Front End Developer Guide 2021

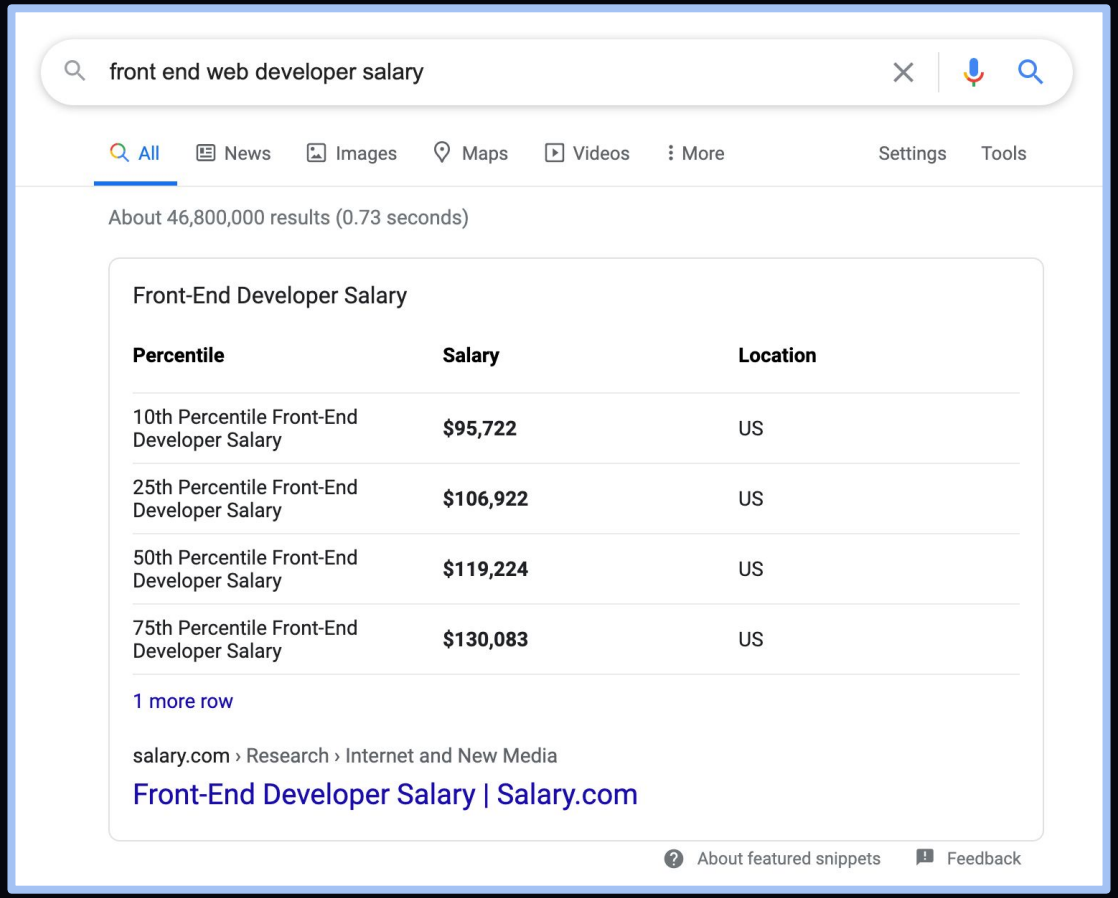
What We'll Cover

- Everything you need to learn to be a job-ready Front End Developer
- The Internal and External game you must play to actually see the finish line
- What does the future hold for front-end web developers
- What's new in 2021
- Shiny Object Syndrome. Why you should ignore the new kids on the block.
- How to build professional experience and add a source of income while learning to code

Why Follow This Guide?

- I started learning to code in 2010-2011. Wrote my first html/css in fourth grade.
- Failed for 5+ years. Broke through in 2015.
- Tutored 1-1 over hundreds of students both locally and remotely
- Mentored dozens students 1-1 (most were bootcampers and some were pursuing master's degrees) All of them have landed jobs. Some even work at top dogs like Microsoft, Hulu.

Why?


front end web developer salary
All
News
Images
Maps
Videos
More
Settings
Tools
About 46,800,000 results (0.73 seconds)
Front-End Developer Salary
Percentile
Salary
Location
10th Percentile Front-End Developer Salary
$95,722
US
25th Percentile Front-End Developer Salary
$106,922
US
50th Percentile Front-End Developer Salary
$119,224
US
75th Percentile Front-End Developer Salary
$130,083
US
1 more row
salary.com › Research › Internet and New Media
Front-End Developer Salary | Salary.com
About featured snippets
Feedback

In my opinion, it is more fun than pure back-end since you get to work on many different aspects of the product. Design. Performance. Code. Colors. Typography. UX. Serverless. Hot new tools. Different frameworks. Hell, you even need to interact with the end customer/client.

A lot of what used to be done by “pure” back-end developers are being done by front-end developers today.

Exponential room for growth. Easier/smoother to get into. Harder to master. Therefore, much more room for growth.

This is probably the worst time to become a front-end web developer in terms of difficulty because today, there is a septillion tools, ways, frameworks, languages, paradigms to do things for front-end and things are changing so very rapidly.

However, this also makes it the best time to get into the field if you're willing to put in the hard work and invest into it 100% because the payoffs are well worth it.

There is always going to be a high demand for good front-end developers because there is always going to be chaos. Developers become obsolete if they don't stay in the game for a few years.

Computer Science.

50 hrs

Fundamentals

- CS50 by Harvard. Best CS course.
- Don’t need to go through all of it. Just go through the first few sections.
- Understand what computer science is, how computer works, compiling, abstraction, bigO notation, efficiency, sorting algorithms, searching algorithms, history of CS etc
- It’s really cool sh*t.

Preparation

5 hrs

Bird’s Eye View

- Take a look at the industry as a whole, look up all the technologies and get a high-level understanding of all players involved
- Understand what computer science is, what software development is, what web development is and the differences between all of them.
- How the web works, what is http, what is a server, what is a client, their relationships etc

0.3

Misconceptions

Bird’s Eye View

- No, you won’t become a developer in 3 months.
- Even with coding bootcamps, people usually study 3-6 months before AND after until they land a reliable job.

HTML. CSS.

Time: 100hrs


DC 6 months ago
HTML and CSS - 100 hours. Let me break this down for you -> HTML - 5 hours, CSS - 95 hours :)
86
REPLY
View 16 replies
1
HTML. CSS.
Time: 100hrs

What?

HTML

Hyper **T**ext **M**arkup **L**anguage

Content. Words/text. Images.

Videos. Paragraphs. Structure.

CSS

Cascading **S**tyle **S**heets

Colors. Design. Layout. Animations.

Positioning. Responsiveness.

