**1**

**Why you should learn C++**

**It can be challenging to choose which programming language you should learn in 2021. There are hundreds of great programming languages out there. Out of those, choosing which one you should learn is a challenging task. C++ is known to be the most powerful programming language right now. And also, it’s the most popular programming language in the world as well. It’s used for everything from building operating systems to crafting video games and creating 3D movies. While it may have a steeper learning curve than others, C++ has incredible potential. There are many reasons to learn C++. Although C++ is difficult to learn, it’s an overall great programming language for any future tech enthusiast. Many multinational companies like Microsoft, LinkedIn, Opera, NASA, Facebook, etc.. use C++ for their development activities.**

**C programming language is the mother language of C++. C was created by Dennis Ritchie in 1972. Later on, in 1985 Bjarne Stroustrup developed C++. C was initially developed for UNIX operating systems and to make programming easy. Today, C++ is used to write programs compatible with Windows, Linux and macOS operating systems. C++ is also an object-oriented programming language like Java, which is a type of development using objects and classes.**

**C++ is known for being a more difficult language to learn than others such as Java, Python and C as well. This can be challenging for those starting with programming. With that said, investing the time to learn C++ can bring a number of benefits to you and your career. The main benefit you can gain from learning C++ is syntax. The C++ syntax is similar to many other programming languages like, So, Rust and Scala. This means once you mastered C++, learning the other complicated programming languages are really easy for you to pick up. C++ is also used in hardware development as well. Knowing how to code in C++ gives you a better sense of how both software and hardware work together.**

**Let’s talk about the time it takes to learn C++. It obviously depends on your skillset. If you are fairly new to programming, I recommend you start with a language like C or Python and work your way up to C++. If you have some experience in programming and had the chance to gain hands-on experience in languages like Java and Python, you might be able to master the syntax of C++ within 2 to 3 months. But, to really learn a language, expect to learn at least a year of studying and building projects. The best way you can learn a programming language is by building small projects. But, you should learn the basic syntaxes of any programming language first. Although there are C++ developers in large multinational companies, they haven’t even done studying yet since, technology updates day by day. So, If you want to become a good C++ developer you must have the discipline to keep up with new updates and new coding tricks.**

**2)**

# ****Programming Languages to Avoid and Learn in 2022.****

## **Things I didn’t know when I started coding.**

**(Scroll down for a cut-to-bs version)**

We speak so many languages but, how many do we actually CODE in?

It is hard to choose just **one** programming language which is

* in demand in job market and
* will continue to be in —
* that too with a high pay scale,
* developer satisfaction and
* productive! Yes.

and it’s not always that easy to pick one, never will be, :(

there will always be some trade-offs (or is this even a right question to ask?)

“because no programming language is perfect!”

one is easy to learn but sometimes slow(Python),  
one is easy to learn but has a tiny userbase and hardly any libraries (Nim)  
one is very fast but a bit hard to learn(C/C++),  
one is incredibly fast, but incredibly hard to learn (NASM)  
one is verbose but in demand(Java/C#),  
one is made by Google(Go/Dart),  
One is everywhere but no one talks about it because it’s favoured by script kiddies (PHP)  
one is used everywhere but saturated job market(JavaScript) and so on…

“Languages need hype to survive; I just wish people didn’t have to be blinded by it.”

you get the feel, right?

## A SHORT OVERVIEW:

This article is my personal opinion. Every language is as good as a tool, it’s just how you, the developer, use it.

You probably won’t finish this article. In fact, I may have already lost you

* to another tab in your browser.
* Or an email from your boss.
* Or a ping from a coworker.
* Or any number of other digital distractions that have come to define your modern life.

[The below recommendations are for the context of the United States, United kingdom, and other 1st world and some 2nd world (Russia, and neighbours, etc) countries. Scroll down for Indian and Chinese context.]

## YOU SHOULD LEARN:

1. **Javascript** — Most used, versatile, easy, large developer community and WORA (Write Once Run Anywhere), scripting, frontend, backend, ML, games, mobile and webApps (be aware of its dependency hell).
2. **Python** — easy to learn, backend, ML, data science, data visualization, scripting, hacking, automation, education and finance sector, web scraping, adoption continues to grow.
3. **Dart** — One codebase for Android, iOS, Web App, Windows, macOS(beta), Linux(beta) application, embedded systems (unstable), servers → Flutter.
4. **C#** — Platform agnostic, simple, general-purpose language, desktop, console and web application, windows and web services, game development, VR, AR.
5. **Go** — Solves scalability issues, simple elegant code, easy to learn, the language of the cloud, DevOps, backend, servers, microservices, DNS, cloud native development. (Uber, Netflix, Docker, Kubernetes, Soundcloud, Prometheus)

## YOU SHOULD CONSIDER LEARNING:

1. **Java** — In high demand worldwide, big data, android, finance, databases, system software, testing tools, enterprise king, the core language of industry-standard tools like Apache Kafka, spark, hive, MTLAB is Java based, distributed apps, IoT, Hadoop, etc.
2. **R** — statistics, ML, Data Visualisation, fintech, research, retail, automobile industry, but slow and data-intensive.
3. **Kotlin** — server-side, client-side web and Android, (embedded systems, macOS and iOS is coming), gradle plugins, microservices, smart-contracts, backend, data-science.

**Swift** — Develop for anything that has  on it (iPhone, iPad, macOS, watchOS, tvOS and apps for all of them) versatile, simplification of existing native app development methods, productive than obj-C

1. **Rust** — memory management, systems programming, Linux kernel, game engines, operating systems, file systems, browser components and simulation engines for virtual reality.Takes a little time to learn(relative to Go). Good pay.
2. Others → C and Julia.

## YOU SHOULD NOT CONSIDER:

1. **Ruby** — webApps, automation (WATIR), scraping, servers, DNS, but better tools are available now. (Sidenote: Github, Gitlab are written in Ruby on Rails framework, and still the first choice of many startups for easy MVP and fast deployment).
2. **PHP** — PHP originally stood for Personal Home Page, but it now stands for the recursive initialism, **H**ypertext **P**reprocessor.  
   **→**Desktop GUI, web backend, lacks good debugging tools (compared to other languages), relatively low pay, still extensively popular, tough competition from java, E-commerce and public websites King. [Earlier versions of PHP paid less attention to security features, presently, it’s as robust as any other language]
3. **C++** — bridge between LLP and HLP, memory unsafe, game engine development, OS level, embedded, fintech, trading, compilers, browsers, VMs, can be frustrating to work with. (Visa, Mastercard, Amex all use C++ for their backend system). Do not consider C++ if gaming and hardware programming does not interest you.
4. Super-new and very niche-specific languages which do not have a healthy community.[For more on that, please refer to the end of the article]

[Edit: Java is promoted to the ‘You should consider’ section after consideration with industry devs. ]

## FOR WEB3:

## Languages:

Solidity, Javascript, Rust, C++, Substrate.

## Frameworks:

Truffle Framework, Hardhat Framework, Brownie Framework, OpenZeppelin SDK, Chainlink SDK.

