Table of Contents

Prepare to Code! - Pre-Work Curriculum

"Give me six hours to chop down a tree, and I will spend the first four sharpening the axe." - Abraham Lincoln

---

Welcome!

Congratulations on your acceptance to the Full Stack Flex Program!

You are about to embark on an intense and transformational journey that will dramatically change your life. At times, it will feel challenging, frustrating, and utterly bewildering, but trust us; it *will* be worth it in the end.

Be warned! This program's curriculum is not easy. You will be exposed to new ideas at breakneck speeds and will be challenged to complete difficult exercises. Be prepared to push yourself and to be pushed.

That said, remember every one of us, behind the scenes, is rooting for you. You were accepted into this program because we believe in your potential to succeed. You will have excellent instructors, dedicated TAs, an extremely rich curriculum, and supportive peers to nudge you forward. If you put in the long hours and hard effort, we're confident you will find the success you're looking for.

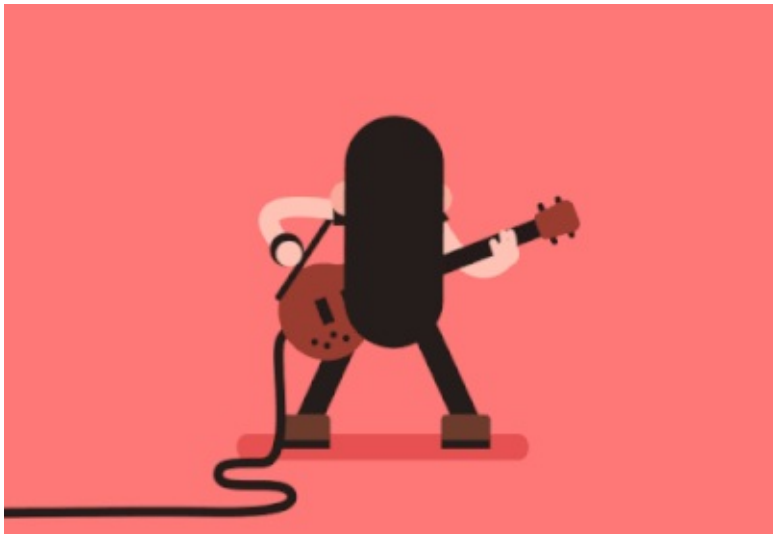
The Importance of Pre-work

Here's the thing; those long hours and hard efforts should begin now. Because of the challenging nature of this program, we want you to come ready to ***sprint*** on Day 1. To help prepare you, we've created a series of modules to complete before your first day. These modules include a mix of reading assignments, videos, and exercises meant to prepare and challenge you for the content ahead.

It's recommended that you read the chapters sequentially, but you can complete the assignments in any order. Just remember, you **must** complete Modules 1-11 before the first day of class.

Good luck! Have fun! Get pumped!

You are in for an amazing ride.



Pre-Work Modules

Module #1 - Prepare for Pre-Work! (Required)

Prepare to be prepared! Let's have a pre-work conversation about pre-work in this module.

Module #2 - So You Wanna Be a Web Developer (Required)

Web Developer—what a funny phrase. Get answers to your most basic questions about the role in this module.

Module #3 - Get Your Mind Right (Required)

This boot camp won't be easy. Set the right attitude and expectations through the help of this module.

Module #4 - Type Like a Tiger (Required)

Every line of code begins on the keyboard, and so should you! Practice your typing skills with this module.

Module #5 - My Machine is Ready! (Required)

Ready your computer for bootcamp battle! Tools, installers, and readmes abound in this module.

Module #6 - Sort of Code with Pseudocode (Required)

Code is complicated, but life doesn't have to be. Pseudocode your way to clarity with this module.

Module #7 - Where's the $@#*'ing Error? (Required)

You'll find yourself confused many a time in this course. Prepare your strategy in this module.

Module #8 - HTML Hotness! (Required)

Time to finally code! Get a head start on the first day of class through this module.

Module #9 - See Me CSS (Required)

I'm so pretty! Style up your simple HTML with CSS, and further your head start in the course with this module.

Module #10 - Jiggle into JavaScript (Required)

Dang! You're diving into the good stuff! Get a preliminary dose of JavaScript (Week 3) in this module.

Module #11 - Time to get Employable (Required)

Prep your online persona and explore your professional side in this final module.

Module #12 - What Next? (Recommended)

All done with the required modules, and looking for more practice? Check out our recommendations in this module.

Module #13 - Whiteboarding Interview Prep (Recommended)

Think you're ready to prepare for your technical interview? Head here!

Copyright

Module #1 - Prepare for Pre-Work!

But... My Vacation!

We know what you're thinking - *"Thanks for ruining my break, man."*

But think of it this way - completing this pre-work will be your second big step towards your desired career (the first was signing up for the bootcamp). As you approach the intensive bootcamp, it goes without saying - that you are in for a long and challenging road ahead. This pre-work curriculum will provide you with preliminary skills and perspectives necessary for you to succeed in (or... survive?) the first few weeks.

Each of the modules comes with a required chapter of reading, an associated assignment, and optional content. You can complete the assignments in any order, but we suggest reading the chapters sequentially. Prior to class, you must complete **all of the required assignments (modules 1 - 11).**

In some cases, the pre-work assignments will be fairly challenging and require significant research and self-learning. Overall, you should expect to spend between **15 - 40 hours total** completing this pre-work.

The time and effort required to complete this pre-work will give you a taste of the challenge to come. Just remember that perfection isn't everything. Give each of these modules your all and submit whatever you create. You'll have six months (three months if you're in a full-time program) to smooth out the edges.

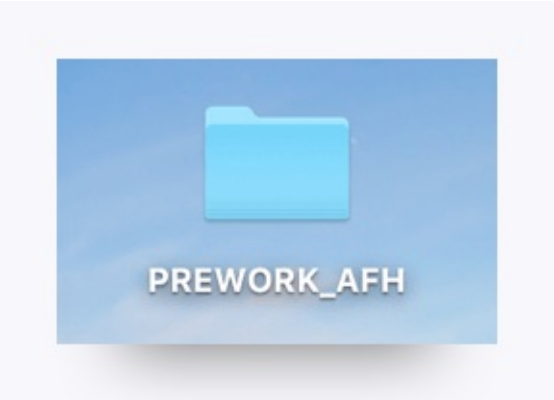
Good luck!