Best Resources

- MDN documentation (mozilla developers' network) It's free.
- Go through a book. However, I'm not familiar with any HTML/CSS book that's up-to-date. That's why I recommend the MDN documentation.
- Be very careful on Youtube. It's very easy to waste your time on crappy tutorials. Avoid crash courses unless you know why you're doing it. And if you're a beginner, you probably don't so don't.
- If you're going for free material, make sure they're not wasting your time. If you can afford it, always go for paid.

How?

1. Learn the basics of the languages from a book or the first few things that comes on google. (MDN)
2. Go to your favorite websites. Ex: Google, Facebook, Reddit, Dribble
3. Curate all the micro-components that you like from these sites.
4. Layouts, positioning, flexbox, grid, box model, specificity
5. Start building.
6. Repeat.

The Process

Find

Look for atomic micro-projects to build. For example, navbar, different navbars, hamburger, header, hero, contact form, footer etc

Learn & Build

Don’t think. Start coding. Look up tutorials on how to build a specific thing and start coding. Knowing without taking action is not knowing.

Repeat

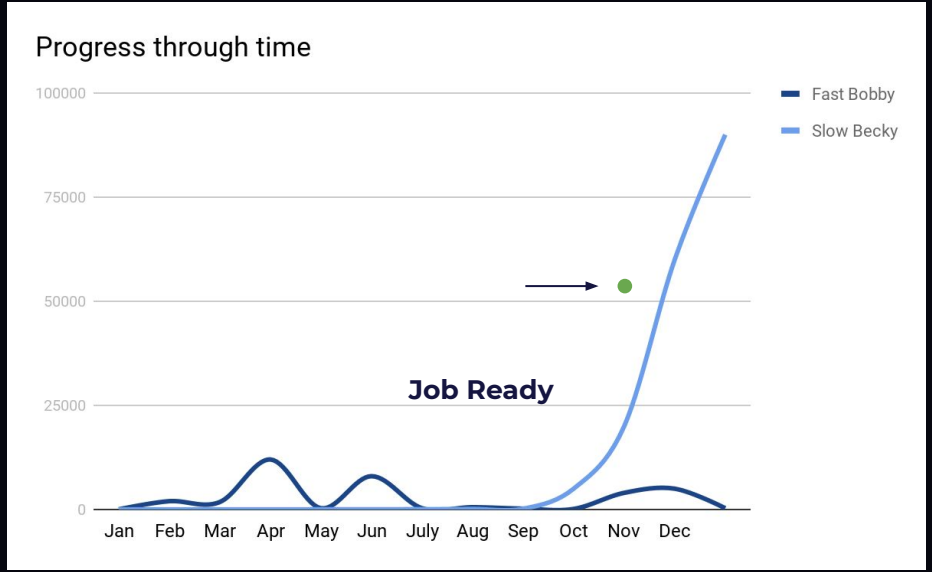
Building something once is not enough. You need repetition. A lot of it. Building multiple navbars, footers, hero, hamburger, contact forms etc

More HTML & CSS

- Responsive Design, Fluid Design, Media Queries
- Typography, Fluid Typography, Responsive Typography
- Css is easy to get started. Difficult to master.
- I know “full-stack” developers with years of experience not be able to do very simple things in css.
- Take your time. Learn it slow. Learn it well.
- If you go fast, you’ll crash.

SLOW === FAST

Knowledge compounds.



The Process

Find

Look for atomic micro-projects to build. For example, navbar, different navbars, hamburger, header, hero, contact form, footer etc

Learn & Build

Don’t think. Start coding. Look up tutorials on how to build a specific thing and start coding. Knowing without taking action is not knowing.

Repeat

Building something once is not enough. You need repetition. A lot of it. Building multiple navbars, footers, hero, hamburger, contact forms etc

When to move on

- You have built over *at least* a dozen micro-components without Javascript
- You have built *at least* 5 -10 single and multi-page websites
- You are comfortable with css fundamentals, positioning, flexbox, grid, media queries etc
- If I told you to make a simple navbar that collapses responsively in mobile and tablet, it should take you no more than 15 minutes.
- **Spend about 100 hrs learning and building things with pure html/css. 49% learning. 51% building.**

Git.
Github.
Command Line.

Time: 10-20 hrs

What?

Git

- Version Control System

Github

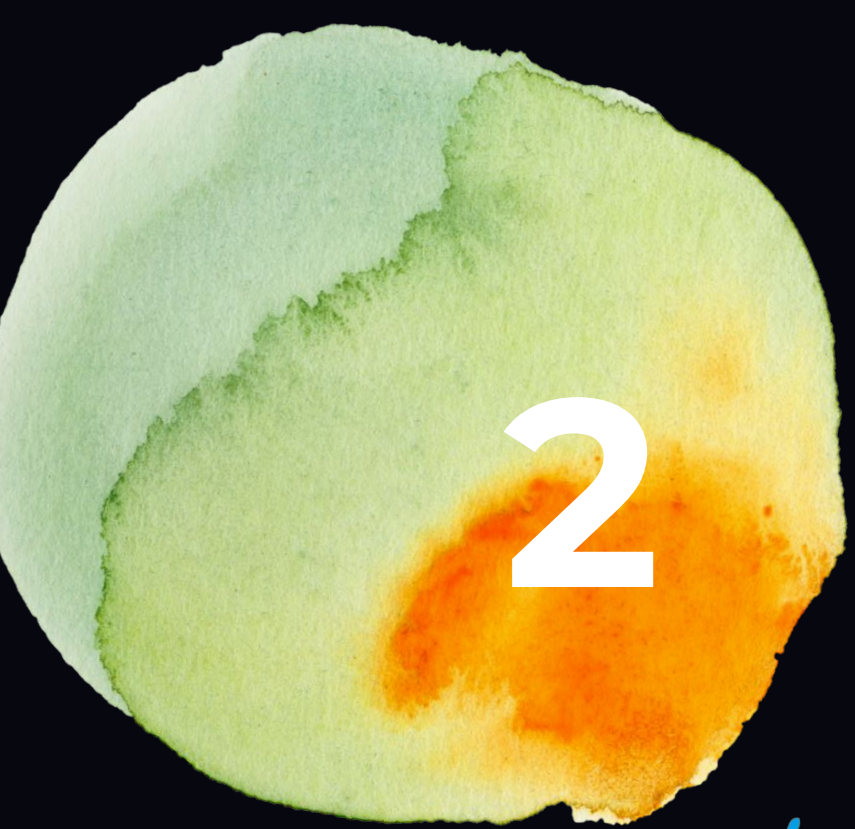
- A website with VCS (git) in the cloud

Command Line Interface

- It is an interface” or a means to interact with the computer in which you type your instructions rather than point and click. (like with a mouse)