## FOR INDIAN CONTEXT:

1. **Java** — Most in-demand language according to Indian Job Market, Safe bet.
2. **Javascript** or **Python** — Master anyone and you are good to go. Period.
3. **C++** — Use it for Data Structure and Algorithms and Competitive Programming (at Codeforces, Codechef, Topcoder, SPOJ, etc and participate in Hashcode, Kickstarter, Codejam, Hacker Cup, ACM ICPC, etc)

Recommendation: **Java** and **Javascript**

## FOR CHINESE CONTEXT:

1. **Java** [29.28%]
2. **C++** [16.08%]
3. **Javascript** [15.09%]
4. **C#** [10.95%]
5. **Python** [8.21%]
6. **Go** [6.94%]
7. **PHP** [5.19%]
8. **Matlab** [1.48%]
9. **Lua** [1.28%]
10. **Swift** [0.83%]

[Source](https://github.com/EricWebsmith/china_job_survey/) here, [xx%] indicates the percentage of developers using that language per 100 dev.

All the programming languages are built for some specific purpose, over time, they deviate towards general-purpose languages (can be used to code almost everything, e.g Javascript, Java, and to some extent Python).

You have a very slim chances (if you are a fresher) to directly land your first Job if you just know :

1. **Rust**, but not **C++**  
2. **Go**, but not **Java**  
3. **Kotlin**, but not **Java**  
4. **Swift**, but not **Objective-C**  
5. **Julia**, but not **Python**  
6. **Dart**, but zero experience with any native app development language.

You all won’t agree with me on this but thats okay!

Just have a closer look at the Job description of companies who hire folks who know cutting edge technologies, they will have “must have 1–2 years of experience working with [the-language-which-you-hate] language”

**People claim PHP runs the world**,  
→ Sure, according to % of websites PHP is ahead but hey 1/3 of websites on this planet are WORDPRESS sites and Wordpress is built on PHP, people who do not know to code (or do not want to code) use Wordpress templates and use them, it’s just like duplicating same codebase with minor textual changes and hence PHP wins here, nothing else. Why people moved to Javascript if PHP was that great? Tell me.

**3)**

# I spent 3 months applying to jobs after a coding bootcamp. Here’s what I learned.

**A less-talked about part of the bootcamper’s journey is what happens after you graduate — when you’re searching for that six-figure developer position.**

**I completed Hack Reactor in July 2016 and took almost 3 months before accepting an offer with Radius Intelligence. I applied to 291 companies, did 32 phone screens, 16 technical screens, 13 coding challenges, 11 on-sites, and received 8 offers. The offers ranged from $60-125k in salary from companies all over the US, and for both front end and full stack roles. In total, 2.8% of applications became offers.**

**Here are 5 things I wish I’d known before I began my job search.**

# ****Insight #1: Get through to real people****

**At first, I applied for companies using the shotgun approach. I applied through Indeed.com, AngelList, LinkedIn, StackOverflow, Hacker News, company websites, and even Craigslist.**

**I’d submit a resume for any role that wanted React, Node, or JavaScript experience. In the first week, I applied to 15–20 companies a day.**

**Pro-Tip: Find companies using this**[**easy-application repo**](https://github.com/j-delaney/easy-application)**.**

**My yield was low. Less than five percent of companies responded to me. I was throwing applications into a black hole.**

**Everything changed when one of my cohort-mates, a former recruiter, shared a guide to the job search. He told us to send emails directly to real people with each application. It could be anybody. As long as someone read it.**

**From then on, whenever I submitted an application, I searched for the company on LinkedIn and emailed someone on their engineering or hiring team.**

**For most small companies or C-level executives, the email format is usually firstName@dreamCompany.com. For larger companies, it may be firstName.lastName@dreamCompany.com.**

**To verify emails, I used [Rapportive](https://rapportive.com/" \t "_blank) to cross-check emails with social media accounts.**

**The results were amazing. With 150+ emails sent, my response rate was a whopping 22%.**

**It also felt great to hear from real people. Surprisingly, CEOs and CTOs responded to me. Sometimes they even interviewed me themselves.**

**Takeaway: If you’re applying through the front door, make sure you’re getting to human beings.**

# Insight #2: Start small and work your way up

**You will face Level 1 interviews (a non-tech company that needs any dev), where interviewers ask you nothing more than JavaScript trivia.**

**You will face Level 9 interviews (Google/Facebook level), where interviewers ask difficult data structure and algorithm questions.**

**I strategically set up my process so that I had lower-level interviews earlier, and higher-level interviews later on.**

**Early on, I gained experience, built confidence, and secured offers from companies that had less intensive interviews.**

**As I got more experience, I effectively “leveled up.” I became capable of completing interviews at companies with higher hiring bars. This is illustrated below as a linear correlation between the number of weeks I was into the process and the base salary I was offered.**

# ****Insight #3: Study like your future job depends on it (because it does)****

**I hate to break it to you, but the most important thing you could be doing at any point is studying and preparing.**

**Why? Because you won’t get the offer if you don’t have good answers to the questions they ask you.**

**People won’t refer you if they don’t think you’re prepared for their interviews.**

**Coming out of Hack Reactor, my weaknesses were data structures and algorithms.**[**A study by Triplebyte**](http://blog.triplebyte.com/bootcamps-vs-college)**has found that bootcamp grads are weaker in these areas than computer science grads.**

**So I learned and practiced. Every day.**

**I devoted entire days to learning sorting algorithms. Other days, I focused on understanding how the internet worked.**

**If I didn’t fully understand a concept, I’d spend the day watching YouTube videos or searching StackOverflow until I did.**

**I found the following study materials useful:**

* [**InterviewCake**](https://www.interviewcake.com/)**: My favorite resource for data structures and algorithms. It breaks down solutions into step-by-step chunks — a great alternative to Cracking the Code Interview (CTCI). My only gripe is that they don’t have more problems!**
* [**HiredInTech’s System Design Section**](https://www.hiredintech.com/classrooms/system-design/lesson/60)**: A great guide for system design interview questions.**
* [**Coderust**](https://www.educative.io/collection/5642554087309312/5679846214598656)**: If you’re avoiding CTCI like the plague, Coderust 2.0 may be perfect for you. For $49, you get solutions in almost any programming language, with interactive diagrams.**
* [**Reddit’s How to Prepare for Tech Interviews**](https://www.reddit.com/r/cscareerquestions/comments/1jov24/heres_how_to_prepare_for_tech_interviews/)**: I constantly used this as a benchmark for how prepared I was.**
* [**Front End Interview Questions**](https://github.com/h5bp/Front-end-Developer-Interview-Questions)**: An exhaustive list of front-end questions.**
* [**Leetcode**](https://leetcode.com/)**: The go-to resource for algorithm and data structure questions. You can filter by company, so for example, you could get all the questions that Uber or Google typically ask.**

**Takeaway: There’s no such thing as too much preparation.**

# ****Insight #4: Put your best foot forward****

**Breaking into the industry is hard. You have to perform well, even when you’re not fully prepared. In order to succeed, you have to be your own advocate.**

## **Sell Yourself**

**At Hack Reactor, we’re trained to mask our inexperience. In our personal narratives, we purposely omit our bootcamp education.**

**Why? Otherwise, companies automatically categorize us into junior developer roles or tag us as “not enough experience.”**

**In one interview with a startup, the interview immediately went south once they realized I’d done a bootcamp. One company used it against me and made me a $60k offer, benchmarking against junior developers.**

**Ultimately, you need to convince companies that you can do the job.**

**At the same time, you need to convince yourself that you can do the job.**

**You can. Focus on your love for programming. Focus on what you’ve built with React and Node. Focus on demonstrating your deep knowledge in JavaScript and any other languages you’ve learned.**

