

THE ULTIMATE

GUIDE TO GETTING STARTED AS A SOFTWARE ENGINEER

THE ACCOUNTABILITY, INSIGHTS, AND KNOWLEDGE FOR SUCCESS

BY RANDALL KANNA

This guide will help you
get started coding with
low risk to your financial
future and time.



Table of Contents

Typical Day of an Engineer	4
Setting up your IDE and Terminal	9
Basics of Programming	15
Git + GitHub	21
HTML and CSS	23
Databases	26
Staying Motivated	27
Find a Mentor	29
Computer Science Fundamentals	30
Picking a Specialty	31
Google is Your Best Friend - How to Find the Answer for Everything	33
Building a Portfolio: How to Accelerate Your Coding Abilities Fast	34
What About a Bootcamp?	35
Job Search	38
Acing the Interview	38
Networking	40
Starting a Technical Blog	41
Day One of a New Career	42



People reach out to me all the time and ask ‘How do I get started?’

I will tell you what I tell them: it is not an easy feat to jump into engineering successfully without a mentor or paying a lot of money for a bootcamp.

I attended the original Bootcamp, Dev Bootcamp, and my cohort started with sixty people. By the time of graduation only 10 remained. It is not uncommon for students to join a bootcamp and realize halfway through that the program isn’t for them. I want to help you avoid that fate.

With this Guide, you’ll be able to get your feet wet sorta speak with low risk to your financial future. You are going to find out if coding is for you before you enroll in an expensive bootcamp, go back to school for a CS degree, or quit your job to become an engineer.

When you’ve finished this Guide, you’ll know how you *can* become a software engineer and more importantly—if it is a good career choice for you.

Let’s get started!



Typical Day of an Engineer

One of the most fundamental things in choosing a new career is determining if the day-to-day and the work will be a fit for you.

Being an engineer is all about constant learning. Technology is constantly changing so you'll need to get used to feeling *uncomfortable*. **All the time.** You'll always need to be learning and adding more skills to your toolbox. That's a great thing about becoming an engineer—you will never feel bored!

You'll always be learning something new and doing something new at your job. Engineering may not seem like it, but it's a very creative field. You'll never be doing the same thing day after day.

Another great thing about being an engineer is flexibility. You could work in some new cool tech startup, a finance company, a non profit, or a biotech company. The opportunities are endless. Almost every company needs engineering teams.

The dream job for many? You can get a remote job at a startup and work from home in your pajamas.



Here's what a typical day in the life of an engineer working at a startup office looks like:



9-10AM

You'll most likely just be getting to work. (This might even be the earlier side for a lot of engineers). You'll probably check to see if you have any urgent slack (<https://slack.com/>) messages from your team. If you're working for a San Francisco company, you'll probably enjoy a great breakfast at the office. Some companies have great french toast for breakfast. :)



10-10:30AM

Standup. A standup is a meeting with your product manager (someone who keeps the team on track, determines what's important to work on next, and more), your fellow engineers, and potentially a designer. You'll go around one by one and provide updates to share what you worked on yesterday and what your plan is for the day. Sometimes you'll discuss the blockers you have. A blocker is an engineering term for when you're stuck on something and might need the team to help.



10:30-11AM

You'll start to dig into some code. You could be continuing a ticket you started another day, fixing a bug, creating documentation, or adding tests to an existing feature.



11AM - 12PM

Technical interviewing. Your company is expanding so it's likely that you'll need to give a technical interview to a potential candidate. You'll either pair program with them on a problem or ask them some engineering questions. If you're working at a smaller company, this will be pretty infrequent. But as your company grows or if you start off at a large company, eventually you'll be expected to give technical interviews for your team and determine if someone is a good fit.





12PM

Sometimes if the day is busy, you might work at your desk through lunch. On other days when you're not in crunch mode, you'll enjoy the company provided meal or go out to lunch with your coworkers.



1-2PM

You might spend some time in your day fixing a bug. When a user has an issue with your application, you'll need to figure out first how to recreate this bug, and then to fix it in the code. After you determine the issue locally, you'll need to deploy it to production to help your users.



2-6PM

The most productive part of your day will be a large chunk when you don't have any meetings so you can go heads down and focus on your code. You might need to create a new React component with tests that you'll focus on. Or you might need to build a new API so users can access data from your application.



7PM

You'll head home. A lot of engineers take their work home, but you don't have to if you're productive during core working hours. For me, I'm most productive late at night or on weekends when I can be fully focused with no meetings and let my mind be most creative.



11PM

You'll probably dream about code.



Other meetings an engineer might have:

- **Sprint planning** - At the beginning of a new sprint (a two week period where you are assigned some work and are expected to finish that), you'll meet with the entire team and individuals will be assigned tasks and you'll discuss priorities for that sprint.
- **Retros** - Every two weeks, you'll most likely have a meeting with your entire team where you discuss what went well and what went wrong on the team that week.
- **One on ones** - You'll definitely have a manager that will meet with you either once a week or a few times a month. Your manager will ideally help you to make sure you're meeting your own goals, advocate for you to get your next promotion, and let you know how you're doing in the company.
- **Giving technical interviews** - At some point in your career, you'll need to give technical interviews at your company. You, along with a group of other coworkers, will be asked to assess someone's technical competence (see how well they can code) before they give that engineer a hire.

Here's what my schedule looks like as a remote engineer.

(Of course, it's a lot harder to find a remote job as a junior engineer but not impossible).



6-7:30AM

Wake up, eat breakfast, read and meditate. Plan for my day. Sometimes fix a few small things at work like a bug or catch up on emails or slack messages.