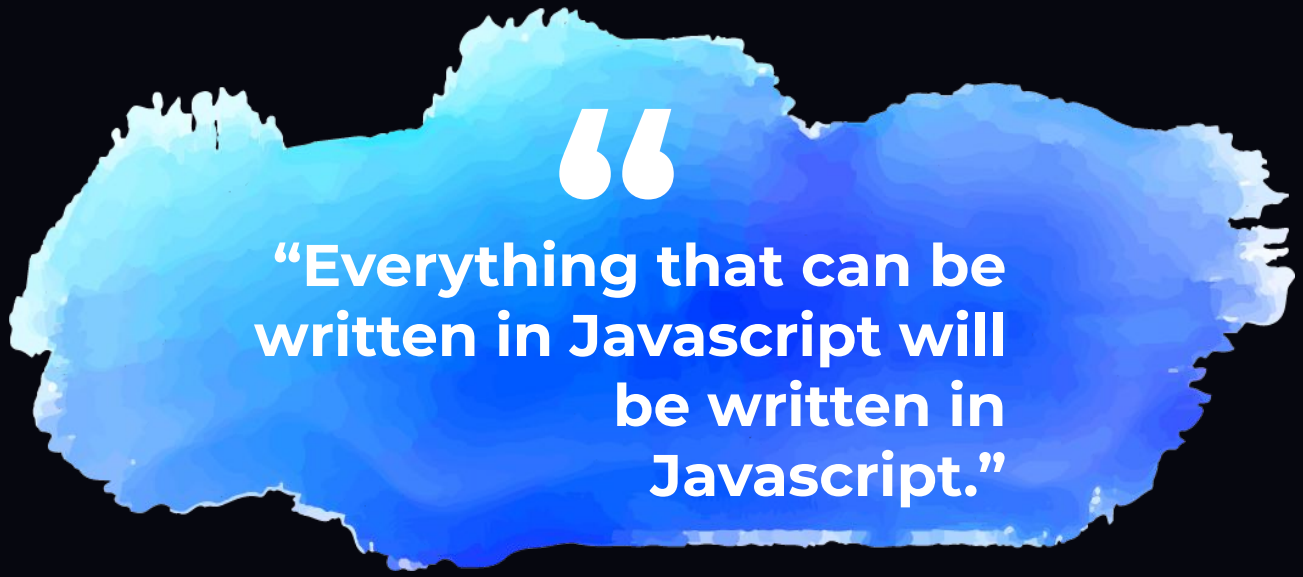
How?

- Git documentation or youtube
- For cli, youtube or documentation
- If you're on windows, install git bash on your computer and start learning unix shell. Don't use powershell. You can also install linux bash (ubuntu) on windows and do it that way. Git bash is easiest.
- As you're building projects, practice with git, github and cli. And keep pushing to github. A lot of employers now look up your github profile to see your work.



Javascript.

Time: 400 hrs


“Everything that can be written in Javascript will be written in Javascript.”

What?

Functionality. Interactivity. Form submissions. Buttons.

Advanced animations. Logic. Games.

Best Resources

Books

- Eloquent Javascript (3rd edition)
- You don’t know Javascript

MDN documentation

Spend more time on in-depth blog tutorials for projects. They’ll be well written and more updated.

JabJabJavascript. One purpose. Get good at pure Javascript.

Two things

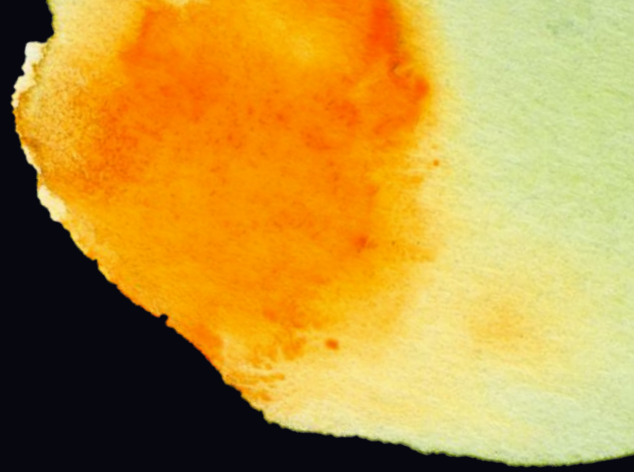
1. **Solve Algorithmic Challenges**

Most people avoid this. Don't. It's going to be tough in the beginning. The only way is through. This will help tremendously and will teach you **how to think like a programmer.**

2. **Build Projects**

If you're not doing algorithmic challenges, you're building projects. Algorithmic challenges will help you with the logic when it comes to building projects.

Don’t quit



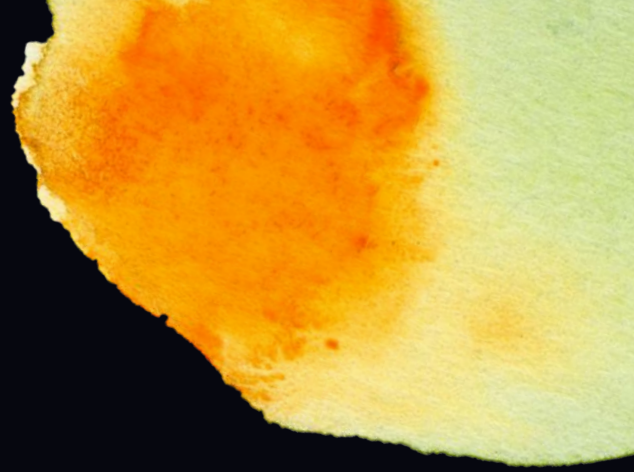
You need to sit with the problem longer. Fall in love with the pain.

When you’re doing research on a project, don’t quit when you can’t find the solution to your problem within the first few pages. Go through dozens of resources until you find them. It’s not a question of whether the solution is there. It’s if you are going to find the solution.

“ It’s not that I’m so smart. It’s just that I stay with problems longer. ”

- Albert Einstein

The Process

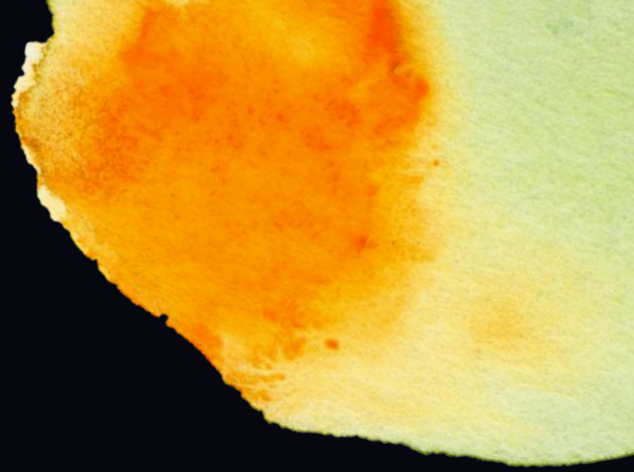


1. Start with a book or/and a good course. If you're going through a course, just make sure the course is good and it's not a buffet. Pause and think about the purpose of the course.
2. Deviate away from the book/course once you have learned something new, think about what's something small you can create with what you've learned immediately. Work on that. Come back and continue.
3. This is why in my course, I constantly tell my students to do 1-3 problems/projects for every 1 thing in my course.