How to Submit Pre-Work

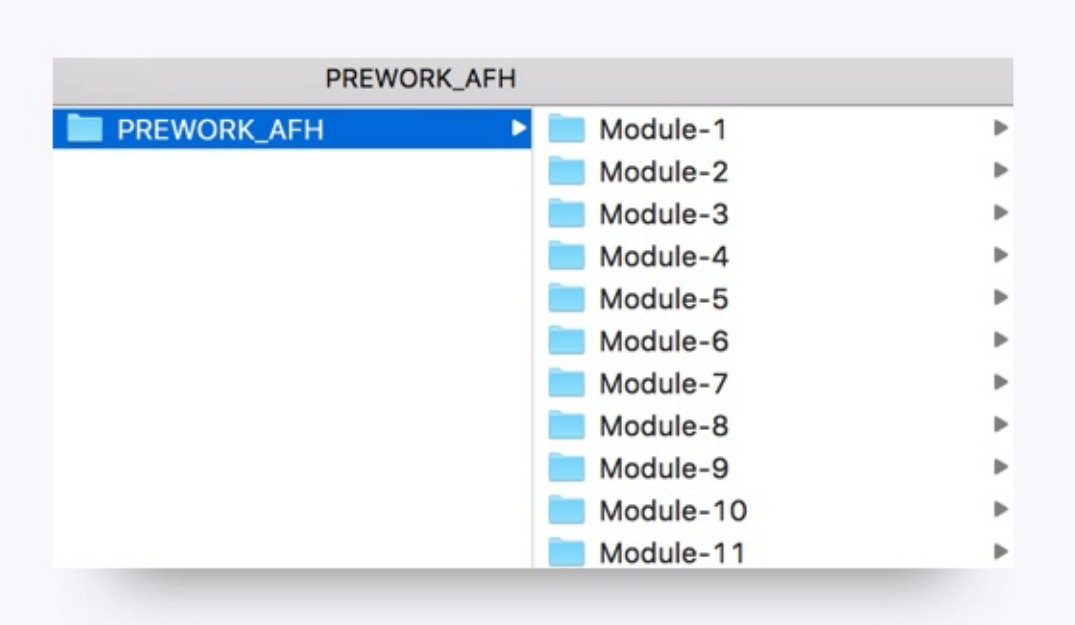
Ultimately, you will be submitting each of the completed assignments in a single folder onto Bootcampspot-v2.com. You will be given access to the site by your Student Success Manager 7 days prior to the first day of class.

Once you gain access, you can use the below instructions as a final submission guide.

1. Create a local folder on your computer titled: PREWORK_{INITIALS} (with your initials replacing the stuff in brackets, e.g. "PREWORK_AFH").



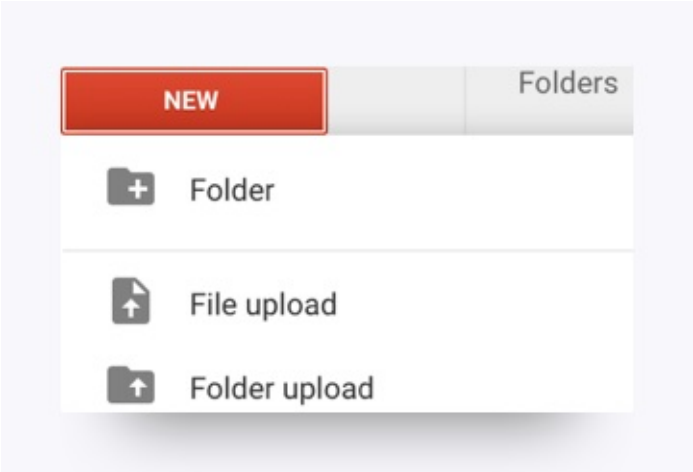
2. Inside of the folder create sub-folders for each of the assignments. Your folder should look something like the below.



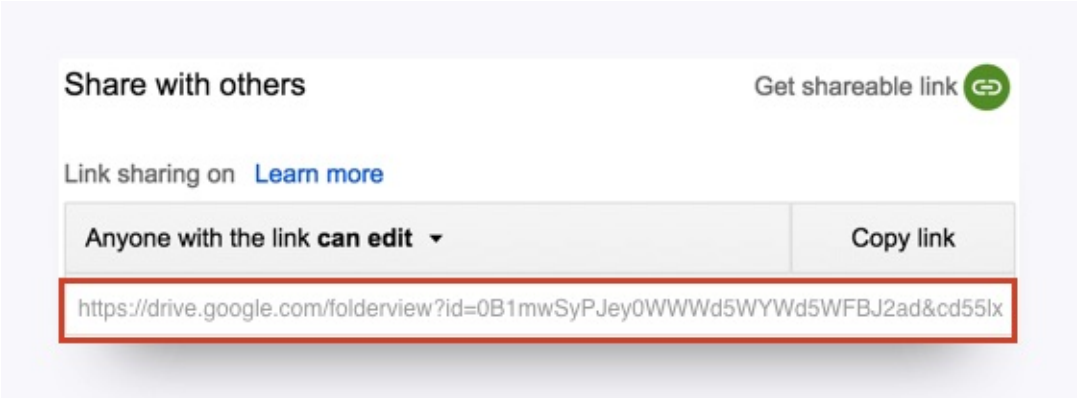
3. Upon completing each assignment, save your solution into the associated sub-folder of your pre-work folder. In some cases, this will mean copying multiple files into the sub-folder.

4. Then create an account on Google Drive if you do not already have one.

5. Once you are done with all of the pre-work, utilize the "Folder Upload" button on Google Drive to upload the folder onto Google Drive.



6. Then right click the folder on Google Drive and change the sharing settings such that anyone with the link can edit.



7. Copy the link associated with your folder in Google Drive.
8. Then log in to Bootcampspot and find the Pre-Work Assignment associated with your class.
9. Click on the assignment and paste the Google Drive link so your Instructor and TA can review your assignment.

And that's it. Now get cracking on those modules!

Assignment:

- None!

Supplemental Resources:

- Google Drive

Copyright

Module #2 - So You Wanna Be A Developer... (Required)

Well, you've come to the right place! Over the course of the next six months (three if you're full-time) you will be undertaking rigorous training to develop the skills necessary to become a web developer.