7:30AM

Standup with my remote team. We chat a bit at first about random stuff and then everyone provides an update





8-4PM

Right now, we're working on improving our products and creating new features for our customers. I'm really proud of the features I've helped the company launch over the last year. We're trying to make executive assistants' jobs easier and we've released a lot of cool new stuff lately. I work as a full-stack engineer so my job varies a lot. Sometimes I'm creating a new UI for our users and sometimes I'm working with our database and creating new API's that the frontend can consume. Some afternoons I'll have a 1-1 with my manager where we get aligned on goals and priorities for the company. Occasionally, we'll have a team retro about every two weeks or a team huddle where the entire startup gets together to talk and discuss priorities.



4PM

Generally since my work day starts so early, I can sign off a little earlier which is nice so I can get some rest for the next day!



ASSIGNMENT:

Reach out to an Engineer on Twitter or LinkedIn and ask a few questions about their job.

Tip: When reaching out to someone through social media, it's best to send a few prepared questions opposed to just asking for their time. If you put together a list of three questions that will answer some questions about engineering for you. You're more likely to get a response if someone knows upfront what kind of time commitment they're making.



Setting up your IDE and Terminal

To become an engineer, you'll need to be able to use an IDE. An IDE is an integrated development environment. An IDE will be where you will write code. If you were a writer, you would use Google Docs or Microsoft Word. If you were an analyst, you would use spreadsheets. If you were a professional speaker, you'd use Powerpoint. As a coder, you'll use Atom or VsCode or Pycharm.

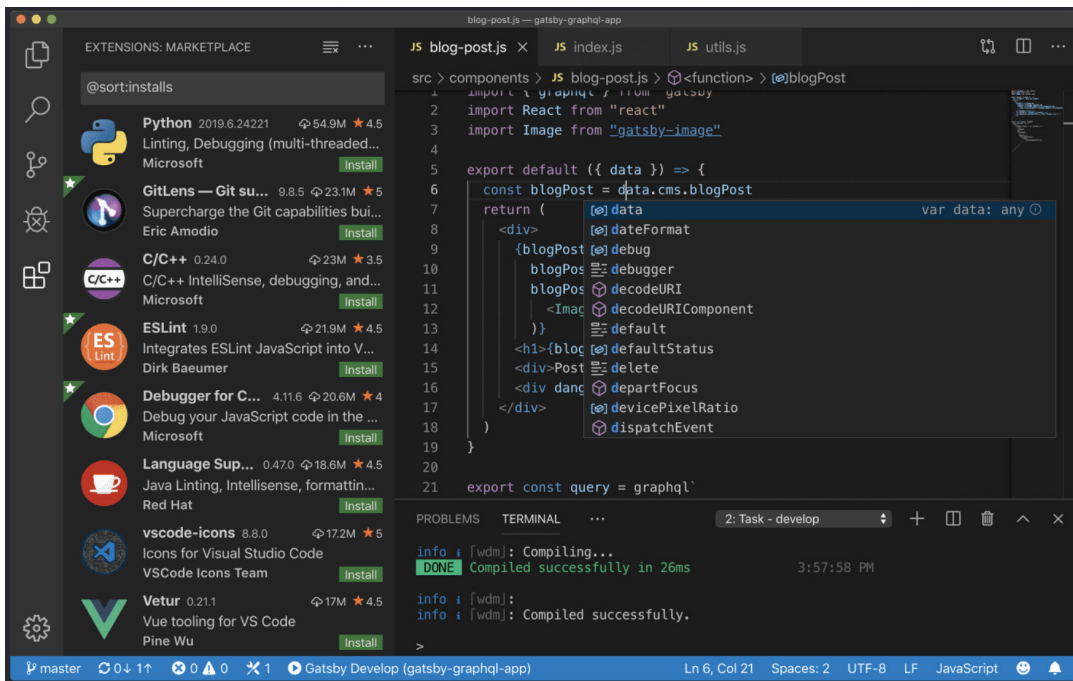
An IDE can do all sorts of useful things for engineers. They include syntax highlighting which lets you know if you do something wrong, or if your code is messy and won't make your teammates happy. An IDE can even autocomplete some of your lines of code. You can run tests inside of some IDE's and even debug your code.

Right now, one of the most popular and easiest to use IDE's is Visual Studio Code. I personally like to suggest that beginners use Visual Studio Code because it has the most beginner friendly resources.

Let's get started by downloading it and getting it running.

Navigate to <https://code.visualstudio.com/download> and download Visual Studio code. Follow your machine's prompts and get the installation started.





Once you create projects, you can open your project in your editor and write code to the files there. We're going to create a file shortly so keep reading!

Resources

Introduction Videos: <https://code.visualstudio.com/docs/getstarted/introvideos>

Setup: <https://code.visualstudio.com/docs/setup/setup-overview>

Tips and Tricks: <https://code.visualstudio.com/docs/getstarted/tips-and-tricks>

What is an IDE? <https://www.codecademy.com/articles/what-is-an-ide>

Next, let's learn about using a terminal.

What is a Terminal?

The terminal is where we can type in commands and execute them.

We can run many different commands in the terminal. We can install software, run the frontend or backend for our code, push up code to GitHub, deploy code, or run tests.





iTerm2

iTerm2 is a terminal emulator for macOS that does amazing things.

[Home](#)[News](#)[Features](#)[FAQ](#)[Documentation](#)[Downloads](#)[Donate](#)

What is iTerm2?

iTerm2 is a replacement for Terminal and the successor to iTerm. It works on Macs with macOS 10.12 or newer. iTerm2 brings the terminal into the modern age with features you never knew you always wanted.

Why Do I Want It?

Check out the impressive [features and screenshots](#). If you spend a lot of time in a terminal, then you'll appreciate all the little things that add up to a lot. It is free software and you can find the source code on [Github](#).

How Do I Use It?

Try the [FAQ](#) or the [documentation](#). Got problems or ideas? Report them in the [bug tracker](#), take it to the [forum](#), or send me email (gnachman at gmail dot com).

[Download](#)

We can even play around with code in the terminal. If you have a MacBook, you'll have a terminal that comes installed. I prefer to use <https://iterm2.com/> though. But either will work. Go ahead and either install iTerm2 or use the built-in terminal computer.