Major Keys

- Javascript Fundamentals (data types, conditionals, functions, logic, objects etc)
- Build common micro components like modal, lists (todos), navbar hamburger, slides, gallery, side navbar, form logic,
- Ajax, Promises, Async Await , es6+
- Prototypal Inheritance, OOP, difference between them
- Pure functions, functional programming, Design patterns*
- **Go through a f**king book.**

Build small things

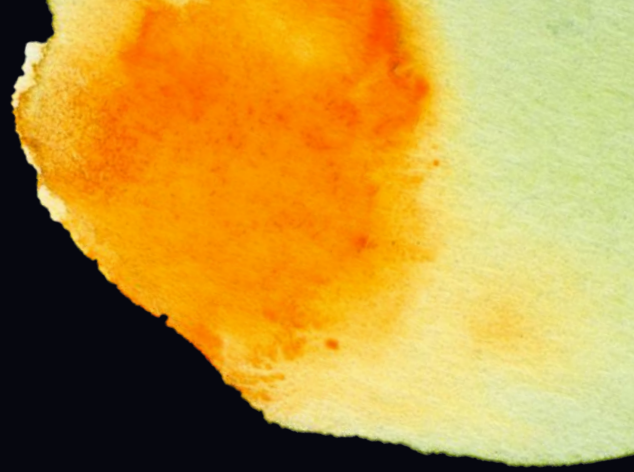


- A large website like Amazon or Facebook is a combination of small components. Example, login form, sign up form, navbar, like, comment, make-post, animation, stories, checkout, Add to cart, search, autocomplete etc
- But if you're going to build anything and anything small, no matter how small, you better make that sh*t good. If I'm going to hire you, you don't need to be the best but you better be good at whatever you say you're good at or have built already.

Larger Projects

- Don’t overthink this. Look at websites that are already built and make a mini clone of it with the major functionalities.
- A Ecommerce site like Amazon
- A forum site like reddit
- A social media site (FB, Tiktok, twitter)
- A music player app like spotify
- Misconception. Large doesn’t necessarily mean the number of pages or components. It depends on the complexity. If you still don’t get it, don’t worry. If you have done all the aforementioned things/projects, you’ll understand at that point.

Design Patterns



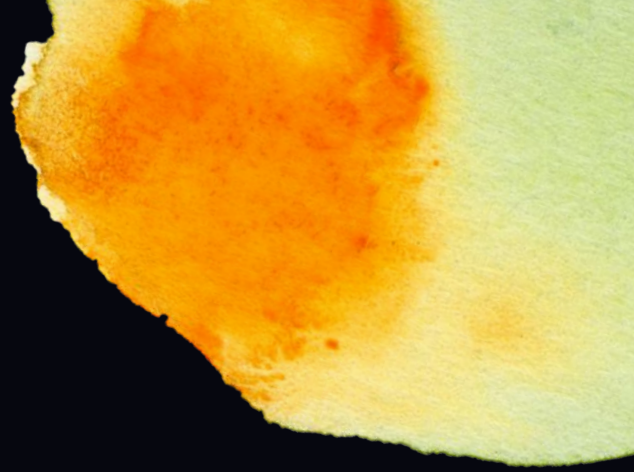
Don’t touch any frameworks.

Build at least 10 projects using pure Javascript. The day you touch jQuery, React, Angular, Vue before you’re ready, you have just shot yourself in the foot and slowed down your rate of progress incredibly.

Learn about different design patterns, why they’re important.

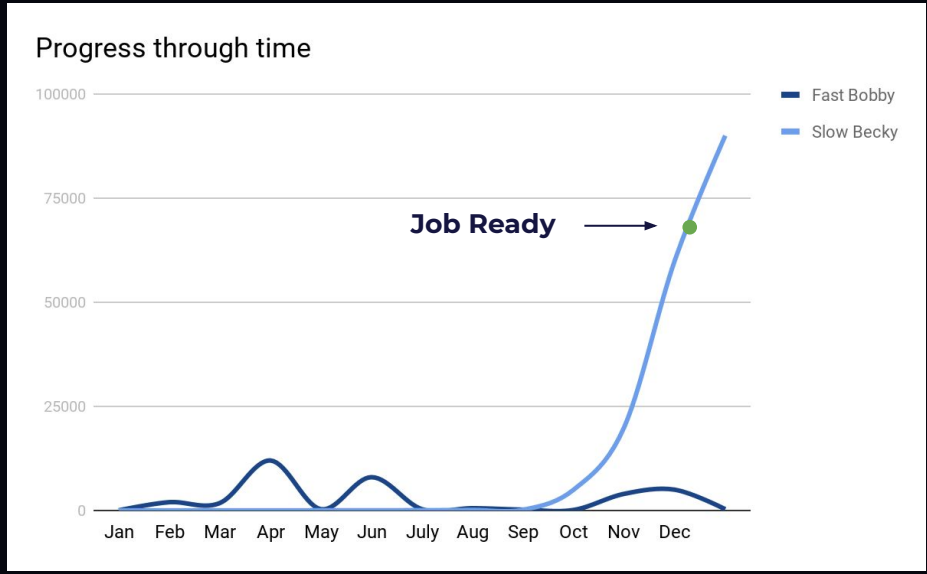
Constructor, Module, Prototype, Factory, Observer, Facade, Adapter etc

