

Guide: How to Become a Web Developer

[resource-wiki](#) [guide](#) 12 votes

galinapodst Codecademy Team Member

Oct '19



Guide: How to Become a Professional Web Developer

About this Guide

I'm a developer and I've been tutoring new coders for years, I made this guide to help people to figure out how they can chart a path to **becoming a web developer**. Wanna know more about these guides (there are others), me, and why I made them? See [here](#).

And, watch this video! [How long does it take to become a web developer](#)

Learning the Right Skills

First, check out my guide on [getting started](#), that has things like picking a path and how long to expect it'll take to learn.

Then, watch [this](#) video on how to get a job as a developer. It's a high level, 101 sort of thing but will help you to get some context. Among other skills, that video says (and it's correct) that as a web developer you'll need the following, in this order:

1. [HTML](#) (course)
2. [CSS](#) (course)
3. [JavaScript](#) (course)
4. Command Line, Git and GitHub... there's a lot of topics here in other developer tools, check out [this resource guide](#) for details as if I added it all in here

One of the key things that you'll need in order to get a job isn't just to know a language or technology, it's *how to use it*. After all, memorizing a Russian dictionary doesn't mean you can speak Russian. That's what employers are going to want to see – can you do the job? Can you build great websites and web apps for them? You show that by *building a portfolio*. [Watch this](#) to learn how.

If you're using Codecademy Pro, they have portfolio projects in courses as you go, tailored to what you've learned so far so you practice a bit and stretch yourself a bit. Whether you do or you don't have Pro, you should be building your own projects. One thing I'd recommend is building projects as you go that use what you've learned. These can be little mini-projects like something simple in Codepen or on your own computer, and/or you could build a few big things as you go. For example, build your own personal "about me" site and add to it a little bit each time with everything new that you learn, e.g. you've just learned HTML forms, how can you use that in your personal page... maybe add a "contact me" section! At this stage, worry more about the practice than about the end product or showing this to an employer, you'll be able to polish things up or build a new job-ready portfolio towards the end.

As soon as you learn GitHub, start using it as much as possible as you keep learning, e.g. with uploading and maintaining versions on your projects. This will not only give you Git & GitHub practice, but it'll look great to a future employer. Check out my [other guides](#) for more detail.

When you run out of tutorial content, and you will, you'll wonder "okay, what next." The big thing to do is to keep coding in building out projects. Use documentation, Google, and ask for help on what you don't know in order to finish your portfolio projects. This will be one of the most challenging parts of your development, but also the most rewarding and *closest to the finish line*.

Front-End, Back-End, or Both?

Watch [this](#) video for more info on front-end development and [this](#) video on back-end. You will have noticed by now that my starter list above essentially avoids back-end (even though you *can* use JS server-side). Even if you want to focus on back end and not front-end, or vice-versa, to be effective you'll need to know at least a little of both, and even for people who want to do back-end, starting with some front-end is usually a better path. In essentially all web development jobs, HTML, CSS, JS, CLI, Git, and GitHub are involved as the first three are part of essentially every company's tech "stack" and the latter are vital developer tools. A "stack" is a combination of technologies, languages, frameworks, and tools that make up a software product, and those stacks vary from company to company, project to project. Read [this](#) for more info.

Keep in mind what stacks *your* company or *your* dream job uses and what might be right for *you*, not just what's popular. Seeing what pops up in job openings that appeal to you is a great and continually up-to-date way to find out what to learn, but keep in mind that a lot of job postings are wish lists and not "real" requirements (read [this article](#)). You can get a job if you have a niche (read [this](#) to find yours) and can prove you can get the job done with a portfolio and good interview performance.

If you know a lot of both front-end and back-end and can do both, you're what's known as a "full stack" web developer. You don't need to *be* full-stack, as cool as it sounds, it's fine to specialize and it's unrealistic to be an expert at everything ([this](#) article is good at explaining why), but you at least need to have *some* skills on both sides because they fit together. Though you can learn languages in isolation, in order to actually build a website or web app you'll need to tie together and use multiple technologies. Pretty much every beginner web developer starts with HTML, CSS, JS in part because you'll need them to finish essentially any web project.

Front-End Technologies

If you're focusing on front-end, take the [Sass course](#) after you've learned CSS. The free courses on [making](#) and [deploying](#) a website, in that order, will also be very useful after learning CSS.