Note: Be wary of running commands at first if they aren't from a very popular tutorial or website. Don't follow any commands that someone gives you if they look suspicious. Use your common sense and read through things. When you just start out, you can stick to very basic commands.



We'll also take this opportunity to install a few things. This course is going to focus on JavaScript. So, let's install Node first. Node is a runtime environment that you can use to write the backend of your application. We'll discuss the difference between the frontend and backend later. For now, the backend is what stores your user data and returns data.

Let's play around with the terminal. Let's start by creating a file in vscode.

1. Create a folder in your documents called 'code'
2. Next, install Node on your computer. Follow this tutorial:

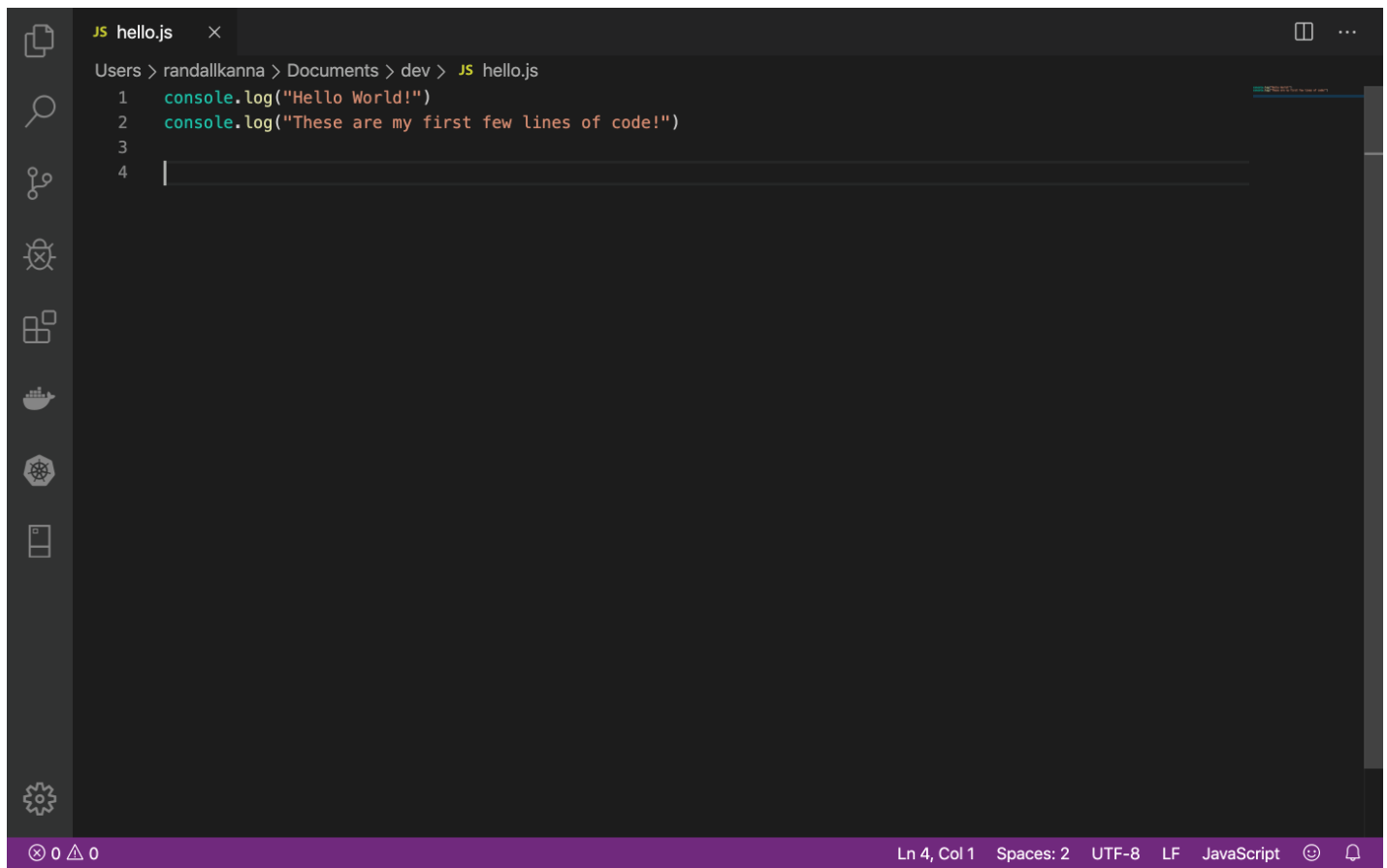
<https://blog.teamtreehouse.com/install-node-js-npm-mac>. Remember, being an Engineer is a lot about being stuck and being able to google well and figure things out. So if you get stuck or see an error message, copy and paste it into google and figure it out! If you get REALLY stuck, create an account here <https://stackoverflow.com/> and post your question. Stackoverflow is a great community where developers can post questions and get answers. It's also not a bad idea to start building up your Stackoverflow street cred either by having an account.

3. After you have node installed, create a new file titled hello.js and save it to the new folder you created. Inside that file, add the following lines of code

```
console.log("Hello World!")  
console.log("These are my first few lines of code")
```

Tip: Did you remember where the code needs to go? Add your code in Visual Studio Code.





The screenshot shows a code editor window with a dark theme. The title bar indicates the file is 'JS hello.js'. The breadcrumb navigation shows the path: 'Users > randallkanna > Documents > dev > JS hello.js'. The code content is as follows:

```
1 console.log("Hello World!")
2 console.log("These are my first few lines of code!")
3
4
```

The status bar at the bottom displays: 'Ln 4, Col 1 Spaces: 2 UTF-8 LF JavaScript' along with icons for error, warning, and search.

4. Navigate to where the file is in your terminal. `cd documents/code`
5. Run this command: `node hello.js`
6. If you followed all the steps, you should see 'Hello World' printed out in your console.



```
→ ~ cd Documents/dev  
→ dev node hello.js  
Hello World!  
These are my first few lines of code!  
→ dev █
```

Congratulations. You wrote your first portion of code and saw the output in the terminal. That was your first step towards becoming an Engineer.