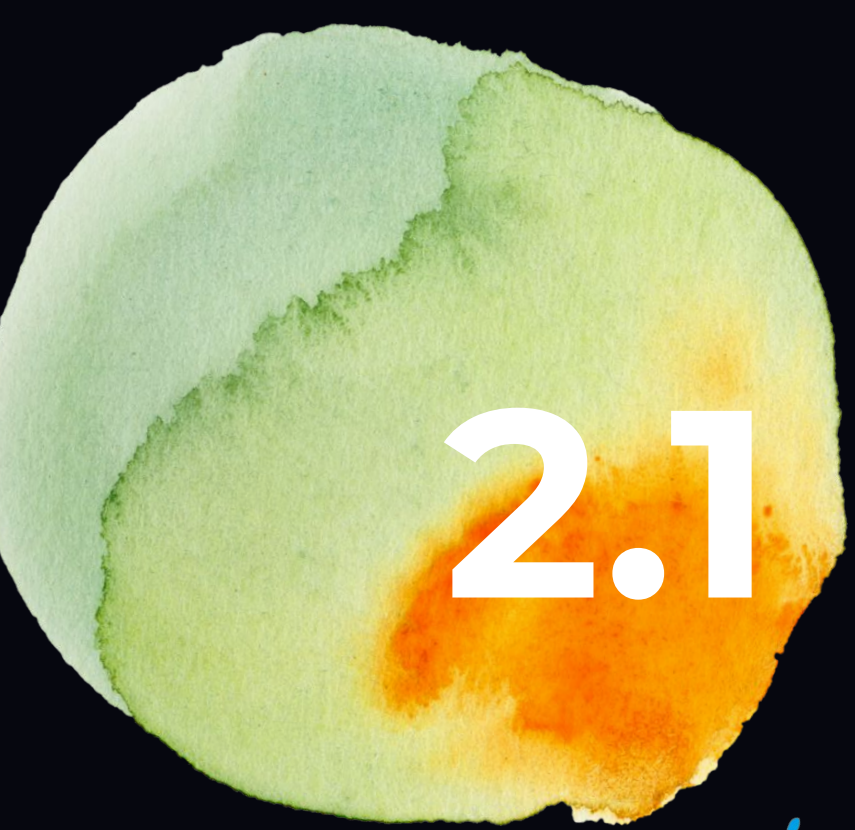
Checklist



- Build at least 8-10 micro projects
- Solve at least 50-100 programming algorithms. Easy to intermediate.
- Understand Javascript fundamentals well
- Understand few of the common design patterns and why they’re important, how they help
- Don’t just make it work. Make it right.
- Learn Webpack at some point. You’ll know when.

SLOW === FAST


Progress through time
100000
75000
50000
25000
0
Job Ready
Fast Bobby
Slow Becky
Jan
Feb
Mar
Apr
May
Jun
July
Aug
Sep
Oct
Nov
Dec



Money.

Make $1k-$3k a month.

NodeJS.

Express. Mongo.

Time: 100 hrs

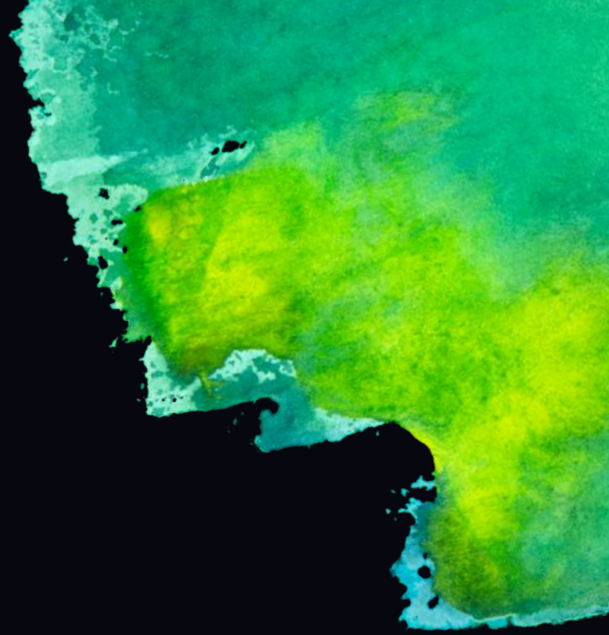
What?

A Javascript runtime. Not a language. Not a framework/library.

Uh…

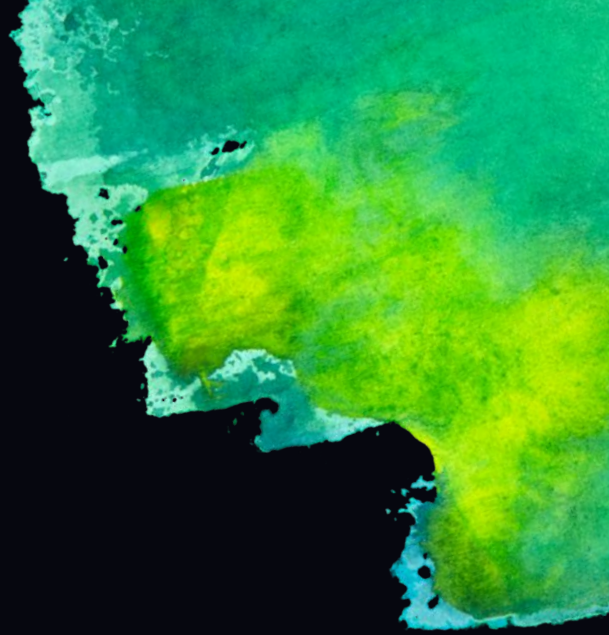
This allows you to write Javascript outside of the browser which means you can do much more than just create web apps.

ok…



Overview

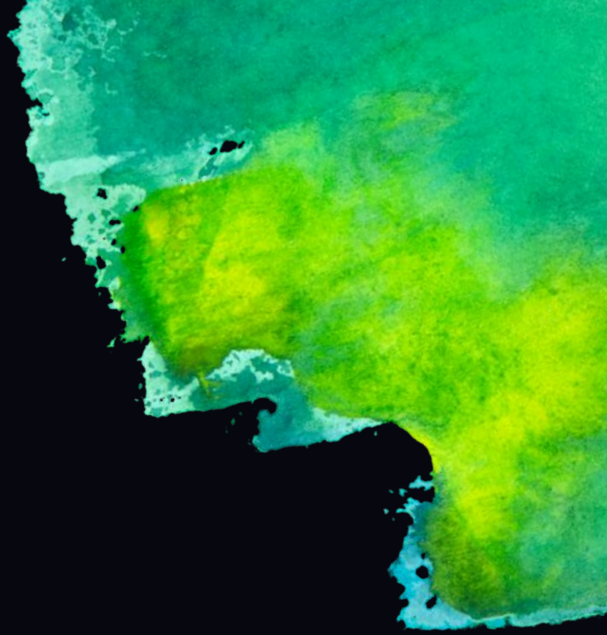
- Javascript is a language. And you write that in a file with the extension “.js”. But you need a web browser to understand and run that code. A web browser contains a “runtime” environment which includes a program (v8 engine) which includes an interpreter that can understand and execute your “.js” file.
- NodeJS is just a program that also contains a “runtime” environment with the v8 engine. But this time, this “runtime” environment has extra features added that allows it to be understood outside of just the browser. (computer, ipad, phone, robots, machinery etc)



Why?