You're going to learn a lot of JavaScript to do front-end, and you're not alone (read [this](#) article to learn why). JavaScript has different frameworks (read [this](#) for more info), and libraries (learn about libraries [here](#)) which are basically tools that can help you to speed up your coding process dramatically. There are many options, and you can't (or shouldn't) learn them all, so pick one – read [this](#) for some guidance. I think everyone should learn a bit of jQuery (free course [here](#)) *after* learning some "vanilla" (library/framework-free) JavaScript because it's got fantastic "bang for your buck" in the cool things you can do but with minimum effort in learning and using it. Beyond jQuery, in 2019 the top libraries or frameworks to learn are, in order of popularity (click the link for the course):

1. [React](#)
2. [Angular](#)
3. [Vue](#)

Angular is seen all over the place but generally getting less popular and though lots of developers love Vue, not so many projects use it yet. Different technologies have communities associated with them, and that community is a vital part of being able to use that language

(it's almost impossible to code without the coding community), so the popularity of that tech and thus the size of the community is a big selling point. If you just want to save time and pick as few courses as possible, learn jQuery and React. Codecademy has courses in all of them, so you can always try them all and see what you prefer. Though when I wrote this, Vue was newly released and so Pro-only, you can finish it in the 7 day Pro trial that you get with new accounts.

This all makes a front-end path look something like this:

HTML -> CSS -> Sass -> Make & Deploy a Website -> JavaScript -> jQuery -> Command Line -> Git -> GitHub -> React

Back-End Technologies

There are more options with back-end technologies than front-end ones, and though you will wind up specializing in one over the others, and though you can and should do some research on what's best for you, you shouldn't worry too much about getting the right one from day one.

Server-side technologies have more in common than you might think. An analogy might be to how languages that evolved from the same ancestor are very similar. Latin turned into French, Italian, Spanish, and Portuguese so if you've learned one of them you have a huge leg up on learning or using the other. A lot of tech languages are like that, sharing common principles like object-oriented programming (if you're unfamiliar with the term, watch [this](#) from the Coding Train, a great YouTube channel), so you don't have to rewire the way you think to switch languages, you can start with learning the differences and nuances. That's why as a beginner you shouldn't worry too much about picking the *right* language, because switching isn't that hard and may actually make you a better programmer. So, don't stress. Even if you change your mind later, it will *not* have been wasted time to learn a language you don't eventually use, especially when Codecademy courses are relatively short. If your research doesn't give you a strong answer or if you just want to get stuck in, just pick one or sample a few and see what you like best.

Quick “Bonus” Recommendation: SQL and Database Management

If you're learning any of the below, you'll find SQL is quick and easy to pick up. It's a technology that will help you a lot with pulling and manipulating data, which isn't just a data analysis thing – it'll be really helpful to you on the job and is arguably a prerequisite. Learn it quickly, stick it on your résumé, and thank me later. Click the links for free courses.

1. [SQL](#)
2. [SQL Data Transformation](#)
3. (optional extra bonus) [SQL for Business Metrics](#)

Ruby / Rails

My bootcamp taught Ruby and Ruby on Rails, with a path of HTML -> CSS -> JavaScript -> jQuery -> React -> Ruby -> Rails. Why? Ruby and Rails were literally designed to be fun to use and (relatively) easy to learn, so a lot of people learn it *and* a lot of companies use it. That philosophy and community is a major part of why Ruby and Rails devs love it so much (read the story [here](#)). Take these free courses in this order to learn Ruby and Rails:

1. [Ruby](#)
2. [Rails](#)
3. [Rails Authorization](#)

Alternatives

As time has gone on, “full-stack JavaScript” (you can use JavaScript server-side!) and even Python for web development have become more and more popular in bootcamps too. Even though most that I know of still teach Ruby / Rails, part of that is because that's how they've been running for so long. The main reason for going full-stack JS is that it means you don't

have to learn and use as many new languages. The big reason for Python is that it's so incredibly popular (read [this](#) for why), it's arguably the world's most popular programming language and / because it's versatile.

Outside of the bootcamp world, PHP is still seen all over the place, particularly in bigger, not exactly brand-new companies or anything requiring WordPress (which is built using PHP) but it's not usually the language of choice for new projects these days. PHP might be needed for many jobs now, but it will probably not stick around forever. Because of this, Codecademy took down their PHP course entirely last year, so if anyone has some good alternative recommendations on PHP or WordPress drop them below or if you're a super user just update this guide to suit!

Full-Stack JavaScript

If you're doing full-stack JS, make extra sure you've taken the bonus SQL content above. Then take these next two free courses.

1. [ExpressJS](#)
2. [NodeJS](#)

For the record, this is the stack that Codecademy recommends for learners and it's what they have in their massive [Web Development Pro Path](#), which puts all these skills together with projects as you go.

This gives you HTML -> CSS -> JavaScript -> jQuery -> Command Line, Git, GitHub -> React -> ExpressJS -> SQL-> NodeJS.

After all this you should also learn MongoDB (free tutorial [here](#)) or Postgres (free tutorial [here](#)). MongoDB is a common part of many tech stacks, and that in combo with the above gives you one of the top "full-stack JS" stacks referred to as MERN, or Mongo, Express, React, and Node. MEAN (swapping React out for Angular) is another top stack, but as we said above Angular is less popular.

Python for Web Development

Python is doing almost everything these days, including web dev.

