

How to learn a language/framework from scratch

Make a copy of this, turn it into a pdf, or store it on your computer. We think these are the most important points to consider when learning a language or framework from scratch. Read, research, and explore more to choose what's best for you! Let us know how we did @ 30dayscoding@gmail.com. Good luck, you'll do great!

This document is mostly focused on frontend frameworks like React.

Introduction

- First few days of learning are an investment and will determine whether you stay with it or not. Most people leave right after the first few minutes or when they don't get their setup right!
- Read about what the framework is actually about - what problems it solves, how it is better than others, and how you can use it to the fullest. Read for a basic understanding.

Setup

- This is probably the most important step. It can sometimes be painful and annoying to set a path, install things, and finally get started with a basic app, but all of it is necessary.
- Things work out eventually. Watch a YouTube video if you're stuck, Google search the errors, and try to solve them. Don't stop before you get it right.
- You can also start with platforms like **Stackblitz** which we've discussed below.

Stackblitz

- DON'T set this up unless you're actually familiar with the framework. How can you become familiar with it? By simply going to [StackBlitz: The online code editor for web apps. Powered by Visual Studio Code.](#)

- Go to Stack Blitz, create a simple app, see how files are structured, and play around with it. Create some things and see the new changes. Add routing, make new pages, and create super small projectl.
- Google 'table example stack blitz' - you'll get hundreds of links of other people's Stack Blitz who have implemented those changes. This is super simple + effective. You can make even bigger things here. Although, creating something locally and then uploading to Github + Netlify has a better feel overall.

Learn the basics

- Start with the basics. Learn the building steps - how to add a button, how to change the text, etc. Try bigger things later.
- Read the **documentation**: there are multiple ways to implement a single thing and it's important to be aware of this. Reading the documentation is generally an important thing and will help you when building future projects.
- For example: Angular
 - HTML, CSS, Components
 - Input, output, different pages
 - Routing, transitions, passing data
 - Simple TODO app, couple pages

Tutorial hell

- Don't fall in the trap of just taking courses, watching tutorials, and copying code. Instead, be curious about problems and try to solve them yourself.
- Think about a **new feature**, Google how to do it, and then TRY it, instead of looking for courses/tutorials.
- Think of different **implementations** for the same thing. What can you do with firebase? What all can you do with a button and some URL redirection?
- One good approach is to find similar projects related to your problem. So, for instance, you want to make a page that has a card component and you want to replace it with something on a button click. Search how to make a quiz app. You'll find a similar approach that you can follow.

- Don't make a social media app. If you are following a tutorial which tells you to make one, close it, think of an idea similar to it, and try to make it by using what you learned in the course. Don't just blindly follow the same things that others are doing.
- https://www.youtube.com/watch?v=_3sPBX9TpyA&ab_channel=ChrisSean

Make a simple project

- Think of simple ideas, problems, and how you can solve them. Maybe you always wanted to make your portfolio website or something. Make it. Whatever you want to build: Make it. Start small and then go big.
- For example:
 - A website that allows you to find lost items
 - A landing page that displays products
 - Clicking inside a product to see more details
 - Extra: Add more features once you've completed the first ones
 - Add another page, smaller features
 - Add multiple users, authentication

Games

- Games are a great way to learn how complex things work. They're also a great way to understand OOPS. Most gaming classes will have `gameOver()`, `start()`, `find()`, and similar methods which help you write better code and understand how to avoid duplication.
- Most games also have a tricky `gameOver()` logic. For example Tic tac toe - you need 3 of the same things in the same line (in any direction). So, it's a good challenge to think about the logic before implementing the method.
- It's also exciting to see how frontend and logic work together when you're making games.

Bigger project

- Go to hackathons, find people, make connections, and work with someone. Working alone on a big project is the easiest path to not completing it. Therefore, you should find like-minded people, share your ideas with them, and finish a project together..
- Keep in mind your audience. It is good to make something that people might actually end up using! It could be anything from a lost-and-found website to something in education, travel, Chrome extensions, work life balance etc. There are tons of problems these days, so choose something and start building. It could even be as simple as <https://30dayscoding.com> if you like that stuff.

Don't make something for your resume - it won't last long.