Basics of Programming

For this course, we're going to focus on JavaScript. JavaScript is a dynamic programming language. JavaScript can be used both in creating frontend applications with React or backend applications with Node.js. JavaScript is one of the most popular programming languages out there. I strongly suggest you focus all of your attention on JavaScript. It's the most multi-purpose and popular programming language you can learn. It's the best language to start out with for a beginner as well because of the many resources available online free centered around JavaScript.

To play around a little with some basic programming concepts, let's use our console. First, let's open up your console. If you're using Chrome (definitely use the Chrome browser for developing! It's the best for engineering), simply press `opt + command + i`. If you're using another browser, you can google 'How to open developer tools' in whatever browser you're using.

Here are a few resources to learn how to use your developer tools correctly.

<https://blog.galvanize.com/how-to-use-chrome-devtools-for-the-absolute-beginner/>

<https://developers.google.com/web/tools/chrome-devtools>

<https://www.keycdn.com/blog/chrome-devtools>

Variables

Variables are how we store information in JavaScript. For instance, if we wanted to store a reference to our favorite food, we could create a variable like this,

```
let favfood
```

And then we could store our favorite food inside the variable we just created.

```
favFood = 'pizza'
```



Storing it in quotation marks like that, means it's a string.

We can also declare variables with their value directly as well.

```
const favFood = 'pizza'  
var leastFavFood = 'spinach'
```

In the example above, these are both variables that are holding a string. Variables can also hold numbers or even more complex data.

We can also hold numbers in our variables. Not just strings.

```
var two = 2  
let three = 'three'
```

Arrays

We can also hold many different types of data in an array. An array is a variable that is used to store different types of data.

An array looks like this:

```
var leastFavFoods = []  
// above is an array  
// this is a code comment  
// below is our full pizza array!  
const favFoods = ['pizza', 'ice cream', 'cheese']
```

In this example, our data structure is an array. Instead of just holding 'pizza' like in the above example, it can hold many different types of data. Below is an example of a more complex array that holds bold strings and numbers.




```
const favNumbers = ['5', 1, 4, "seven"]
```

Basic Programming Concepts

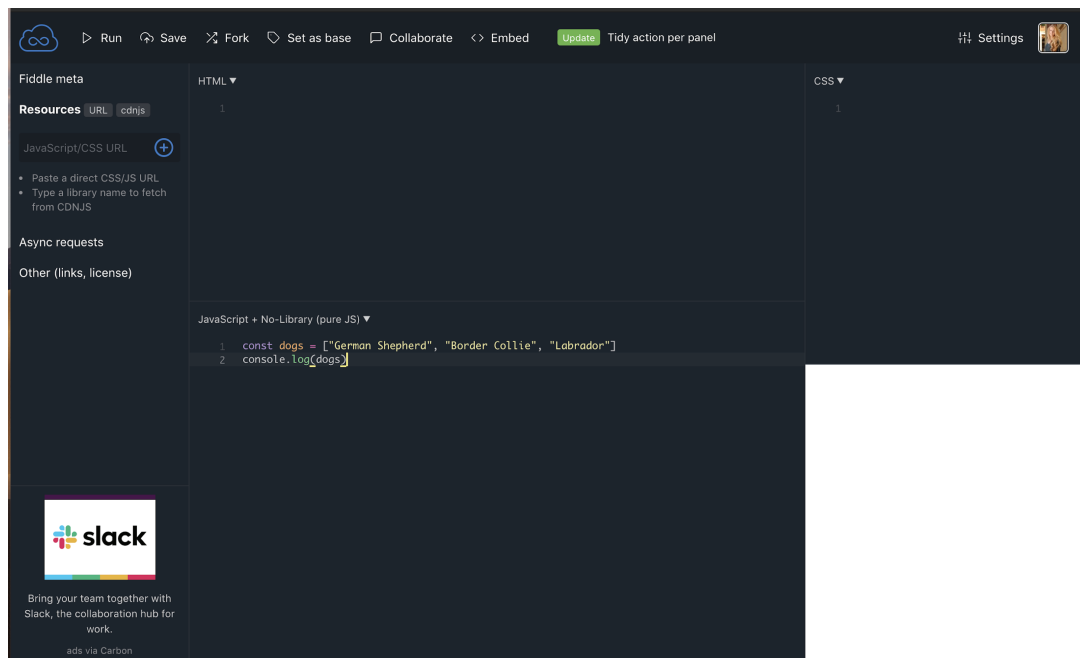
Let's take the few concepts we've just learned and use them. Let's stick to the console for now to make it easy. Another great resource if you want to try something new is to use is <https://jsfiddle.net/>. You can experiment with HTML, CSS and JavaScript right in the browser.

First, create an array that contains strings and print it to the console.

Let's make it about... dogs.

```
const dogs = ["German Shepherd", "Border Collie", "Labrador"]  
console.log(dogs)
```

Go ahead and press the "Run button" in JSFiddle.



Gotcha. Nothing happened. Did you figure it out? If not, that's ok. We just need to open our console to see our array printed.

```
▼ Array(3) ⓘ  
  0: "German Shepherd"  
  1: "Border Collie"  
  2: "Labrador"  
  length: 3  
  ► __proto__: Array(0)
```

Let's try out a fun concept. Let's create a function and print out the breed of each dog that we created in our array earlier. We need to start by creating a function. A function is a programming concept that lets you call an action on something to perform a task.

Here's what a function looks like:

```
const dogs = ["German Shepherd", "Border Collie", "Labrador"]  
console.log(dogs)  
  
const myDogFunction = (dogsArray) => {  
  // this is where our function contents will go later  
}
```

Above, we've created a function called `myDogFunction`. Do you see the "dog" text after our function name and equal sign? That is our functions params. A param is something that your function takes in and performs an action on. In our case, we will call our `myDogFunction` and pass in our dogs array that we created.