But you may be wondering - *"What exactly is a web developer?"*

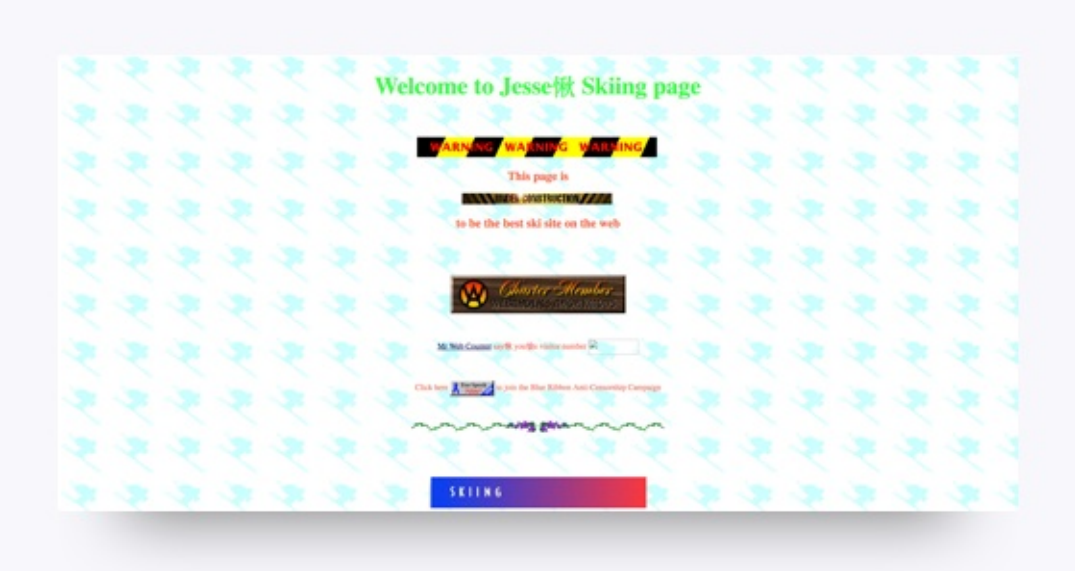
Great question. For many of us, the idea of a *web developer* is one that evokes vague notions of sketching designs, creating websites, and wearing hipster clothing. But the day-to-day work of web development remains a mystery. To get a grasp on a better answer, it's worth taking a step back to understand how the Internet works.

How the Internet Works - in 50 seconds...

Most of us take it for granted that when we click a link online - *something* happens. Maybe a chat window pops up... maybe a video loads... or maybe we end up on a new page completely.

This *something* happens seamlessly, despite the Internet being composed of hundreds of thousands of loosely connected networks of machines and software. In fact, each time you visit a webpage - you are actually *communicating* with one of these networks. The network then recognizes your *request* and, in turn, *responds* with a viewable webpage.

In the old days, this was the end of the story. Viewable webpages were simple documents composed of a little code describing how they should look and act, and because the pages were *static*, the transmission was a simple one-way street. A great example of a *static* website would be a simple Geocities page:



But as time went on, the demands and expectations of users increased. Users were hungry for websites that felt live, responsive, and richly interactive. Because of this, new technologies advanced such that more complex code could be embedded into a webpage's makeup.

Instead of a network being restricted to sending a static webpage, it became possible to send dynamic websites capable of doing much more. For example, the modern website - or *web application*, as they are called - is capable of doing any of the following:

- Displaying real-time data to users
- Responding to multiple user interface elements simultaneously
- Synchronizing visual elements between users across locations
- Live streaming videos
- Maintaining encrypted channels for private information
- Algorithmically predicting user preferences based on historic choices
- And much more!

Because of this added complexity, creating these features require a broad expertise in various technologies and programming methods. As experts in the field, *web developers* play the essential role of being the architects behind users' web experiences.

So, is it sort of like... Graphic Design?

Unfortunately, not really. The tools used to architect the web are far from the point in which developers can simply *drag and drop* visual components onto the screen. Instead, developers use specialized languages like HTML, CSS, and JavaScript to lay out their designs and functional features in code. This makes the process more challenging for beginners, but at the same time, this is the reason every browser on the web can view webpages in the same way.

Sounds technical... Do I need to have a Computer Science background?

Absolutely not! In fact, according to studies, nearly 50% of all professional developers are self-taught. Think about this for a moment. Today, there are few skills that you can learn that present the breadth of job opportunities, the quality of job opportunities, and the creative freedom that come with web development. Yet, unlike most industries, the field isn't restricted to those with a formal degree. It's a powerful opportunity for those looking to make a career change or to reinvent themselves.

If I don't need a degree... how do I prove I'm actually a developer?

Another great question! If you remain hardworking, diligent, and focused on learning, then you'll be able to prove your expertise by having a portfolio of works. For many of you, this will be a liberating experience. Instead of being judged solely on the basis of your past credentials or work experiences, your actual *works* will become a gateway to your future career.

For this reason, throughout the program, you will be constantly tasked with building web applications for your homework assignments and projects. Take these seriously! They will often feel very challenging - but swing with the punches and try your hardest on each assignment. If you stay consistent and remain confident, your work will *gradually* become better and give you the proof you need to demonstrate your skills in the professional arena.

I'm still a bit fuzzy.

That's understandable. Take a stab at the next assignment and get one step closer to clarity.

Assignment (Required):

- My Future Job... I think?

Additional Reading / Viewing:

- What Does a Web Developer Do?
- How Do I Compete as a New Web Developer?

Copyright

Assignment #2: My Future Job... I Think?

Overview:

This assignment is a quick research task to help you understand the web development landscape.

Instructions:

1. Read through the "So You Wanna Be A Web Developer..." chapter of Pre-Work.
2. Then, visit a job site like Indeed or Dice.
3. Search for any of the following job positions:
    - Web Developer
    - Frontend Web Developer
    - Backend Developer
    - JavaScript Developer
    - Node Developer
4. Read the job postings for at least 10 of the listings. As you are reading, take note of at least one technology, tool, or term, with which you are unfamiliar, from each listing (e.g., *Angular, API*). See the example below:

5. Then, search online for a definition or description of that technology or term. Try to understand what role it plays in the context of how web applications were described in that chapter.

6. Open a Word or Text file and paste in your description for each of the technologies you researched. You should have a total of 10 descriptions listed.

Note:

- Don't overthink this. We just want you to begin contextualizing the technologies you'll be learning about in class and beyond.