**Only then can they justify giving you the job.**

## **It’s a Two-way Conversation**

**Interviewing is a mutual exploration of fit between an employee and an employer. While it’s your job to convince employers to hire you, it’s also their job to win you over.**

**Don’t be ashamed of using the interview as an opportunity to evaluate the job opportunity.**

**I talked to any company, even if I had only the slightest interest.**

**I did on-sites all over the country with any company that invited me out. I asked questions, and sucked up knowledge on engineering team organization, technologies and tools used, company challenges, and system architecture.**

**Pro-Tip: During interviews, ask the following questions:**

**What are some technical challenges you’ve recently faced?**

**What do you enjoy about working at X company?**

**How are teams structured and how are tasks usually divided?**

**I treated every interaction as a learning opportunity. Each interaction helped me improve my presentation, interview, and technical skills. Each failure helped me find my blind spots.**

**4)**

# Tools I wish I had known about when I started coding

# Chrome Extensions

**Now that I am a self-proclaimed web developer, I practically live in my Chrome console. Below are some tools that allow me to spend less time there:**

* [**WhatFont**](https://chrome.google.com/webstore/detail/whatfont/jabopobgcpjmedljpbcaablpmlmfcogm?hl=en)**— The name says it all. This is an easy way of finding out the fonts that your favorite website is using, so that you can borrow them for your own projects.**
* [**Pesticide**](https://chrome.google.com/webstore/detail/pesticide-for-chrome/bblbgcheenepgnnajgfpiicnbbdmmooh?hl=en)**— Useful for seeing the outlines of your <div>s and modifying CSS. This was a lifesaver when I was trying to learn my way around the box-model.**
* [**Colorzilla**](https://chrome.google.com/webstore/detail/colorzilla/bhlhnicpbhignbdhedgjhgdocnmhomnp?hl=en)**— Useful for copying exact colors off of a website. This copies a color straight to your clipboard so you don’t spend forever trying to get the right RGBA combination.**
* [**CSS Peeper**](https://chrome.google.com/webstore/detail/css-peeper/mbnbehikldjhnfehhnaidhjhoofhpehk?hl=en)**— Useful for looking at colors and assets used on a website. A good exercise, especially when starting out, is cloning out websites that you think look cool. This gives you a peek under the hood at their color scheme and allows you to see what other assets exist on their page.**
* [**Wappalyzer**](https://chrome.google.com/webstore/detail/wappalyzer/gppongmhjkpfnbhagpmjfkannfbllamg?hl=en)**— Useful for seeing the technologies being used on a website. Ever wonder what kind of framework a website is using or what service it is hosted on? Look no further.**
* [**React Dev Tools**](https://chrome.google.com/webstore/detail/react-developer-tools/fmkadmapgofadopljbjfkapdkoienihi?hl=en)**— Useful for debugging your React applications. It bears mentioning that this is only useful if you are programming a React application.**
* [**Redux Dev Tools**](https://chrome.google.com/webstore/detail/redux-devtools/lmhkpmbekcpmknklioeibfkpmmfibljd?hl=en)**— Useful for debugging applications using Redux. It bears mentioning that this is only useful if you are implementing Redux in your application.**
* [**JSON Formatter**](https://chrome.google.com/webstore/detail/json-formatter/bcjindcccaagfpapjjmafapmmgkkhgoa?hl=en)**— Useful for making JSON look cleaner in the browser. Have you ever stared an ugly JSON blob in the face, trying to figure out how deeply nested the information you want is? Well this makes it so that it only takes 2 hours instead of 3.**
* [**Vimeo Repeat and Speed**](https://chrome.google.com/webstore/detail/vimeo-repeat-speed/noonakfaafcdaagngpjehilgegefdima?hl=en)**— Useful for speeding up Vimeo videos. If you watch video tutorials like most web developers, you know how handy it is to consume them at 1.25 times the regular playback speed. There are also versions for YouTube.**

# VS Code Extensions

**Visual Studio Code is my editor of choice.**

**People love their text editors, and I am no exception. However, I’m willing to bet most of these extensions work for whatever editor you are using as well. Check out my favorite extensions:**

* [**Auto Rename Tag**](https://marketplace.visualstudio.com/items?itemName=formulahendry.auto-rename-tag)**— Auto rename paired HTML tags. You created a <p> tag. Now you want to change it, as well as its enclosing </p> tag to something else. Simply change one and the other will follow. Theoretically improves your productivity by a factor of 2.**
* [**HTML CSS Support**](https://marketplace.visualstudio.com/items?itemName=ecmel.vscode-html-css)**— CSS support for HTML documents. This is useful for getting some neat syntax highlighting and code suggestions so that CSS only makes you want to quit coding a couple of times a day.**
* [**HTML Snippets**](https://marketplace.visualstudio.com/items?itemName=abusaidm.html-snippets)**— Useful code snippets. Another nice time saver. Pair this with**[**Emmet**](https://emmet.io/)**and you barely ever have to type real HTML again.**
* [**Babel ES6/ES7**](https://marketplace.visualstudio.com/items?itemName=dzannotti.vscode-babel-coloring)**— Adds JavaScript Babel syntax coloring. If you are using Babel, this will make it much easier to differentiate what is going on in your code. This is neat if you like to play with modern features of JavaScript.**
* [**Bracket Pair Colorizer**](https://marketplace.visualstudio.com/items?itemName=CoenraadS.bracket-pair-colorizer)**— Adds colors to brackets for easier block visualization. This is handy for those all-too-common bugs where you didn’t close your brackets or parentheses accurately.**
* [**ESLint**](https://marketplace.visualstudio.com/items?itemName=dbaeumer.vscode-eslint)**— Integrates ESLint into Visual Studio Code. This is handy for getting hints about bugs as you are writing your code and, depending on your configuration, it can help enforce good coding style.**
* [**Guides**](https://marketplace.visualstudio.com/items?itemName=spywhere.guides)**— Adds extra guide lines to code. This is another visual cue to make sure that you are closing your brackets correctly. If you can’t tell, I’m a very visual person.**
* [**JavaScript Console Utils**](https://marketplace.visualstudio.com/items?itemName=whtouche.vscode-js-console-utils)**— Makes for easier console logging. If you are like most developers, you will find yourself logging to the console in your debugging flow (I know that we are supposed to use the debugger). This utility makes it easy to create useful console.log() statements.**
* [**Code Spell Checker**](https://marketplace.visualstudio.com/items?itemName=streetsidesoftware.code-spell-checker)**— Spelling checker that accounts for camelCase. Another common source of bugs is fat-thumbing a variable or function name. This spell checker will look for uncommon words and is good about accounting for the way we write things in JavaScript.**
* [**Git Lens**](https://marketplace.visualstudio.com/items?itemName=eamodio.gitlens)**— Makes it easier to see when, and by whom, changes were made. This is nice for blaming the appropriate person when code gets broken, since it is absolutely never your fault.**
* [**Path Intellisense**](https://marketplace.visualstudio.com/items?itemName=christian-kohler.path-intellisense)**— File path autocompletion. This is super handy for importing things from other files. It makes navigating your file tree a breeze.**
* [**Prettier**](https://marketplace.visualstudio.com/items?itemName=esbenp.prettier-vscode)**— Automatic code formatter. Forget about the days where you had to manually indent your code and make things human-legible. Prettier will do this for you much faster, and better, than you ever could on your own. I can’t recommend this one enough.**
* [**VSCode-Icons**](https://marketplace.visualstudio.com/items?itemName=robertohuertasm.vscode-icons)**— Adds icons to the file tree. If looking at your file structure hurts your eyes, this might help. There is a helpful icon for just about any kind of file you are making which will make it easier to distinguish what you are looking at.**