Add the code below. Let's use a JavaScript console.log and see if our function is being called.

```
const dogs = ["German Shepherd", "Border Collie", "Labrador"]
console.log(dogs)

const myDogFunction = (dogsArray) => {
  console.log("This worked!")
}

myDogFunction(dogs)
```

Now, let's run our code by pressing the Run button in JsFiddle and see if it works. If your code works, you should be able to see your string printed in the console.

► (3) *["German Shepherd", "Border Collie", "Labrador"]*
This worked!

Our function isn't very exciting though so let's fix it up a little. Let's use a programming concept called loops and print each dog breed along with a string. A loop is an instruction that is repeated until a certain condition is repeated. Basically, if we create a loop, it will iterate through each item in the array that we pass in and perform an action on it. Let's create something simple.

```
const dogs = ["German Shepherd", "Border Collie", "Labrador"]
console.log(dogs)
// ignore all the old code above

const myDogFunction = (dogsArray) => {
  // below is our 'loop'
```



```
// it will iterate through each item in our list
dogsArray.forEach((dog) => {
  // on each loop, it will perform this action on
  // the item that it's currently looping through
  console.log("A " + dog + " is the cutest type of dog!")
})
}

myDogFunction(dogs)
// don't forget to call your function!
```

Go ahead and run this code. If it runs properly, you should see this in your console:

```
► (3) ["German Shepherd", "Border Collie", "Labrador"]
A German Shepherd is the cutest type of dog!
A Border Collie is the cutest type of dog!
A Labrador is the cutest type of dog!
```

Congratulations. You've just written a fun JavaScript function.

Here's a link to the fiddle: <https://jsfiddle.net/randallkanna/np3Lyh0s/2/>



ASSIGNMENT:

Play around a little more with JavaScript. Go to <https://www.codecademy.com/> and create a free account to try out the web development track.

Stretch Assignment: Select a certificate to start at <https://www.freecodecamp.org/learn>. This is the most amazing free resource out there. It will walk you step by step to creating your first projects. Some of the certificates are over 300 hours of free information.



Git + GitHub

Git is a version control system. You can think of it like Google Docs in a way. You'll be able to work with many different people at the same time with a system that won't get rid of any changes.

Every Engineer needs a GitHub account so let's go ahead and create one! Later on when you have a portfolio, you'll push up code to GitHub so other people can take a look at it. Sometimes an engineering interview will ask you to build a small take-home project and push it up to Github where they'll pull it down and run it.

Setting up GitHub

1. Go to <https://github.com/> and create a GitHub account.
2. Download Git - <https://git-scm.com/downloads>
3. Set your username in Git.
<https://help.github.com/en/articles/setting-your-username-in-git#setting-your-git-username-for-every-repository-on-your-computer>
4. When you pull down a repository, you'll need to authenticate it. Walk through the steps here:
<https://help.github.com/en/articles/caching-your-github-password-in-git> to use a keychain helper so you don't have to enter your password each time.
5. Follow the steps here to create a new repo: https://kbroman.org/github_tutorial/pages/init.html

Great job. You should be good to go with GitHub now!





ASSIGNMENT:

Push up some code to Github to start. This could be something really simple!

Resources

<https://rogerdudler.github.io/git-guide/>

<https://help.github.com/en/articles/set-up-git>

<https://git-scm.com/book/en/v2>

<http://gitimmersion.com/>

<https://vimeo.com/16395537>

<http://ftp.newartisans.com/pub/git.from.bottom.up.pdf>



HTML and CSS

Everything that you see on the Frontend of a website is HTML and CSS.

HTML is a markup language that we used to create the Frontend of a web application. For instance, a Button is created with the following HTML.

```
<button>Submit</button>
```

Here's another text element you might recognize. We use elements like, headers, paragraphs and links to build websites and show content to our users.

```
<h1>Header</h1>
<h2>A smaller header </h2>
<a href="link">Click me</a>
<p>I'm a paragraph!</p>
```

We can even include an image on our application or website by using the image tag.

```
<image src="cutepup.png">
```

Let's start by building out a simple HTML page so you can see some of these concepts in the wild.

Open up a new JSFiddle and add the following code

```
<H1>About Me</H1>
<p>I'm an aspiring software engineer!</p>
<h2>My favorite food!</h2>
<image width="200px"
src="https://www.simplyrecipes.com/wp-content/uploads/2019/09/easy-pepperoni-pi-
zza-lead-4.jpg">
```



You can replace the image in strings above with a picture of your favorite food.

Make sure you paste the code into the HTML section of JSFiddle. Now, run your code and see what happens!

About Me

I'm an aspiring software engineer!

My favorite food!



If your code ran, you should now see your image and text in JSFiddle. Here's a link to my JSFiddle:

<https://jsfiddle.net/randallkanna/jqy2vbze/3/>

Exercise: Add some text of your own to the page and maybe another link or image.



CSS

Imagine you are building a house. If HTML is the foundation of the house, CSS is the paint and wallpaper of the house. We use CSS to make our websites and applications look good. Cascading Style Sheets (CSS) can help you to add color, fonts, transitions, positioning and more.

Luckily, we don't have to write CSS from scratch. A lot of online solutions exist already. Most companies will use some type of UI framework. A UI framework makes building out webpages much easier because it provides you with things like popovers, buttons, sliders, nav bars and more. Some examples of frameworks are Material UI, Foundation or Bootstrap.

Let's try out some CSS. Let's open up our About Me JSFiddle again and add some CSS so we can see it in action.

First, let's add a class to our HTML code. A class is used to specify which HTML element that we're referring to. We'll use that in a minute in our CSS to apply the styles we want to add.