Copyright

Module #3 - Get Your Mind Right (Required)

As every successful person will tell you, "Attitude is everything."

For this reason, as you embark on the challenging endeavor ahead, it's critical that you align your attitudes and expectations with the realities to come. Don't underestimate the amount of mental and emotional discipline this bootcamp will require. Take the time to prepare yourself by reading through the following tips and suggestions.

P.S. Don't breeze through these! They are *important*!

Major Keys to Success

1. A Little Humility Never Hurt Nobody...

Whether you believe it or not, this bootcamp will be one of the most challenging learning experiences of your life. For most of you, the skills being taught in the program will be fundamentally different from any you've been exposed to before. In fact, your past skills, schooling, and experiences are unlikely to help you become a successful developer. Instead, in order to succeed, you will need to channel your inner toddler and *learn to learn* again.

This will be a *very* challenging identity for many of you to take on. If you've had past academic or career successes in another trade or industry, you may find yourself resisting the idea that you can *fail* so dramatically at coding.

Yet, heed this advice now: "A humble learner is the readiest learner." Take each class as a new opportunity to be proven a fool and learn anew. Six months (three if you're in a full-time program) from now, you will marvel at the mastery you gained.

2. There is No Coding Pill...

The greatest self-lie you can tell is that any program will *teach* you to code. Instead, it's important to realize that, in order to succeed, *YOU* must own your learning experience.

Coding, like any craft, requires diligent effort, constant experimentation, and a relentless desire to self-improve. These characteristics cannot be taught in any classroom. From Day 1 through Day 180 and beyond, remind yourself that YOU are responsible for your ultimate success. We will be here to guide, to encourage, to facilitate, to point out your mistakes, and to show you the path, but it's up to you to put in the effort.

Don't get discouraged! We know you can do it.

3. Put in the Hard Hours (At Least 20 Hours!) ...

According to author Malcolm Gladwell, successful people put in a minimum of 10,000 hours of deliberate effort to master their craft. As newcomers to the field of web development, be ready to put in your share of hours.

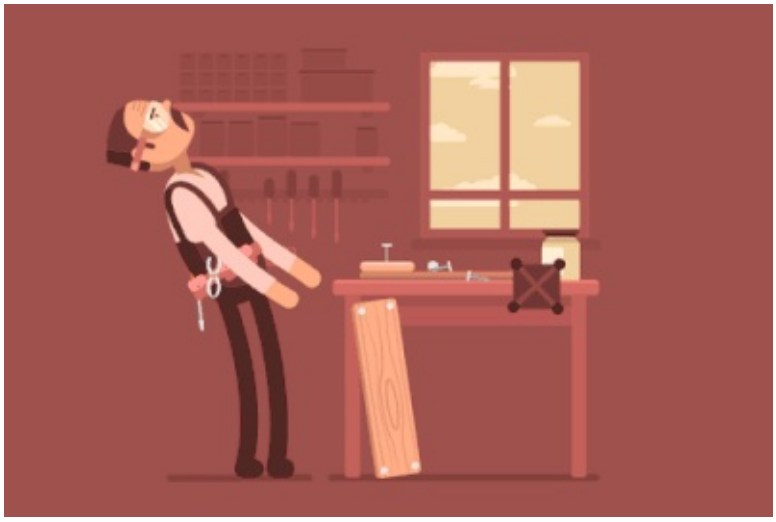
While the bare minimum to *survive* this program is 10 hours of outside class time, we've consistently found that those most successful put in closer to 20 hours of outside effort per week. At times, this number might even need to go upwards of 30 or 40 hours per week during more challenging topics.

Simply said, there is no substitute for long, hard hours.

Consider this time to be an investment in yourself, and know that for every hour you spend, you are guaranteeing yourself a better opportunity after graduation.

4. Patience Makes Perfect...

Tied to the previous suggestion is a second piece, often forgotten. Not only will learning to code require many hours, it will also require many months (and frankly years) to master. Don't rush success!



For almost all of you, the first couple months or so of the program will be particularly challenging. In fact, for many of you, this period will be one in which you doubt your eventual success through the program. Resist the urge to give in or to become hopeless.

Realize that learning this craft will require consistent effort, which will iteratively build your skills and understanding. What will seem challenging, confusing, and distressing in Week 3—will seem completely second nature by Week 24. *Intensity is no substitute for consistency.*

5. Not a Spectator Sport...

As every musician, painter, or craftsperson will tell you, one cannot learn a skill by simply reading, watching, or following along. Instead, it requires hours of deliberate practice. In the same way, coding, just as any other skill, will require you to step away from the spectator stands and to enter the fray yourself.

Know this now: going to class, watching tutorials, reading articles, or participating in other passive activities will only get you so far. You must spend significant hours actually *coding* to succeed in this bootcamp. As a benchmark, consider spending 70–80% of your outside class time writing and reviewing code. Only the remaining 20% is for other passive activities.

But I don't feel ready!

Often, we've seen that students who struggle the most are the ones most intent on *learning everything* before they start coding themselves. Unfortunately, this is a recipe for stagnation. You will not be successful in this program unless you force yourself to work through confusion: writing code, making errors, chasing them down, and learning from your mistakes.

This bootcamp is a mudslide. There's just no way to avoid getting your hands dirty.

6. Crumple the Paper Tiger...

Am I doing this right...?

The five most hated words of every development instructor.

These words aren't hated because your instructor is unwilling or is unable to help. Rather, they're hated because they suggest that a student is afraid to *try* something on their own. As you enter the classroom, learn to abolish this phrase from your vocabulary.

Instead, learn to try things on your own, to test what you can, and to do a bit of debugging first. *Then* turn to your instructor and say, "I tried such and such. It didn't work so I tried such and such. But it still didn't work. What should I try now?"

Notice the change in tone. Instead of timidly asking for assurance, give yourself permission to *just try*.

The best way to fail is on your own terms.

7. Find a Squad...

For whatever reason, Hollywood has created an image that every coder is an isolated loner, programming in a cubicle. Don't let this be you!

While in this program, make every effort to find friends, to form study groups, and to work together in and out of class. Sometimes, the fastest way to overcome a bug or to understand a challenging concept is to have another pair of eyes or to have another perspective. Pair programming is a real thing for a reason!