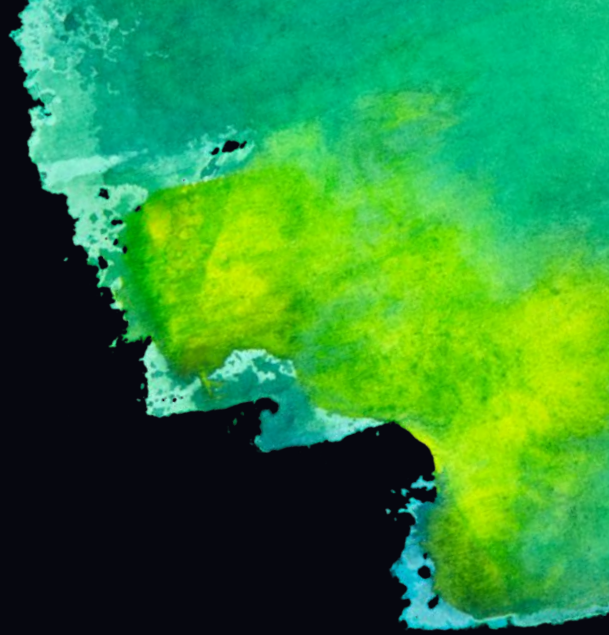
If you want to be a great front-end developer, you must learn the fundamentals of backend or server-side programming.

Dynamic Data allows you to build more “advanced” websites with custom data stored in a database in a server. (blogs, facebook likes, comments, videos)



How?

- Learn how to build APIs (Application programming interface)
- api vs REST api
- MDN documentation, Youtube crash courses, blog tutorials
- Express, databases (mongo)
- a todo list with a database so it doesn't disappear upon refresh
- blog, build anything that takes data from front-end and stores data in the backend



React.

Time: 100 hrs

Many Frameworks

Decide. Stop overthinking and debating on which framework is “best”.

There is no “best”.

Options is a terrible thing. It causes paralysis by analysis. STOP.

Start learning and building.

You learn one well. Done.

I recommend React. More demand. Done.

Why?

Helps you build Javascript applications faster. More scalable and maintainable.

You can do everything with pure Javascript. Doesn’t mean you should all the time.

There’s a reason why frameworks exist. Learn Why. Don’t just jump into it. Learn the WHY.

Flow...

If React js seems hard and you find yourself frustrated and you don't seem to "get it", it's not a react problem. It's a pure Javascript problem.

You should feel excited while learning reactjs.

"whaaaat? this is all I have to do?"

"oh my god! no more state|ui sync nightmare"

State and UI

One of the main reasons why we use these modern frameworks is because keeping the state (your data) in sync with UI (your view) is very hard, especially as your app grows.

For this, if you use pure Javascript, you have to use specific proven design patterns to build applications for scalability and maintainability long term.

State and UI

One of the main reasons why we use these modern frameworks is because keeping the state (your data) in sync with UI (your view) is very hard, especially as your app grows.

For this, if you use pure Javascript, you have to use specific proven design patterns to build applications for scalability and maintainability long term.

Build.

Everything you have built with pure Javascript thus far…

implement all of them with react.

Learn Redux.

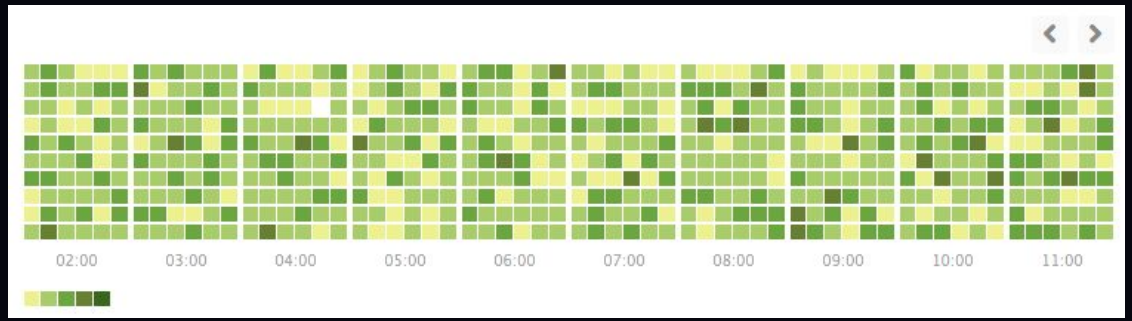
5. Job Ready.

Start applying to jobs like a mad man.

Before you apply...

- Have a simple yet beautiful portfolio that showcases everything you know. Don't tell me. Show me.
- Design matters. Go to sites like behance, dribble and look at phenomenal designs by great UX designers and up your "eye-4-design"
- Work. Confidence is a memory of working hard and winning. Have massive work under your belt.

Your github activity



Work

- 10+ Micro Projects
- 1-2 Large Projects
- Your most important asset: portfolio
- Case studies of one or two of your projects where you talk about what you learned, the journey, the obstacles, cool things you tried etc

  Tell a story.
- 100+ algorithmic challenges solved

Job Guarantee Process

Brute Force Method

Apply to at least ~~20~~ 5 jobs every day. That's 150 in a month. Even if you convert at 1%, that's still 1.5 jobs. Most people don't apply enough. Be ready to move to a different location.

Don't worry about not qualifying. Still apply as long as it's somewhat similar. You can learn things quickly.

Dream Job Process

Sophisticated Monkey Method

- Select 3-5 companies YOU would love to work for. Learn about them. Profile them. Profile their decision maker. What do they exactly want? Make a strategy.
- Send a direct message or personalized email to the decision maker and take them to coffee. Jab Jab Right Hook. Can't land the hook if you don't jab. Never. Ever.
- What can you do to stand out? Be more personal. More human. Do you really give a sh*t?


Good
marketing
starts with a
good product.

SO GOOD THEY CAN'T IGNORE YOU.

CAL NEWPORT

Solify.
Explore.

Time: 100 hrs | Indefinitely

Don’t settle for enough.

Polish your understanding of CSS and Javascript. ⭐

Learn SASS. Learn BEM.

Build larger React/Redux projects. ⭐

Security. Performance. SSR. ⭐

Serverless is the future. GatsbyJS. ⭐

Learn a new language like clojure, scheme, or c++.

(Learn/build things that excite you.)

Congratulations

If you have made this far, you're in the very small minority of people who didn't give up and actually made it through.

Most people have given up at this point. It makes sense. It's hard work. But you love hard work. We love hard work. Hard work is our competitive edge.

Decision.

I would start. Then quit. Start. Then quit again. And my progress just oscillated to a zero for so many years.

Then I committed. I decided. One thing.

Certainty.

Most people fail to take action because of uncertainty.

Uncertainty means you got doubt.

Doubt means paralysis by analysis.

Paralysis equals inaction. Lack of action equals guilt. Now you've got the cocktail of the devil.

