
.penguin {
  --penguin-skin: black;
  --penguin-belly: gray;
  --penguin-beak: yellow;
}

.penguin-top {
  top: 10%;
  left: 25%;

  background: var(--pengiun-skin, black);
}
```