8. Master the Art of Google Fu...

One of the greatest surprises to students entering the program is the amount of *Googling* they are asked to do. This isn't because your teacher is lazy, or the curriculum is unplanned - far from it!

As every professional developer will tell you, coding isn't about *memorization*. Instead, it's about curating bits and pieces of knowledge, scattered across the web and accumulated in various documentations, forums, and QA websites.

To become a good developer requires an ability to work through problems and to quickly *research* solutions ascertained by past developers. The field is constantly changing, and new tools are always on the horizon. As you will find, every good developer is a great friend of Google (and Stack Overflow).

For some really cool tips on how to become better at Google Fu, check out this article!

9. Plan Often...

As you will learn in the pseudo coding module, the best first step in *every* coding challenge is to formulate a plan. Break down the complex task in front of you into discrete, bite-size coding challenges. Once you have a plan, write it out and always refer back to it.

Fundamentally, every task in code can—and should—be broken into smaller tasks. Don't try to bite off everything at once! You'll get lost in your own mind games.

10. Fixing Things Takes Time...

One of the most frustrating aspects to new students of software development is the sheer amount of time it takes to troubleshoot (or debug) issues in code. At times, it might even feel like *fixing* an issue is taking 3–4 times as long as conceiving the original solution.

Know in advance that this is completely normal.

For novices and experts alike, fixing code is often the most time-consuming task of all. Instead of seeing these spent hours as a *distraction*, learn to see them as a critical part of the learning process. Each bug you pursue is a lengthy lesson that adds to your arsenal of understanding.

11. Self-Care is Key...

While we've probably traumatized you with all the talk of challenges, of difficulties, and of effort, we do *want* you to take care of yourself. Throughout the program, be sure to sleep, to exercise, and to eat nutritional meals. These moments of self-care are extremely important for your mind to be healthy. Taking breaks is encouraged!

In fact, you will find that that some of the best problem-solving happens during breaks. "Sleeping on a problem" is often a very real solution to your most challenging coding issues. Try to walk into class each day, ready and refreshed for new learning. We want happy, energized people in our classrooms. Not dead robots.

12. Be a stellar student

As a student, we'll encourage you to collaborate with your classmates, and participate in study groups in and out of the classroom. We'll ask you to look for code examples in the curriculum and online as you're learning. This is good practice for a developer as you'll do the same on the job.

Just make sure you're doing your own work and submitting applications you can be proud of. If you feel like you are borrowing too much, reach out to your SSM for guidance.

There's value in the work you're doing, it translates to hard skills, and you truly get out what you put into this program.

13. You Can Do This!

This last piece of advice is the most important. Remind yourself each day that you *can* do this. We've seen through many classrooms, students from all backgrounds, experiences, and personal situations persevere and succeed through this program.

During the tough times, dig deep into your own personal motivation and remind yourself why you entered the program. Let this be the fuel that you use to keep on.

You have *everything* it takes to learn this craft and to gain the opportunities that come with it. It may feel challenging at times but remind yourself of past challenges you've overcome. Your future self will thank you for all that you endured.

Video (Recommended)

- Grit: The Power of Passion and Perseverance
- James Clear: Successful Habits
- Monica Szabo: Student Inspiration

Assignment (Required):

- My Greatest Challenge

Additional Reading:

- Learning Programming: How I Overcame Initial Resistance
- Learning to Program at Age 30

Copyright

Assignment #3: My Greatest Challenge

Overview:

This assignment is a quick self-reflection to ensure you've read the previous chapter.

Instructions:

1. Read through the "Get Your Mind Right" chapter of Pre-Work.
2. Then identify which one of the *12 Keys to Success* will be most challenging to you. *Really think about it!*
3. Finally, write a 75 - 150-word paragraph describing which of the challenges you think will be most difficult for you and what steps you will take to overcome these difficulties.
4. Save the reflection in a *Word* or *Text* file inside of your Pre-Work folder.

Note:

- Don't overthink this. We just want to be sure you read the chapter and put some thought into it.

Module #4 - Type Like a Tiger (Required)

Welcome back! Hope you had fun with the last module.

Now it's time to crack your knuckles and to put your fingertips to the test.

As you begin your head-first plunge into life as a developer, it's important that you walk in with a few *pre-skills*. One of these pre-skills — often forgotten, yet utterly critical — is the ability to type quickly.

No Hunting and Pecking!

Why does it matter so much in coding? Well, in coding, you'll frequently be *ideating*, *experimenting*, and *testing*. You'll often come up with an idea for a visual layout, type it out in text, and then run into an unexplainable bug. This bug might leave you scratching your head and frantically changing your code for who-knows-how-long before you finally stumble into a solution and begin the cycle again.

This iterative process is often extremely time-consuming and challenging, but it's made even worse if your fingers can't keep up with your ideas. What we've seen repeatedly in our program is that a slow typing speed is a *huge* barrier to student success. The reality is Picasso wouldn't be the painter he was if he was belaboring every brush stroke. In the same way, you can't be a successful developer unless you can type at a reasonably fast pace.

Psh... I already iz fast

Now, we know what some of you may be thinking — *I type plenty fast already. I can type out a paper in my sleep!*

And that's great Mr. Hemingway.

But coding presents a different set of typing challenges. Instead of typing words, you're often typing brackets, parenthesis and other unfamiliar symbols. These characters often trip up new developers who fumble around, trying to figure out where everything is on the keyboard.

Take, for instance, the below block of code (taken from one of your in-class activities). How long would it take you to type?

```
var wellSection = $("<div>");
wellSection.addClass('well');
wellSection.attr('id', 'articleWell-' + articleCounter)
$('#wellSection').append(wellSection);
```

In order to maximize your learning during the first month of classes, it's important that you come in having practiced both your regular and your code typing skills.

For this reason, your task in this module is to achieve the following goals:

1. A minimum *regular* typing speed of 50 words per minute
2. A minimum *code* typing speed of 30 words per minute

These aren't lofty targets, by any means, but they will be a baseline *check* to confirm that you're reasonably ready. Take as much or as little time as you need to reach these targets.

But don't skip this task! You'll thank us later.

Now get to it, Speedy!

P.S. If you ever need an ego boost, consider visiting Nitro Type. It's a fun, little site where you can race against others practicing their typing. Plenty of fifth graders are on there... so it's always fun to smoke them. (Trust us. Over the course of the next six months, you're going to need any ego boost you can get.)