# ****In Conclusion****

**You likely have your own set of tools that are indispensable to your development cycle. Hopefully some of the tools I mentioned above can make your workflow more efficient.**

**Do not fall into the trap, however, of installing every tool you run across before learning to use the ones you already have, as this can be a huge time-sink.**

**I encourage you to leave your favorite tools in the comments below here, so that we can all learn together.**

**If you liked this article please give it some claps and check out other articles I’ve written**[**here**](https://codeburst.io/how-i-cut-my-react-javascript-bundle-size-in-half-with-three-lines-of-code-fe7798ecbd3f)**,**[**here**](https://medium.com/@mariohoyos/a-personal-case-for-gratitude-bff8345b3514)**,**[**here**](https://codeburst.io/my-3-favorite-things-about-the-coding-bootcamp-experience-cd4e1a3fd3dc)**, and**[**here**](https://medium.com/@mariohoyos/life-is-too-short-to-be-miserable-and-why-i-left-a-six-figure-job-in-the-search-for-something-327ab84416a9)**. Also, give me a follow on**[**Twitter**](https://twitter.com/marioahoyos)**.**

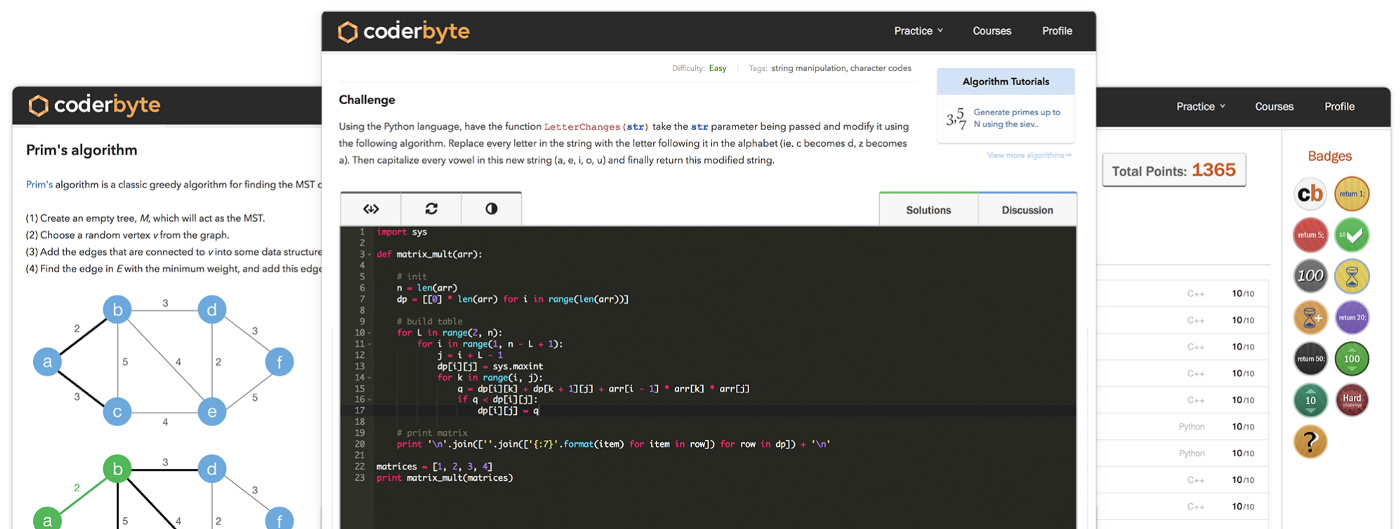
# 5)

# The 10 Best Coding Challenge Websites for 2018

At the end of 2016 I published the article: [The 10 most popular coding challenge websites for 2017](https://medium.freecodecamp.org/the-10-most-popular-coding-challenge-websites-of-2016-fb8a5672d22f). The list was based on a few resources such as popular blog posts, Quora posts, articles, Google searches, and popular posts on forums like [r/learnprogramming](https://www.reddit.com/r/learnprogramming/) and [Hacker News](https://news.ycombinator.com/item?id=13566247).

This updated 2018 list features 10 websites that offer the **best coding challenges**and resources to help new and intermediate developers **improve their skills**, **prepare for interviews**, and **progress in their careers**. The ordering of the list is based on level of difficulty (beginner to advanced).

# [1. Coderbyte](https://www.coderbyte.com/)

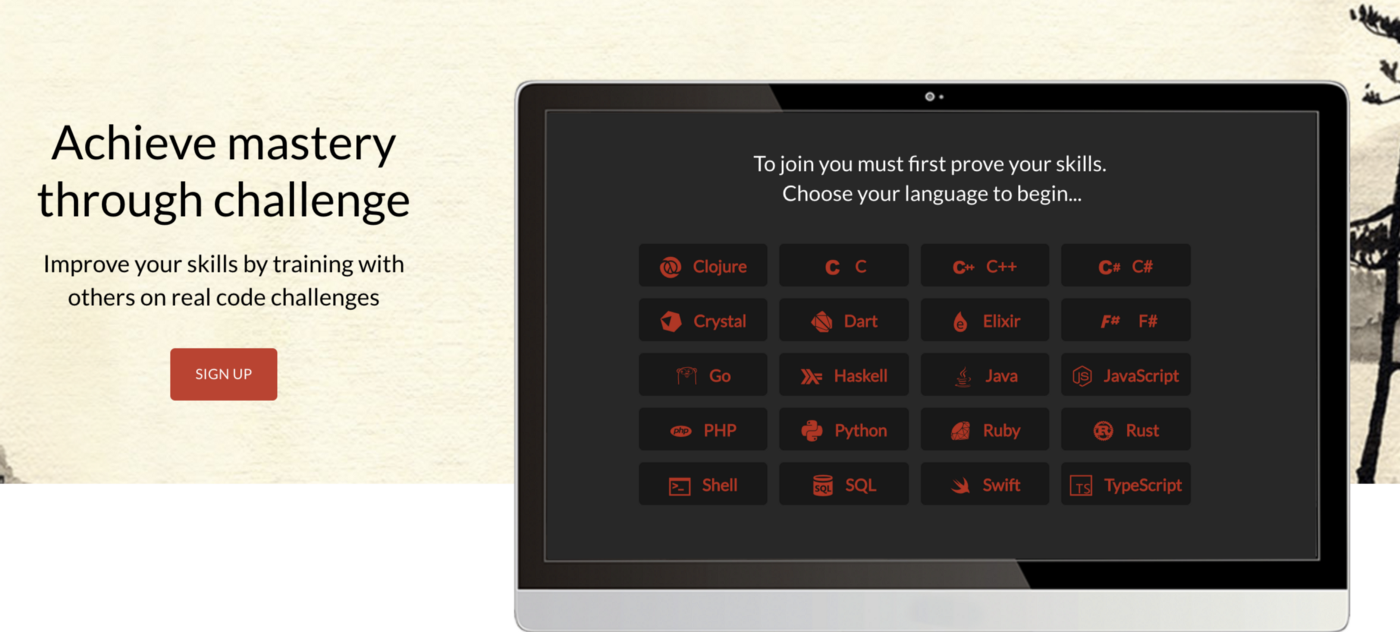


**Beginner — Intermediate**

Coderbyte provides 200+ coding challenges you can solve in an online editor using 10 different programming languages. It also provides official solutions for some of the challenges along with 800,000+ user solutions. Coderbyte is recommended by the [top coding bootcamps](https://www.coderbyte.com/organizations) because of its collection of interview prep challenges.