Remember the code that we used in our about me section before? Let's open that JSFiddle again.

```
1 <H1 class="header">About Me</H1>
2 <p>I'm an aspiring software engineer!</p>
3 <h2>My favorite food!</h2>
4 
5
```

Next, in the CSS tab in JSFiddle, add this code. Now if you press the Run button again, you should see the header has changed colors.

```
1 .header {
2   color: blue;
3 }
```

Google more about CSS and try to create a cool about me page on your own. Post your JSFiddle and a screenshot of your new about me page on Twitter and tag me [@randallkanna](#).



Databases

A database is an organized collection of data. You can save information to a database, query a database, update the information in your database and more. Whenever you see information like your email address, name, or phone number on a website, it's because the engineers have stored that information in the database and they're displaying it back to you.

SQL is a programming language used to query, get and set information in a database. SQL stands for structured programming language.

You can think of SQL like a language but for data. We use languages like SQL to retrieve data from our databases so we can use the data in our websites.

Codecademy has a great free resource for learning SQL:

<https://www.codecademy.com/learn/learn-sql>. You'll learn how to access, manipulate, and set data, build tables, and create aggregate functions.



Staying Motivated

We're almost through the program. Let's discuss a few strategies to keep you from burning out.

Becoming a Software Engineer isn't easy. It's hard to stay motivated every day when you get stuck or feel frustrated. One of the hardest things when you start to learn how to code is staying accountable every day. Learning to code is most effectively done in small chunks every day over a period of time. If you're just spending a few hours every Saturday on learning to code, it will take much longer than if you spend an hour or half hour every day coding. When you can set a daily goal for how long you should code, you'll notice a huge change in the amount of information you retain and how fast you can learn.

Learning to code is the easy part. It's just the everyday dedication and focus and grit that's hard! If you stick with it and work hard, you can become an Engineer.

1. Study more frequently in smaller chunks. It's better to study every day for 30 minutes to an hour instead of studying for large chunks occasionally. This isn't an easy commitment to make. You'll need to study for 6-12 months to even get an entry level job.
2. Focus on the small wins. I cannot tell you how satisfying it will be after you've worked on a problem for a few hours and you finally find the solution. This is probably one of the best things about becoming an engineer.
3. Take frequent breaks. I've been an Engineer for five years and worked at companies such as Eventbrite and Pandora. But... I can still become very frustrated when my code doesn't work! All engineers do. Instead of sitting at your computer for hours, take a break. Go for a walk! Whenever I've been stuck on a problem, the solution has come to me so much quicker when I took a short break.



4. Reach out to others for help on Stack Overflow. You don't have to go it alone on your journey to becoming a software engineer. We'll discuss networking, and finding a mentor later on but many online and free resources like Stack Overflow or an engineering slack group in the programming language you use can help you solve your issues.

Resources

<https://www.amazon.com/Atomic-Habits-Proven-Build-Break/dp/0735211299>



Find a Mentor

Having a mentor is invaluable to your career. A mentor isn't for showing you how to solve every problem but for helping direct your career and helping you find opportunities.

The times in my career that I grew the most was when I had a fantastic mentor. But there are times when you'll also need to push yourself to solve things alone because you'll learn the most that way.

Ideally, your first engineering job will provide you with a great engineering mentor. When you're interviewing for Junior Developer positions, you should ask what type of mentorship and pairing they have in place to onboard new engineers.



ASSIGNMENT:

Reach out to your network. Do you know anyone that is an engineer that would be happy to talk to you once a week about your journey? If not, can you reach out to your network and find a friend of a friend? If you can't find someone to keep you on track with your goals and help you when you get stuck, can you get a programming buddy? Do you know anyone that you can reach out to that also wants to become an Engineer that could become your accountability partner? Twitter is also a great place to potentially post about your progress every day and have the world keep you accountable!

Tip: If you have a particular interest in a language or framework, look out for engineering slack groups that focus on that language or framework.



Computer Science Fundamentals

So many people think you need to have a CS degree to become an engineer. But you really don't. I don't have one. And I know many smart and talented engineers (some of the smartest I've ever known) that don't have a degree. Some didn't even graduate college at all.

If you don't have the time or resources to spend on an expensive Computer Science degree, you might consider some great and free online resources instead. You might also consider asking your first company to pay for you to attend classes somewhere like <https://bradfieldcs.com/>.



ASSIGNMENT:

Pick one of the books suggested from <https://teachyourselfcs.com/> and start working through it. This is the best list you can find online for CS resources.

Stretch assignment: *If you're really ambitious, you might want to consider spending some time doing an online CS degree. HarvardX has some great programs available. If you do go that route, I suggest you find an employer that is willing to pay for it or try out a few edx courses (<https://www.edx.org/course/cs50s-introduction-to-computer-science>) first to make sure you're ready to commit the time.*



Picking a Specialty

There are many different types of engineering that you can try out. If you attend a Bootcamp, they'll probably have different sections throughout the course that will let you try out a little of each.

My advice? Try everything first. Talk to Engineers and ask them why they chose their specialty. Take a few inexpensive online courses before you commit to one specialty at your first job.

Frontend

If you like creating what a user sees, working with product and designers, and building React components, you might want to be a Frontend Engineer. A Frontend Engineer will have a variety of duties. A frontend engineer will take a feature on the UI and build it out, connecting it to the backend and making requests. They'll do anything from creating an entirely new page on an application, to writing an entire component that can be reused throughout the application. After the feature is done, the frontend engineer will add tests.

Resources

The best front-end courses you can take right now are available on Udemy for \$12. Stephen Grider's courses are some of the best courses out there.

<https://www.udemy.com/react-redux/#instructor-1>

<https://www.udemy.com/javascript-es6-tutorial/>

Backend

You might want to become a Backend Engineer if you like to work with data or write API's. If you're a backend engineer, you'll work on a ton of different things every day. You might build out an API that the Frontend will consume to provide data from the database to our users. You might be writing a lot of different tests so you can run them and verify your app is still working properly. Sometimes you'll be deploying code and helping out with potential security flaws.