Assignment (Required):

- My Typing Credentials = Elite

Additional Reading:

- We Are Typists First, Programmers Second

Supplemental Resources:

- Nitro Type - Type Race Against Fifth Graders to Boost Your Ego
- Typing.io - Typing Drills for Programmers
- Typing Test - Site to do a quick typing test
- Learn Typing - Free site to learn touch typing skills

Copyright

Assignment #4: My Typing Credentials = Elite

Overview

In this assignment, you will be completing two tests to evaluate your typing proficiency. The first will be a simple test for your normal typing speed, and the second will evaluate your *code* typing speed.

Instructions

1. Complete each of the following tests:
    - Normal Typing Test
    - JavaScript (Coding) Typing Test
2. Confirm that you meet at least the following WPM:
    - 50 WPM (Normal Typing Test)
    - 30 WPM (Coding Typing Test)
3. If your WPM is faster than the above numbers for each test, take a screenshot of your scores. Save these screenshots and include them in your Pre-work folder. Title the images whatever you like.
4. If your WPM is not faster than the above for each test, continue practicing with the resources below until it does. Once you've achieved the required score, rejoice! Then follow the instructions on Step 3 to upload your score.
    - Nitro Type - Type Race Against Fifth Graders to Boost Your Ego
    - Typing.io - Typing Drills for Programmers
    - Typing Test - Site to do a quick typing test
    - Learn Typing - Free site to learn touch typing skills

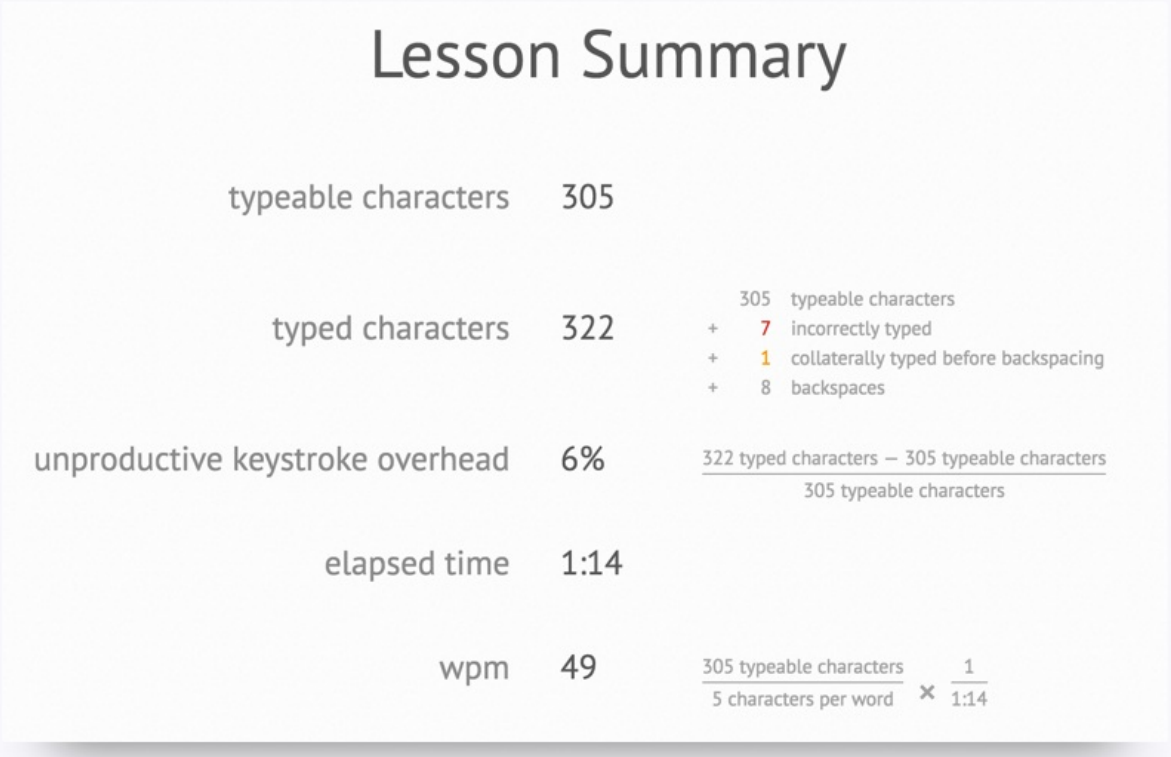
And that's all there is to this module! Good luck with the next set of modules.

Bonus!

Psst... Want a real challenge? Why not try to beat our Curriculum Director's scores?

Curriculum Director's *Normal* Speed

Curriculum Director's *Coding* Speed



Copyright

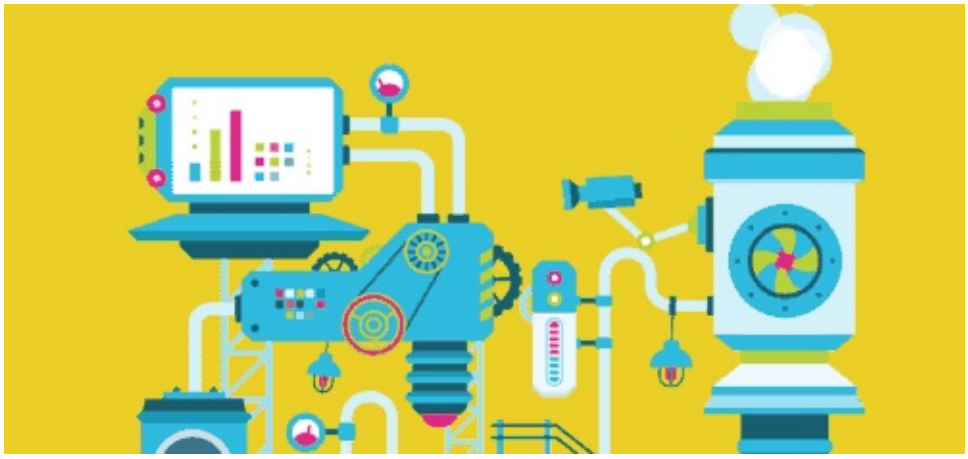
Module #5 - Machine Ready (Required)

More Than Programming...

As every developer knows, becoming proficient at web and software development requires more than just a grasp of programming languages alone. Instead, successful developers must also possess a comfortable familiarity with a wide assortment of tools and technologies. These tools enable software to be created, to be collaborated over, and to be shared with users around the world.