Aside from coding challenges, they provide courses in [Algorithms & Data Structures](https://www.coderbyte.com/course/learn-data-structures-and-algorithms/), [Web Development](https://www.coderbyte.com/course/fullstack-web-development/), and prep courses for [coding bootcamps](https://www.coderbyte.com/course/prepare-hack-reactor).

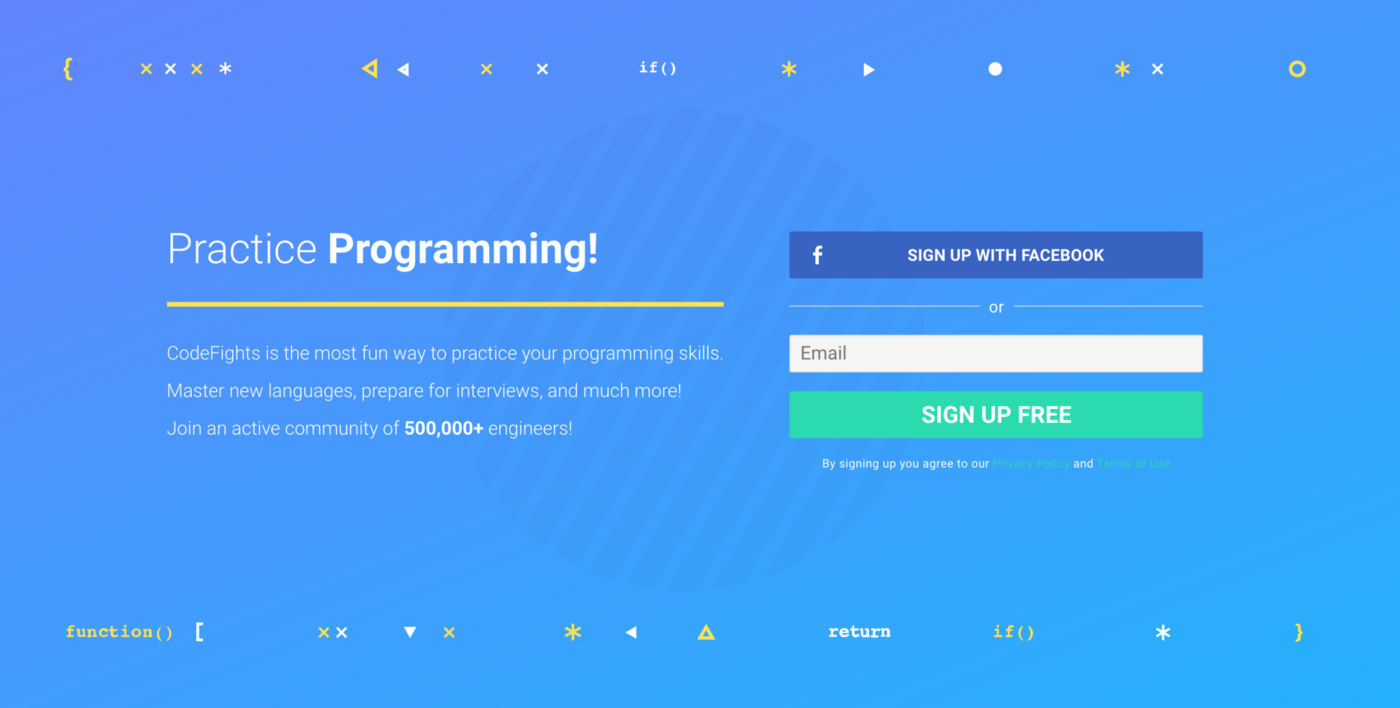
# [2. Codewars](https://www.codewars.com/)



**Beginner — Intermediate**

Codewars provides a large collection of coding challenges submitted and edited by their own community. You can solve the challenges directly [online](https://www.codewars.com/kata/trailing-zeros-in-factorials-in-any-given-integer-base/train/javascript) in their editor using one of 20+ programming languages. You can view a discussion for each challenges as well as user solutions. You can earn points and climb the rankings by solving their challenges.

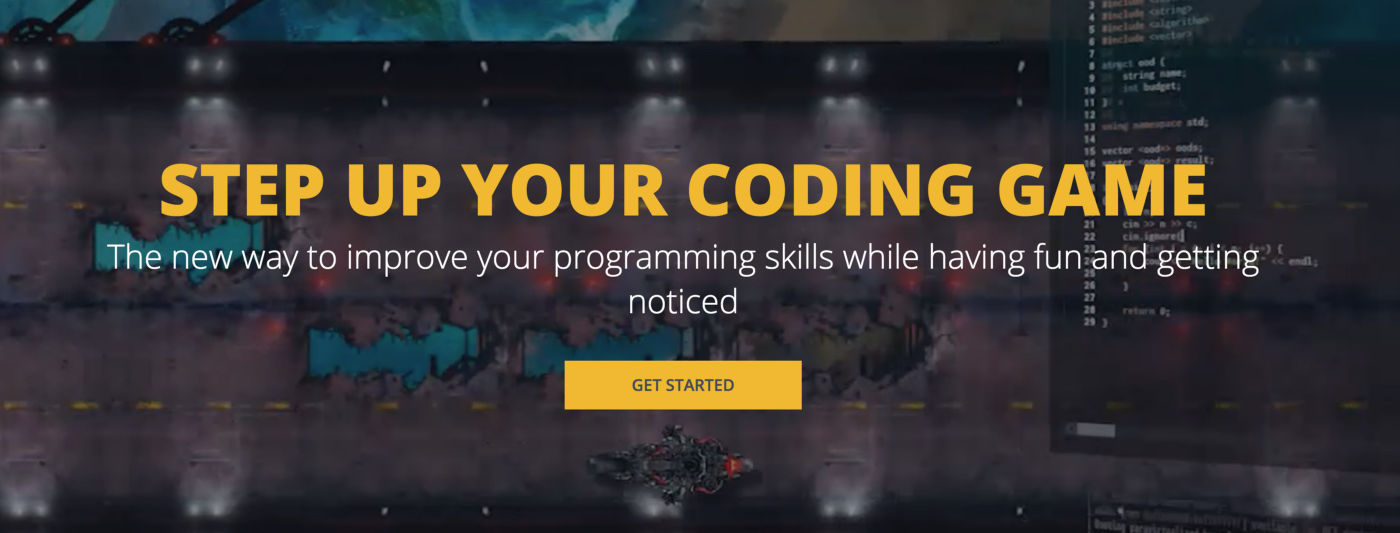
# [3. CodeFights](https://codefights.com/)



**Beginner — Intermediate**

CodeFights provides a collection of coding challenges for you to solve in their online editor and there is a [forum](https://codefights.com/forum) for users to discuss the challenges. They also have a feature called [Company Bots](https://codefights.com/company-bots) where you write code to compete against a “bot” that was programed by engineers at companies like Uber, Dropbox, and Quora.