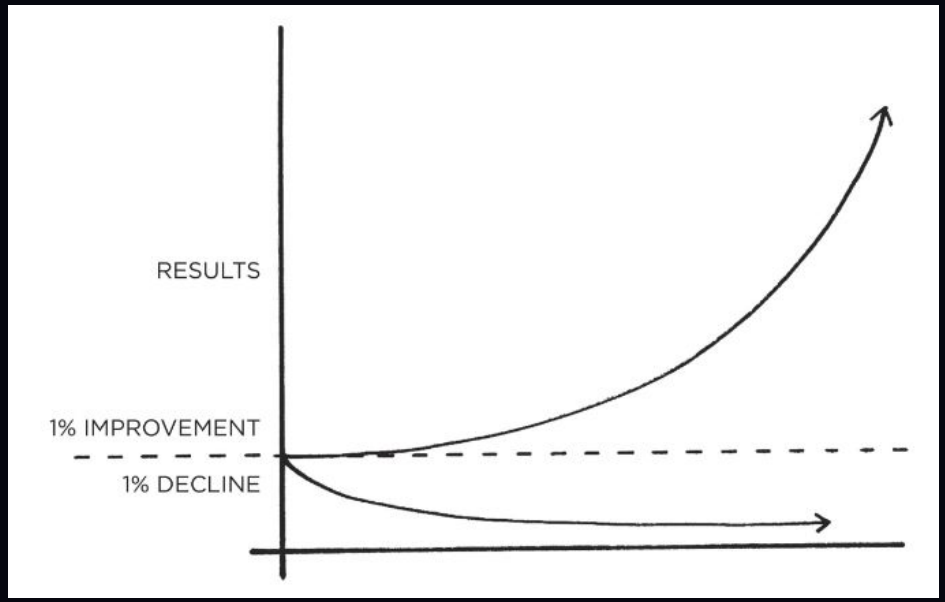
Don't quit. I'm telling you right now that I'm certain each and every one of your faces can make this happen.


action.
The beauty of action is that it’s loyal.

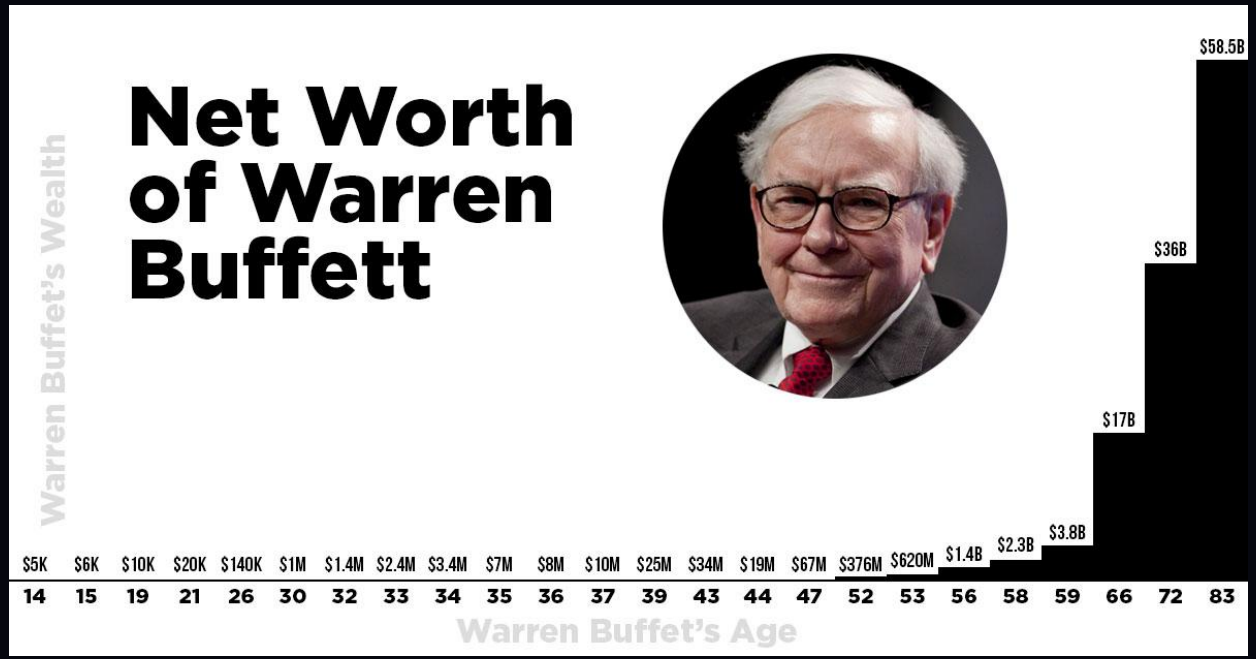
Compound Interest is the 8th wonder of the world.

- Albert Einstein

Success = micro work x time^2



Success = micro work x time^2



Conclusion

- Work. 800-1000 hrs.
- Sleep at least 7 hrs a day
- Don’t rush. Life is long.
- Have fun while doing it. If Victor Frankl can find peace/happiness in the camps, you can definitely do it while learning to code. Worldview matters.
- Every single one of you can do better than me. Question is “Are you going to put in the work?”
- Don’t just do it for the money. You won’t last.

Bonus

- Unfollow everyone on Instagram and social media
- Disable ALL NOTIFICATIONS on your phone except for phone calls.
- Hit the gym. Eat cleaner. SLEEP.
- Sacrifice. Going out. Drinking. Vacations. Parties.
- Go to meetups and find like-minded people.
- Your environment is the ultimate predictor of your success.
- Success is the residue of good design.

Most Important

1. Focus - You must sacrifice all of your other “goals”. If you want to win, you have to focus on one thing. Focus requires sacrifice. You have to go ALL IN.
2. Recovery from deviation - You will deviate. You will fall off. It’s not an “if”. It’s a “when”. The key to success is doubling your rate of failure. The faster you recover, the better.
3. Systems over goals - Forget thinking about big visions and grand dreams. Just master today. JUST TODAY. Every day is a battle. Every single day is incredibly hard. Action beats anxiety.
4. Environment - Maybe the most important thing of all. If you have the right environment, everything will be easier.

That’s it.

If you’re still watching, thank you.

Stick around for a few minutes more. Wanna share something cool that I’m launching right now with this video.

And I think it ties in perfectly with this frontend guide video as well.

The Problem

Learning to code is hard. Doing it alone is harder.

The biggest reason I failed for so many years when I started my coding journey was because I didn't have a community to do it with since I went the self taught option. I was also very antisocial back then.

Stackoverflow was brutal. It destroyed my will to learn or do anything.