When you're first starting out, you should try both frontend and backend and see what you like more. Ideally, you could find an apprenticeship or internship that lets you learn a little of both. If you attend a bootcamp, you'll be taught both so you'll get the chance to learn a little about both.

Resources

Free CodeCamp offers a fantastic intro to backend certification. It's about 300 hours and it will give you some basic insight into the work of a Backend Software Engineer.

<https://learn.freecodecamp.org/apis-and-microservices/managing-packages-with-npm/>

<https://www.udemy.com/node-with-react-fullstack-web-development/>

Mobile

Mobile used to have a much higher barrier to entry. But thanks to React Native, it's a lot easier to pick up mobile development. I've worked as a Swift Engineer and it's also much easier to pick up than Objective C.

<https://www.udemy.com/the-complete-react-native-and-redux-course/>

Machine Learning

A great course about Machine Learning

<https://www.udemy.com/course/complete-machine-learning-and-data-science-zero-to-mastery/>

Blockchain

B9 has a great free introduction course to start to understand the basics of Blockchain and ethereum development. <https://academy.b9lab.com/courses/B9lab/X16-0/2016/about>

My Book: I also wrote a book that will provide you with the same amount of resources as a \$1000 course. You'll be able to walk through building a few smart contracts and a front end to interact with them.



Google is Your Best Friend - How to Find the Answer for Everything

When I first started out as an Engineer, I thought that I had to know everything. I assumed that a 'real' engineer didn't have to look things up and they just knew the answer immediately. I quickly learned how wrong I was. I paired with a senior engineer and he kept stackoverflow open the entire time. I assumed that this was like cheating in the engineering world before I got my first engineering job. I felt so worried that I would start my job and I wouldn't do well because I wouldn't be able to memorize everything and just know the answer.

Engineering isn't about knowing all the answers. It's about knowing how to find them. Here's a great blog post about all the things an engineer googled during the week!

<https://localghost.dev/2019/09/everything-i-googled-in-a-week-as-a-professional-software-engineer/>

In some interviews (at the right company), they'll want to see how you work. And asking the right questions and having the ability to figure out things quickly is really important. Anytime when I've interviewed someone, I've made it very clear that they can google during the interview.

The best thing you can do is just start practicing googling. Anytime that you get stuck, google it. Google is going to be your new best friend.



Building a Portfolio: How to Accelerate Your Coding Abilities Fast

You can't learn to code by only watching tutorials or pair programming or attending a bootcamp. Even when I did attend a bootcamp, I had to teach myself at night and on weekends. There is no magical solution to becoming an engineer. While the bootcamp did give me the confidence to apply to jobs and a structured path to learning, I had to teach myself the framework my first company used to get the job and stay employed.

Ultimately, the best way to learn how to code is to just build a project you're interested in. A tutorial is great when you're just starting out, but the only way to truly learn the concepts is to write the code yourself. Feel free to follow tutorials and video tutorials but try early on to just start building small projects.



ASSIGNMENT:

Your task is to start with a small project. It might not be a full application (and it probably won't be!) but try to build something. Anything. Think of a fun idea that you'll feel passionate enough to keep working on even when you get stuck and frustrated. Don't feel discouraged if it takes longer than you think or you can't think of a good idea. If you can't think of something, look at examples of what other people have built and build your own version. If you just get started, you'll improve at coding.



What About a Bootcamp?

There isn't just one way to become an Engineer. These days, someone can attend a bootcamp, teach themselves, get a degree, get an internship, etc. There are so many paths open to anyone willing to work hard.

I attended a bootcamp but I still had to teach myself during and after and just figure things out. My first company hired me as an apprentice on a trial basis and after I had proven myself, they hired me as a fulltime engineer. My sister offered to work for free at a company for the first few weeks just to prove she could do the work. If you didn't get a CS degree, no worries. There are many other paths to becoming an Engineer.

Bootcamps

A legitimate bootcamp is a great investment in your career. When I graduated college, I felt a little lost on what I wanted to do next. I had always loved coding but had never really pursued it. My aunt knew that I was trying to figure out what I should do for my first real career steps besides internships and part time jobs in college and sent me an email about coding bootcamps. This absolutely saved me. I had never even heard about a coding bootcamp before. I immediately started doing a ton of research. It just seemed too good to be true. I read blogs posts from every single student that had went to the bootcamp that I could find online. I read any review I could find. I started emailing students that attended the bootcamp and begged them to answer a few questions.

After I did a ton of research, I decided to apply to Dev Bootcamp. I was SO nervous when I got in. I didn't even have the money at the time so I had to take a loan from my parents. I couldn't even find a reasonably priced apartment in San Francisco so I was sleeping on a bunk bed with a roommate in a crowded house that had the vibe of the Weasley's house but without the magic.

It ended up being the best thing I had ever done.



Fast forward five years, I'm a Senior Software Engineer. I speak at conferences all the time, I've worked at big public companies such as Eventbrite and Pandora. I've been interviewed for newspapers and on television several times. I've been paid to consult at cutting edge companies, and I'm a published author with the biggest Engineering publisher in the world. I would say a Bootcamp worked out for me.

But it didn't work out for everyone that attended with me. I started out with 50-60 students in my cohort. By the time we graduated, about 10 graduated with us. Some had stayed behind a cohort to learn a little more. Some dropped out early on while they still got a significant refund. But others had decided that engineering wasn't for them partway through the program when the financial loss was high. Others were asked to leave because they couldn't keep up.

A coding bootcamp is one of the largest purchases you'll make in your life. Do your research. Many horror stories exist online of people who paid \$10-20k only to find out the bootcamp was a scam or the teachers were inadequate.

A good bootcamp will start with the first phase online where you'll learn from your home. It'll teach you the basic concepts of programming so that when you get to the onsite portion of the program, you can focus on tougher engineering concepts with teachers around to ask questions.