# [4. CodinGame](https://www.codingame.com/start)



**Beginner — Intermediate**

This website is different than the ones listed above because instead of simply solving coding challenges by producing a set of outputs, on CodinGame you write code to actually play a game online. It’s a great website and some of the games are very fun to watch once you’ve written your code (see [example](https://www.codingame.com/ide/980141293426648e9f31adb0881e1e01ae52d38)). The game graphics are nice and the editor/game UI is very smooth. Overall it’s a great website for beginners to learn coding.

# [5. TopCoder](https://www.topcoder.com/challenges/)

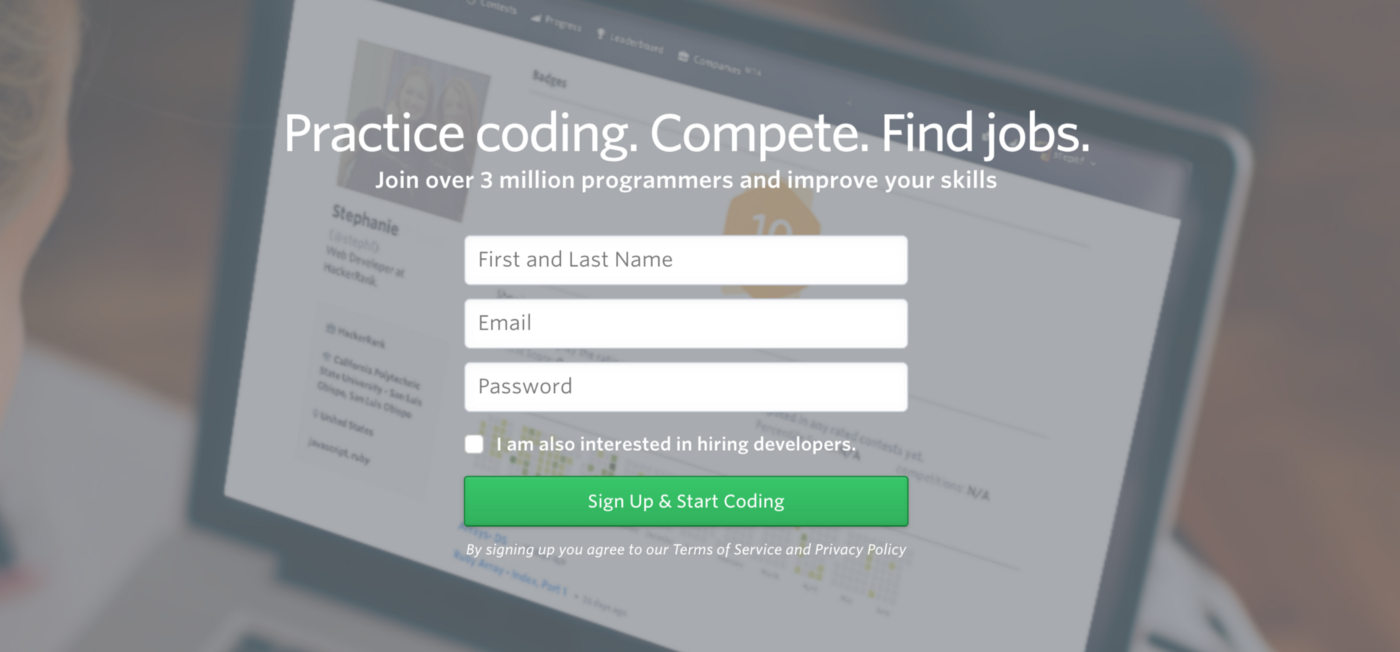


**Intermediate — Advanced**

TopCoder is one of the original platforms for competitive programming online. It provides a list of [algorithmic challenges](https://www.topcoder.com/community/how-it-works/) from the past that you can complete on your own directly online using their code editor. Their popular Single Round Matches are offered a few times per month at a specific time where you compete against others to solve challenges. Here are some [topics](https://www.topcoder.com/community/data-science/data-science-tutorials/) that their challenges may cover.

Aside from solving challenges for fun online, they offer sponsored competitions where you can [win prizes](https://www.topcoder.com/challenges/) for writing the best solution.

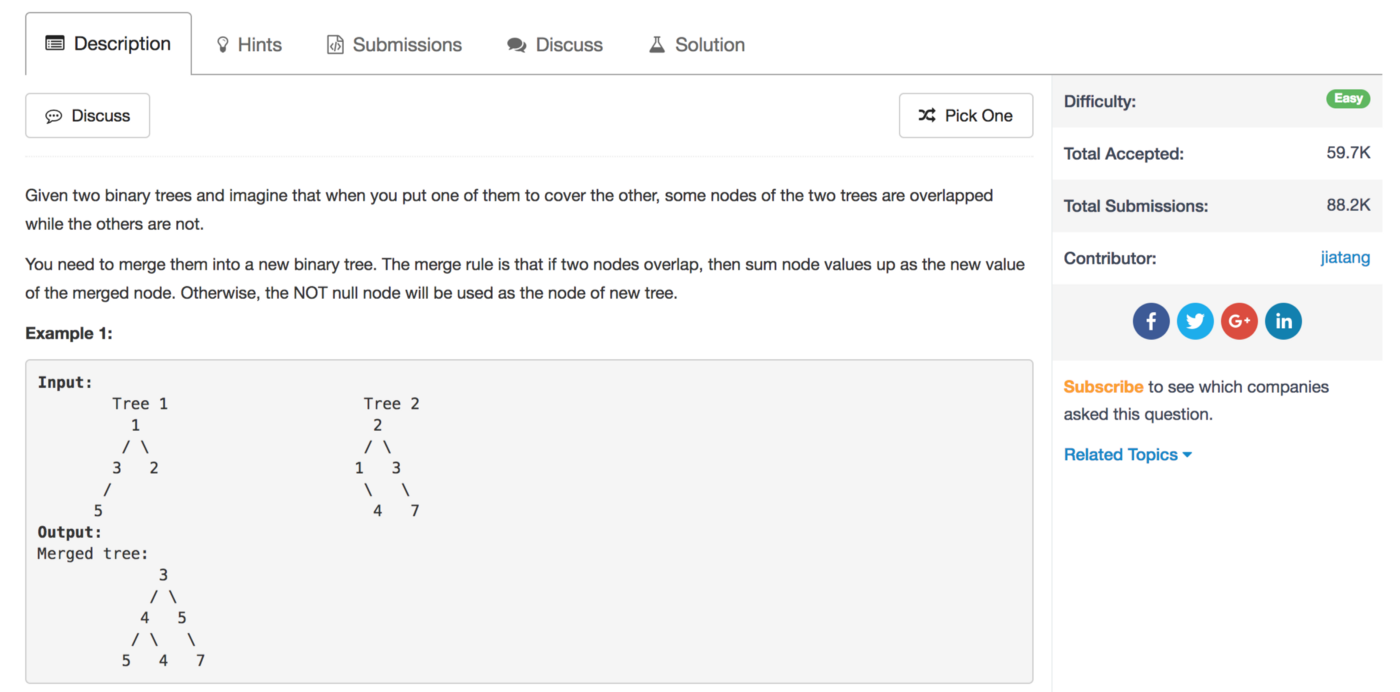
# [6. HackerRank](https://www.hackerrank.com/)



**Intermediate — Advanced**

HackerRank focuses on computer science topics so it provides challenges for several domains such as Algorithms, Mathematics, SQL, Functional Programming, AI, and more. You can solve all the challenge directly online (see [example](https://www.hackerrank.com/challenges/simple-array-sum)). They provide a discussion and leaderboard for every challenge, and most challenges come with an editorial that explains more about the challenge and how to approach it to come up with a solution.