1. Learn Python [2](#) or [3](#) (click for courses).

The Codecademy Python 3 course is newer so it's still in Pro-only preview mode, and it's hard to finish in just a 7 day trial. You can learn Python 2 on Codecademy for free, many companies still use it as standard, and then just learn the differences (article on those differences [here](#)) between the two using developer documentation. Though the future is in Python 3, don't worry too much about learning Python 2 instead, as for beginners the differences are very minor (see [here](#)).

2. [Django](#) (free course).

Once you've learned Python, you'll need to learn a framework to apply it to web development, the most popular of which is arguably Django. There aren't a lot of great free Django tutorials, but again developer documentation should do the trick – the link above takes you to the official free resources on learning it. Flask is a good Django alternative with its own pros and cons.

Cloud Storage / Web Hosting

People are going to need to use your websites and to do that they need to be hosted somewhere. GitHub & GitHub Pages are a great start, but you could and should learn more. Learn about web hosting and domains with [this article](#). Then pick between the following tutorials:

- [AWS](#) (industry standard that you'll probably use on the job, has a basic free tier)
- [Heroku](#) (essentially free, probably best option for personal portfolio use)
- [Firebase](#) (by Google, free to start)

- [DigitalOcean](#) (great community, has some free credit)

From Skills to Job

Once you have skills and a portfolio, there are two major paths from here: freelance or full-time. I'll cover both, but here are some quick resources to read that will help you on either side.

- Your [online presence](#) can help you to get noticed, and [personal branding](#) is a big part of that.
- Know how to [describe yourself](#) on a résumé / CV / to others.
- Make a [good developer résumé](#).
- GitHub can [fast track](#) you to success, or it can de-legitimize your application. Read my guides on GitHub and portfolios for more info.

How will you know you're ready? *You won't*. Employers will, so let them decide. Apply for jobs, ask for feedback. If you have developer friends, ask them for advice too.

Going Freelance

Some people see freelance work as a way to make progress towards a full-time developer job, but really freelance is its own career path with its own pros and cons. How to get work as a freelance developer is a hard question to answer, but [here are some tips](#). Though your portfolio is always important, in the freelance world it's often hard to get going unless you've got a history of doing work for a client. A lot of people start off here offering their time for free for friends, family, non-profits, and local businesses, but there's a fundamentally different dynamic when you charge for your time so consider moving to that stage ASAP or charging from the get-go. Do some research before setting your hourly rate – [this](#) article will help.

Going Full-Time

It can be hard to break into any field without prior experience. No matter whether you did a CS degree, are self-taught, or went to a bootcamp, this will still be a problem. Things that help here are your body of work, your contacts, your network, inbound leads, and luck. Keep your GitHub going, list your coding skills on LinkedIn and AngelList, build out that portfolio, go to meetups and events, build your network, and get out there! If your work is good, recruiters are going to come to you. Those early conversations often won't go anywhere, but don't be discouraged.

Interviews

At some point, you'll get an interview. Coding interviews are scary, but they follow patterns that you can expect and prepare for. See [here](#), [here](#), and [here](#) for some quick reading to help.

Codecademy has a [course](#) on technical interview practice. At time of writing, it's Pro-only, but it's something you can complete in the free Trial window, and they have a [saved livestream](#) for free. It's also taught in Python, so though you can follow along somewhat if you learned Ruby / full-stack JS instead. In that, just think to yourself how you'd solve the problem in *your* language and then actually do it yourself separately. A lot of the theories and principles are the same. In job interviews if you're more comfortable in another language, don't be afraid to say "I don't know Python but in Ruby, I'd solve it like this..."

Some other good reading list resources:

- [How to Master a Take Home Coding Challenge](#)
- [How to Land a Programming Job Without a Degree](#)
- [Working at a Startup vs an Established Company](#)
- How to Negotiate a Salary [in general](#) and [as a Software Developer](#)
- [What You Should Consider Before Accepting a Job Offer](#)
- [Junior Developer Interview Tips to Success](#)

Not The End

Please reply to this thread with your own resources, advice, and feedback! I made this post a wiki so it can be updated and maintained by the community, I'm just starting them off. See my other guides [here](#).

[🔗 FAQ: Learn HTML - Intro to HTML - What is HTML?](#)

[🔗 Community Coding Guides](#)

jsnceo

Jan '19

Could add surge.sh, now.sh, and repl.it to hosting options!!

arc5977963948

Sep '19

You could add Bluemix at some moment, as hosting options too!

ohray




May 26

galinapodst:

memorizing a Russian dictionary doesn't mean you can speak Russian

It's funny to read it from Russia. Спасибо за материал!

CODECADEMY

- About
- For Business
- Shop
- Stories
- We're Hiring
-   

CATALOG

- BY SUBJECT
- Full Catalog
- Web Development
- Programming
- Data Science
- Partnerships
- Design
- Game Development

BY LANGUAGE

- HTML & CSS
- Python
- Javascript
- Java
- SQL
- Bash/Shell
- Ruby
- C++
- R
- C#
- PHP
- Go
- Swift

RESOURCES

- Beta Courses
- Articles
- Forums
- Help
- Blog
- Roadmap