In this class, you can expect to be exposed to a wide range of tools with names you've never heard—tools like *Heroku*, *Git*, *Emmet*, and *Robomongo*. At first, the sheer number of tools you'll be expected to use may seem overwhelming but trust us! With a little time and with a little effort, they will be as familiar to you as a scalpel is to a surgeon or a sewing needle is to a seamstress.

They're all just tricks of the trade.



Ready for Action!

Coming into your first day of class, you will be expected to have a number of tools already installed. This will ensure that you are ready to start coding right from the start. The purpose of this module is to walk you through the process of installing all your tools and to give you a brief primer on the roles they play.

After completing this pre-work module, you will have each of the following installed:

- Google Chrome
- Screencastify

- Slack
- Visual Studio Code
- "Open in Browser" Visual Studio Code extension
- Git
- Git Bash (Windows)
- Terminal (Mac, Pre-Installed)
- Homebrew (Mac)
- Heroku Toolbelt
- Node.js
- MAMP

In addition, you will also have accounts on each of the following websites:

- LinkedIn
- GitHub
- Stack Overflow
- Slack

Having trouble with set up? Not to worry. Your instructional staff will help you troubleshoot any errors and answer any questions on the first day of class. Just sit tight until then!

Tools for Fools

Before we start installing everything willy-nilly, let's take a moment to examine each of these tools to better understand the role they play.

Google Chrome

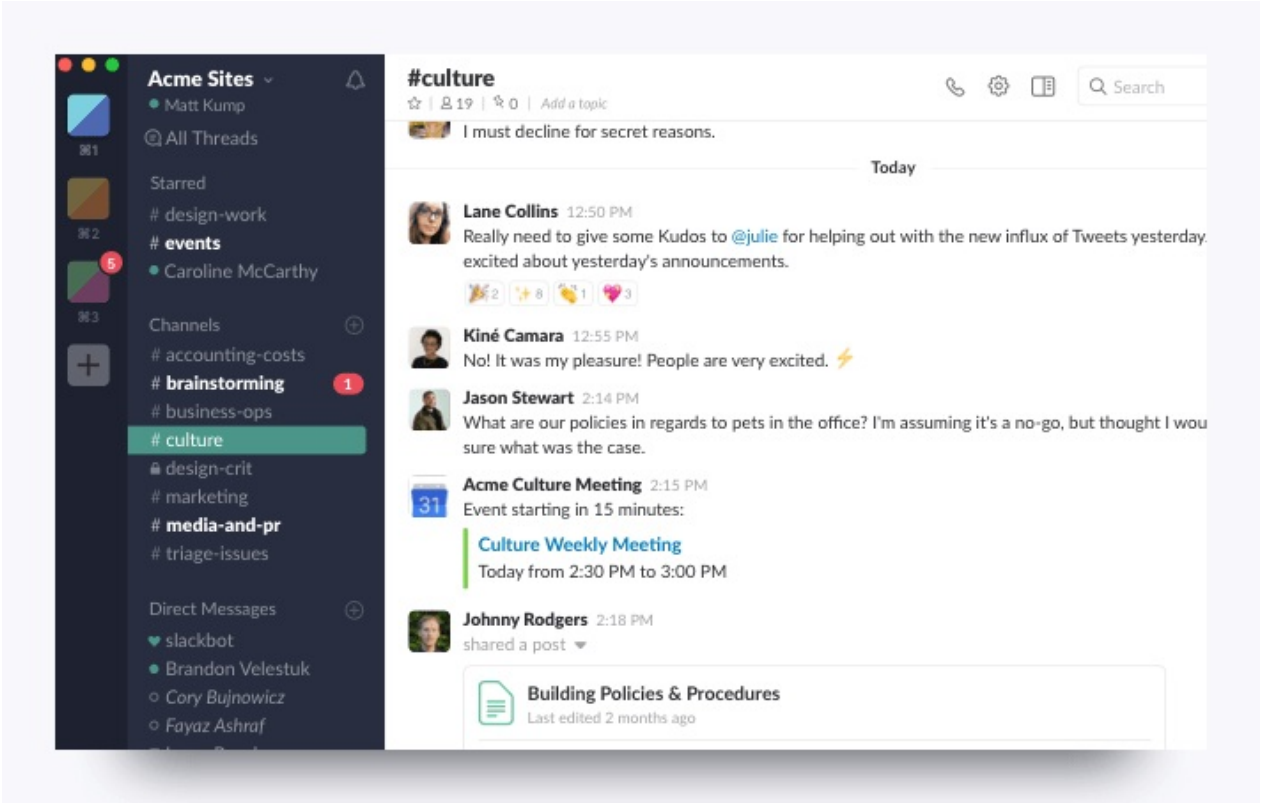
This one is pretty straightforward. It's a web browser. In our case, we'll be using it to quickly *see* if our code is working. While in truth, you can use any web browser, Google Chrome has a number of tools that make it an ideal platform for developing so we strongly encourage you to make the switch.

Slack

This is one you will be using literally every single day for the next six months. Slack is an online communication tool that is a mix of forum, of instant messenger, and of email—all rolled in one. It's a tool that is used by countless organizations worldwide.

In our Bootcamp, we'll be using Slack extensively to send code snippets during class, to relay important announcements, and to facilitate group exercises. You will receive the link to your class-specific channel during orientation. You will definitely want to have this *installed*

on Day 1. (Note how we said *installed* and not simply logged into. While the web client is good, for our class, you will want to install the actual program on your machine.)