# [7. LeetCode](https://leetcode.com/)

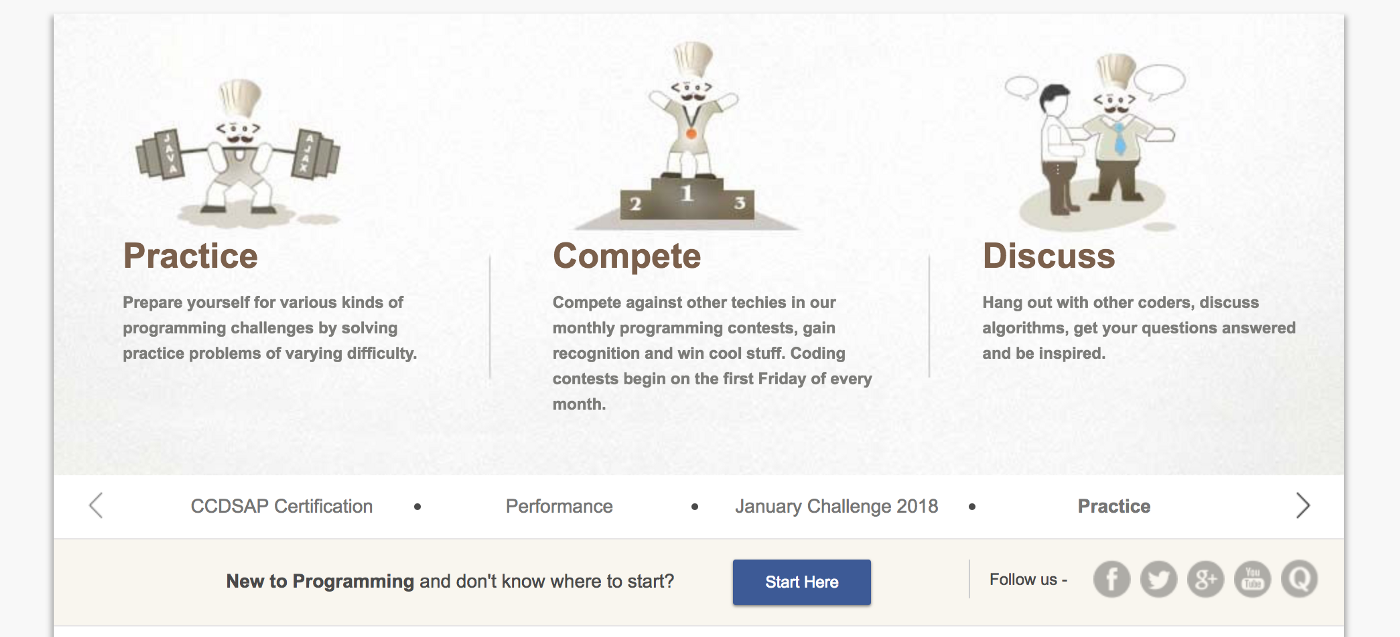


**Intermediate — Advanced**

LeetCode has a collection of some of the best algorithm challenges online today. The topics they cover require knowledge of data structures such as binary trees, heaps, linked lists, etc., which is why their challenges are a bit more advanced than some other websites — but the challenges are great if used when preparing for a software engineering interview.

They also have a [Mock Interview](https://leetcode.com/mockinterview/) section that is specifically for job interview preparation, they host their own coding [contests](https://leetcode.com/contest/), and they have a section for [articles](https://leetcode.com/articles/pour-water/) to help you better understand certain problems.

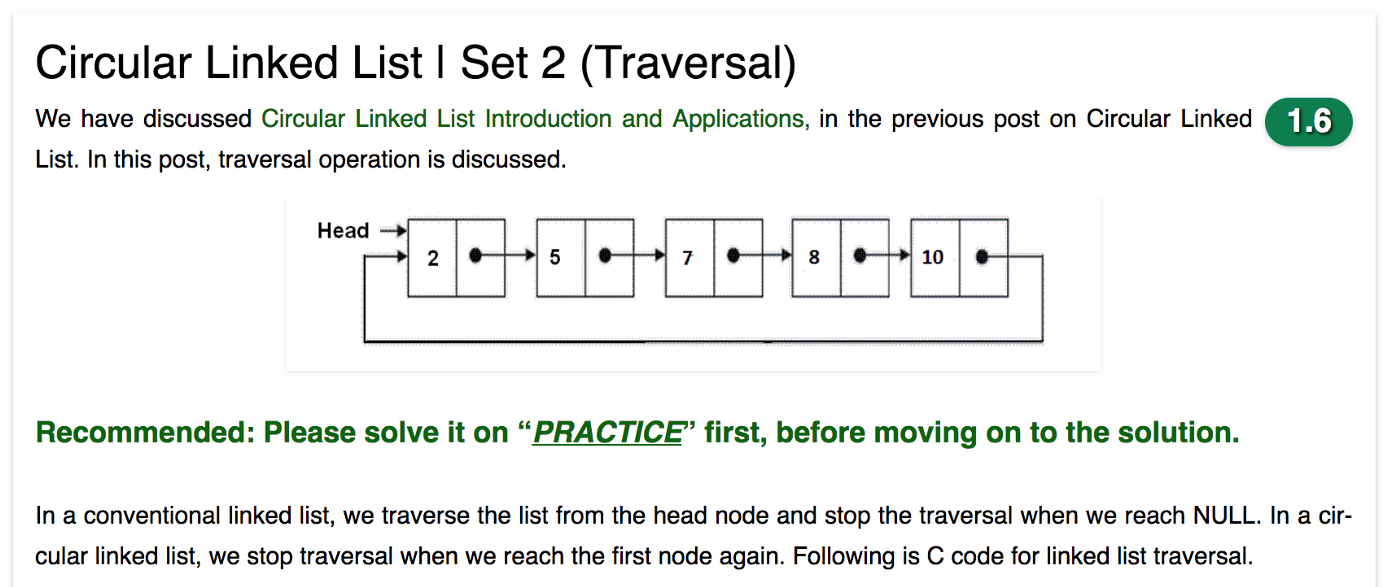
# [8. CodeChef](https://www.codechef.com/)



**Intermediate — Advanced**

CodeChef is an Indian-based competitive programming [website](https://en.wikipedia.org/wiki/CodeChef) that provides hundreds of challenges. You are able to write code in their online [editor](https://www.codechef.com/ide) and view a collections of challenges that are separated into different categories depending on your skill level (see [example](https://www.codechef.com/problems/TREEFUN)). They have a large community of coders that contribute to the forums, write [tutorials](https://www.codechef.com/wiki/tutorial-paying), and take part in CodeChef’s coding [competitions](https://www.codechef.com/contests/).