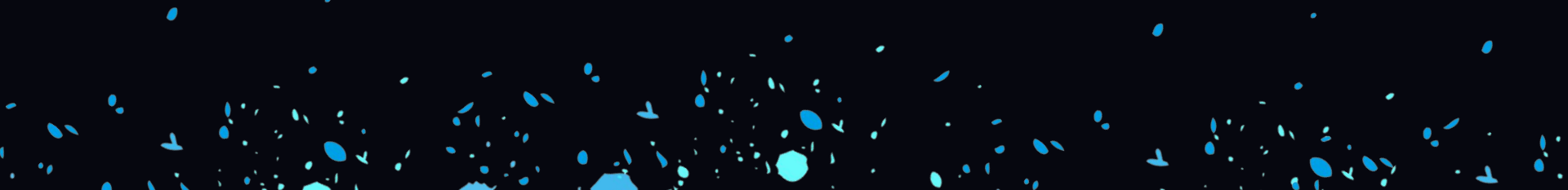
Googling is just that. A black hole of information with all fine sounding ideas. Everytime you google, there's a big chance of you finding a new shiny object to avoid doing actual work.

Information is not the answer.

It’s not about:

- The best online course
- The best framework, tool, language
- The best bootcamp or college
- The best roadmap/guide
- The best strategy

If information was the answer in 2021, we’d all be millionaires and six pack abs.



Transformation requires something else

Following a good roadmap or path. (...like this video)

Taking the right vehicle. (College is not needed. You can do it faster and cheaper online.)

And taking action on it for specific period of time. (12-24 months)

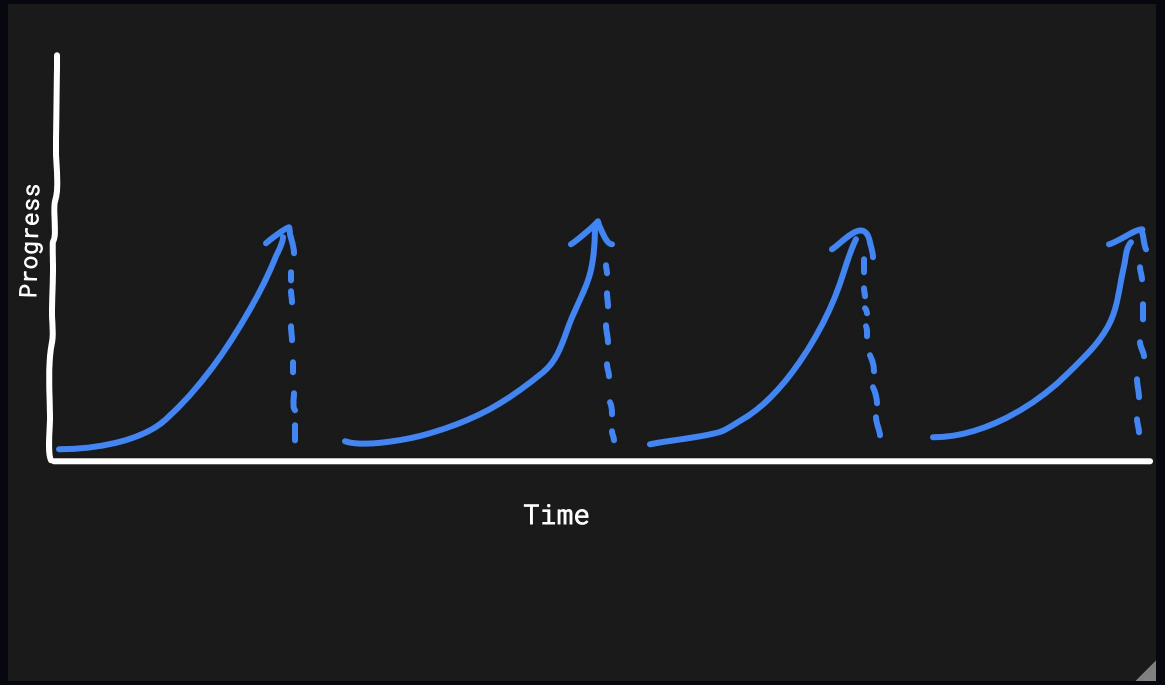
1% of people do this effectively. 99% fail. (I failed like a madman)

Most people never follow through

I’m really bad at this.

I’m still bad at this. And that’s why I need to hack my environment to succeed.

99% of the people

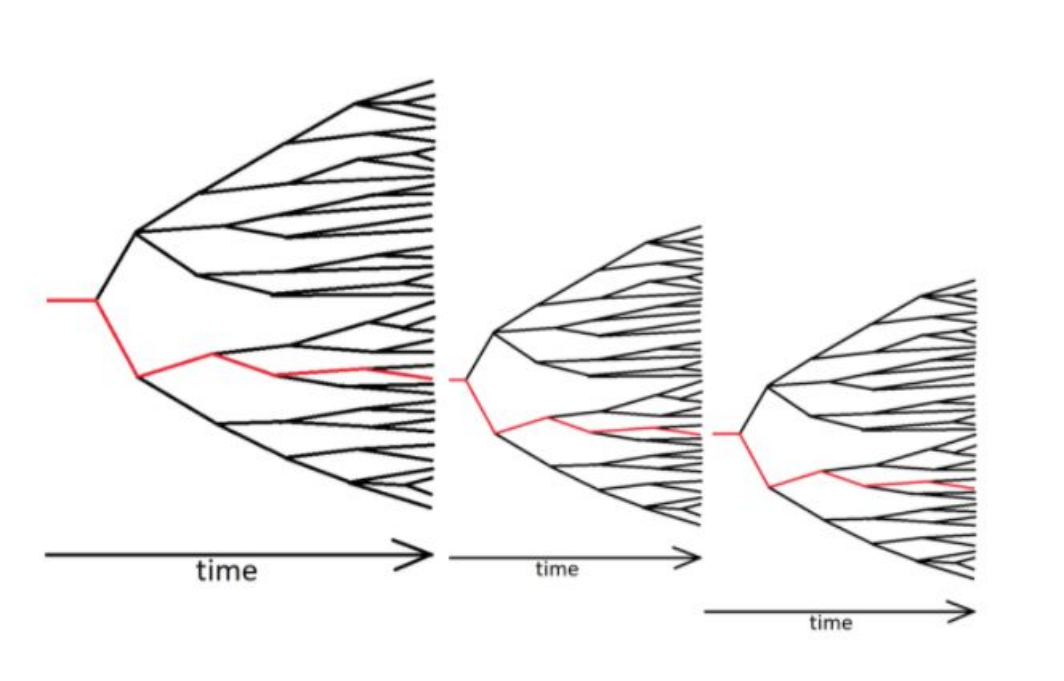


Join the Inner Circle Dev Community

- Goal: Become a better developer and land your first dev job
- A QnA forum where you can ask me any questions at any time
- People just like you who are struggling with staying on the course of being a successful developer whether it be landing a job or starting a business
- Weekly raw unfiltered content by me about career choices, marketing, business, dev jobs etc
- Weekly livestream zoom QnA calls for the first 2 months of launch
- Bonus content for people who commit for yearly membership

Without a community...

Success is not linear.



If you want to become a front end dev in 2021, join the tribe.

whatsdev.com/community

Any questions?

Email
tenzin@whatsdev.com