Make sure that you find a program that also has a phase solely focused on interviewing prep and supplies you with a career team. This was one of the most beneficial parts of my bootcamp. I got my first engineering job because the bootcamp helped me improve my LinkedIn and my company found me through the LinkedIn search.

If you do decide to research coding bootcamps, I recommend you start with the following: HackReactor, App Academy, and Hackbright.



Self learning

If you have the time and can manage your own time well, this option might be a great fit. It's definitely the toughest choice because you will need to keep to a schedule and stay very motivated.

Apprenticeships/Internships

My sister taught herself and then told a company she liked that she would work for free for a few weeks to prove herself. They liked her determination in reaching out and they ended up hiring her as an engineer full time. Most apprenticeships and internships are paid and offer mentorship/support.

You'll most likely need to look for programs from larger companies because startups generally don't have the resources to do these kinds of programs yet. A larger company will also have a more formalized process, which will mean you'll most likely have a dedicated mentor, time to onboard, more resources, etc.

However, while these are internships/apprenticeships, generally the people that do these programs have already graduated with a CS degree or attended a bootcamp or generally are able to hit the ground running. The interview process for these openings are still competitive.

Transitioning to Engineering at your current company

At one of my previous companies, many individuals were able to transfer from different careers to engineering. Some worked in customer support or in QA. It wasn't easy and they still had to teach themselves a lot, but they are now all working full time as engineers. This requires that you are able to get your current work done and have the ability to take on extra work in your own time to learn.



Job search

Set a specific goal to start out your career. Do you want to be an engineer at a big tech company? Do you want to freelance? Do you want to work at a startup?

Often these goals will change later on. But this will help you to narrow your search early on. If you have a family and need to have normal working hours, a small startup might not be for you. If you want to learn fast and have the opportunity to take on a lot of work, a startup might be a great fit.

A large company that has the resources to mentor and onboard you to the company is NEVER a bad choice for your first job though.

Acing the Interview

Engineering interviews are not an easy feat. Even as a senior engineer, they can be painful and trying.

The best thing to do when preparing for a technical interview is to treat your job search as your job. Set a schedule that you'll commit to every day. Treat it like a 9-5. After 5PM, you're off for the day. This will keep you from responding to recruiter emails late at night.

Resources

Pramp (<https://www.pramp.com/#/>) offers free interview practice from actual engineers. However, you will need to be a little more experienced in engineering before you can take advantage of this resource because it requires that you interview the engineer in return.



Cracking the Coding Interview

(<https://www.amazon.com/dp/0984782850/?fbclid=IwAR3aXb7tnSOI7ohBU0YhRrdDjwxEML1lpC1uPnUJwbFAD9uigQiTdVF2Qcl>) is a great resource for preparing for a technical interview. Not all companies will ask you to perform an outdated whiteboarding interview, but this will help you prepare for the ones that still do.

For algorithm questions, there are a ton of online and free resources where you can practice solving these types of questions. Here are a few: <https://www.codewars.com/>, <https://coderbyte.com/>, <https://leetcode.com/>, <https://www.hackerrank.com> and <https://projecteuler.net/>.



Networking

Networking is a great way to find your first job. I know many Junior Software Engineers who hustled their asses off and were able to help get some first opportunities by just attending meetups.

Meetups

Meetups are a great way to start learning and introducing yourself to the community. You can use meetup.com to find meetups near you to attend. Don't be shy about attending early on in your dev journey. Everyone started out right where you are. And most people are happy to talk about their skills and how they got there.

Conferences

Conferences are a great place to look for onsite tutorials taught by some really fantastic engineers. Generally, the day or two before a conference, they'll have an entire day devoted to tutorials and workshops. Sometimes you can score free passes to conferences as well.

Slack groups

Slack is an application that most companies use to stay connected.



ASSIGNMENT:

Go to a meetup that's based on the technology you want to learn. Generally React and Ember.js have some great and open meetup groups all over.



Starting a Technical Blog

One of the best ways you can get your name out there when you start looking for a job is to have a technical blog. Prospective employers might stumble across it and companies that you're currently interviewing at will look at it to find out a little more information about you. It will also show potential employers that you're passionate about coding.

Sometimes the best perspective when you're learning to code is from someone who just learned what you're trying to learn. Senior Engineers can't always remember how difficult it was to learn a basic programming concept because it isn't something they've had to do in a long time. I remember when I just started out and I was in a bootcamp, I would read the blog posts that other people in my cohort had written and learn so much from it. When you're just starting out, sometimes even figuring out how to run a file in the terminal is difficult.

Writing a blog post and teaching others can solidify your own learning. You'll learn much faster if you have to deep dive into a concept and think about how you would teach this to someone else.



ASSIGNMENT:

Select a simple programming topic that you want to teach to someone else. Don't think too much about the topic itself. Just start writing. Often, we try to think of the 'perfect' topic for so long that we could have written three blog posts in the same amount of time.

Remember that everyone had to start somewhere when writing a technical blog. My first few technical blog posts? I've since deleted them because I'm so embarrassed by what I had written in the past! But... They helped me make new connections, get my name out there, and even find a job.



Day One of a New Career?

What do you think? Are you still interested in becoming a software engineer?

Anyone can be an engineer. There isn't one way to become an engineer or a single type of person that can become one. If you have the determination to keep learning every day, you can become an engineer. You don't need a degree. You just need to keep learning and improving your skills.

I've given you a little more direction on where to start your journey as a Software Engineer. Connect with me on Twitter because I'll be releasing more material and courses soon.

<https://twitter.com/RandallKanna>

Here are a few of my favorite resources to continue your learning

Codecademy - <https://www.codecademy.com/>

A great basic introduction to data structures and Algorithms by Stephen Grider. Available on Udemy at a low price. <https://www.udemy.com/coding-interview-bootcamp-algorithms-and-data-structure/>

A great list of free resources if you want to work on the fundamentals of computer science. I have some of the books on the list and they're fantastic. <https://teachyourselfcs.com/>

The best free resource ever: freecodecamp.org/learn