# [9. GeeksforGeeks](https://www.geeksforgeeks.org/)

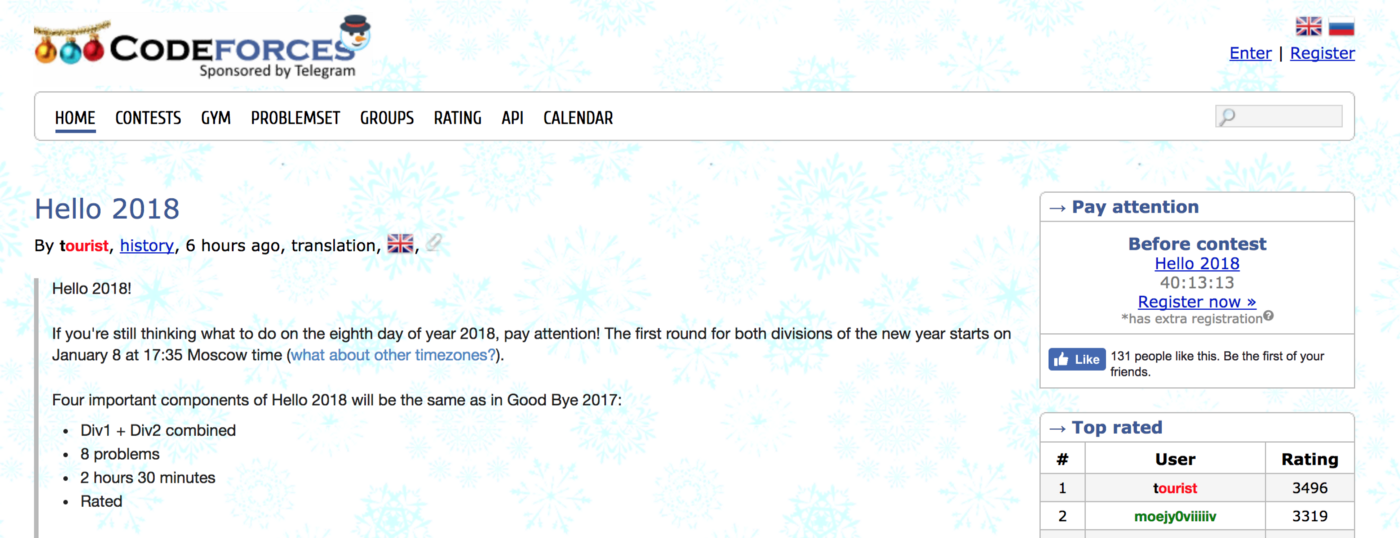


**Intermediate — Advanced**

GeeksforGeeks has the [best collection](https://www.geeksforgeeks.org/fundamentals-of-algorithms/) of articles, explanations, and code solutions for algorithms and data structure topics. You can see an [example here](https://www.geeksforgeeks.org/shortest-common-supersequence/) of how they break down a complex question into simpler parts, explain the solution and then provide a code solution. They also have an online [editor](https://practice.geeksforgeeks.org/problems/rotate-a-linked-list/1) where you can practice solving challenges yourself before seeing the solution.

If you are preparing for a software engineering interview, you should definitely use this website to prepare and brush up on the harder algorithm questions.

# [10. Codeforces](http://codeforces.com/)



**Advanced**

Codeforces is a Russian-based competitive programming [website](https://en.wikipedia.org/wiki/Codeforces) that regularly hosts competitions where some of the [best](http://codeforces.com/profile/tourist) competitive programmers in the world compete online. If you’re a beginner coder, their challenges will most likely be too advanced for you — here are [some](http://codeforces.com/problemset/problem/908/C) [example](http://codeforces.com/problemset/problem/908/C)s of recent challenges. Their challenges usually require advanced math and algorithms knowledge.

# 6)

# CRUD Operations In Firebase using Async Await In Node.js

Hello Everyone this is my second blog on Firebase using node.js, here we will be discussing how to perform CRUD operations In Firebase using the async-await in node.js.

If you are new to Firebase and have no idea about how to connect to Firebase storage and Firestore, don’t worry just refer to my [blog](https://medium.com/@ritik.gupta2018/firebase-storage-and-firestore-with-node-js-for-absolute-beginners-7072f4c1a0f5).

I hope you are now well aware of the Firebase and already had done all the necessary setup as explained in my previous blog.

# ****Adding Data To Firebase****

Before directly jumping to store data, lets first understand how data is being stored in the Firestore.

So basically cloud Firestore stores data in Documents, which in turn are stored in collections.

If you are familiar with node.js mongoose schema then understand this in the following manner, the collection of firebase acts as mongoose schema and each entry of the schema is like a document in firebase and each document can contain various fields such as email, name, password, etc.

**Warning: Don’t mix Firebase schema to mongoose schema, they are different from each other. It is used above just for better understanding.**

Code to add data to the Firebase:

let docRef=db.collection('user').doc('Ritik')let store\_user=docRef.set({email: 'ritikgupta89369@gmail.com',password: 'hello123',});

This is the basic structure of the code that is required to add the data to Firebase.

Facing difficulty to understand the code?

Don’t worry we will go through each line of code.

**db.collection(‘Collection Name’)** creates the collection from the name provided (‘user’ in the above example).

**.doc(‘doc\_name’)** creates the document from the name provided by the user(‘Ritik’ in the above example).

**docRef.set({key:value})** stores each field in the given document.

It’s implementation using REST API and async-await:

|  |  |
| --- | --- |
|  | app.post('/create',async (req,res)=>{ |
|  | let docRef=db.collection('user').doc(req.body.user.name) |
|  | await docRef.set({ |
|  | email: req.body.user.email, |
|  | password: req.body.user.password, |
|  | }); |
|  | res.json({message:'done'}); |
|  | }) |

We can also usethe**add** method instead of the **set**method to add data. The difference between both is that the latter one does not require doc name/id, it auto-generates one itself.

It’s implementation using REST API and async-await:

|  |
| --- |
| app.post('/data',async (req,res)=>{ |
|  | let docRef=db.collection('user') |
|  | await docRef.add({ |
|  | email: req.body.user.email, |
|  | password: req.body.user.password, |
|  | }); |
|  | res.json({message:'done'}); |
|  | }) |

# ****Updating Data From Firebase****

To update some fields of a document without overwriting the entire document, we use the **update()** method.

It’s implementation using REST API and async-await:

|  |
| --- |
| app.post('/update',async (req,res) => { |
|  | let docRef=db.collection('user').doc(req.body.user.name) |
|  | await docRef.update({ |
|  | email:req.body.user.email, |
|  | password:req.body.user.password, |
|  | }) |
|  | res.json({message:'done'}); |
|  | }) |

# ****Deleting Document From Firebase****

It's very easy to delete a document from Firebase, all we need to do is use the **delete()**method in the following way:

|  |  |
| --- | --- |
|  | app.post('/delete',async (req,res) => { |
|  | await db.collection('user').doc(req.body.user.name).delete() |
|  | res.json({message:'done'}); |
|  | }) |

# ****Reading Data From Firebase****

We use the **get()**method to retrieve the data from the Firebase.

To get all the documents present in the collection we can simply use the following code:

|  |
| --- |
| app.get('/get', async (req, res) => { |
|  | let usr=[] |
|  | const users = await db.collection('user').get() |
|  | if (users.docs.length > 0) { |
|  | for (const user of users.docs) { |
|  | usr.push(user.data()) |
|  | }} |
|  | res.json(usr) |
|  | }) |