Visual Studio Code

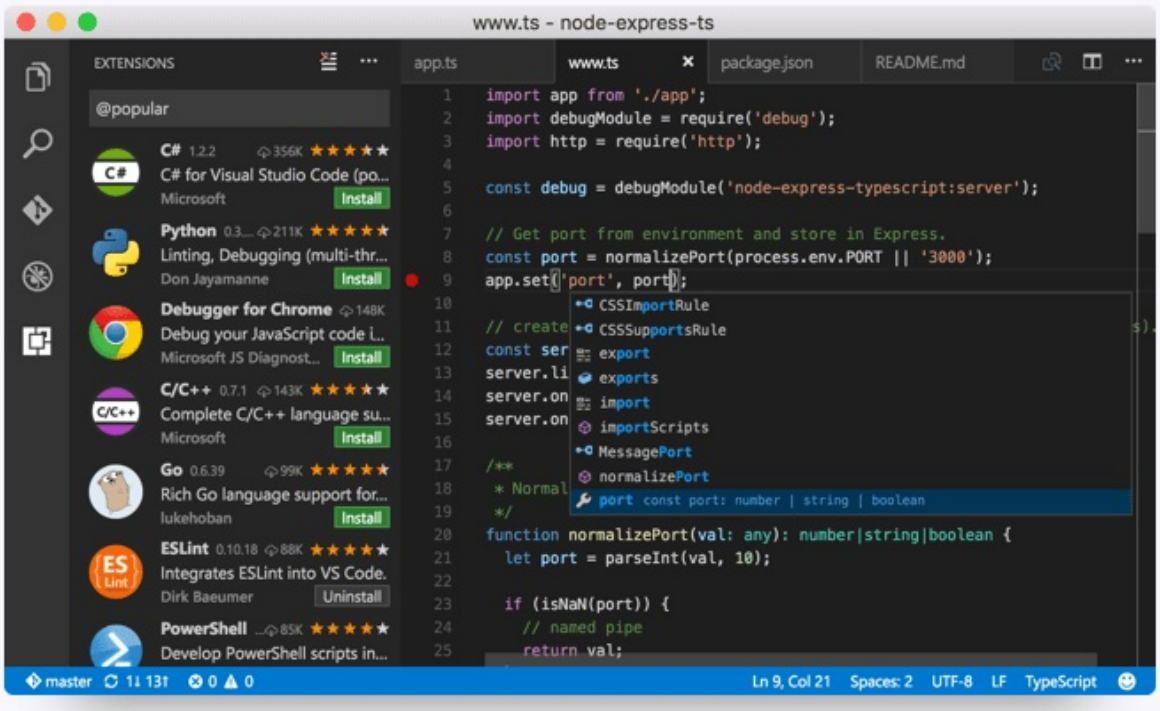
Oh, the power of Visual Studio Code! A little program that does so much!

Visual Studio Code is a free, open-source "text-editor". Now, for the uninitiated, the first thought that comes to mind when we say "text editor" may be something basic like Notepad or TextEdit. But for developers, text-editors are like the cozy pillow on which they rest their head. This is because fundamentally programming is all about creating "text" in files with various extensions. When we create a block of HTML, like the one below:

```
<!DOCTYPE html>
        <html lang="en">
        <head>
                <meta charset="UTF-8">
                <title>Document</title>
        </head>
        <body>

                <h1>Yo Yo Yo</h1>
                <h2>I'm the big man money</h2>

        </body>
</html>
```

What we've really done is just created a block of text. There are funny symbols in there like those angle brackets and what not, but at its most base level -- it's just text.

Now to a simple text editor this is where the comprehension stops. Our block of HTML remains a block of text. But to more powerful text editors like VSCode, on the other hand, these blocks of "text" are immediately recognized as code (so long as you include the right file extension). This means that VSCode can give us a more visually intuitive understanding of the code through indicative coloring, "smart" tabs, and autocomplete functionality. The end result is that creating the above block of HTML becomes a more natural process and one that can be debugged more quickly.



Additionally, as you will find in this class, VSCode is also powerful in its ability to be extended through the use of plugins. This means that we can easily incorporate free add-ons that enable VSCode to make the process of coding even easier than before.

Open in Browser

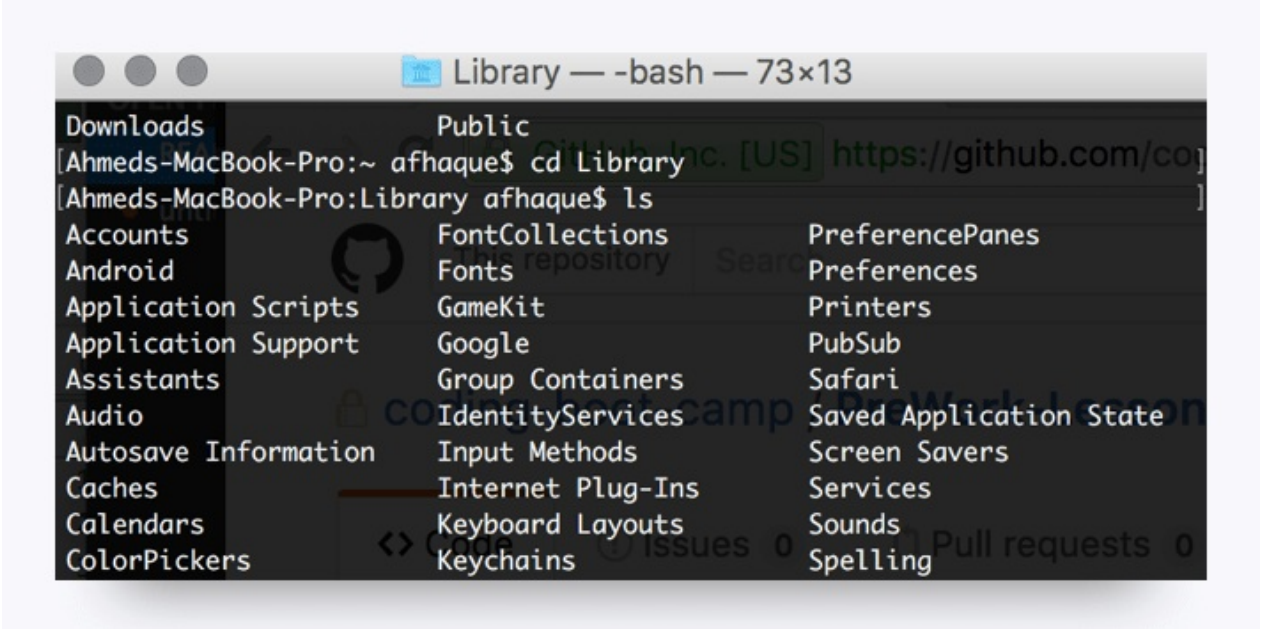
For now, the only extension we recommend installing with Visual Studio Code is the "Open in Browser" extension. This will allow us to open HTML files we are editing in VSCode in our web browsers without having to go through the file explorer/finder.

Git Bash / Terminal

Git Bash (PC) and Terminal offer a command line interface for working with the files and folders on your computer.

So, is it like Finder or Windows Explorer?

Kind of... except there are no pictures or visuals. It's just a box with text.



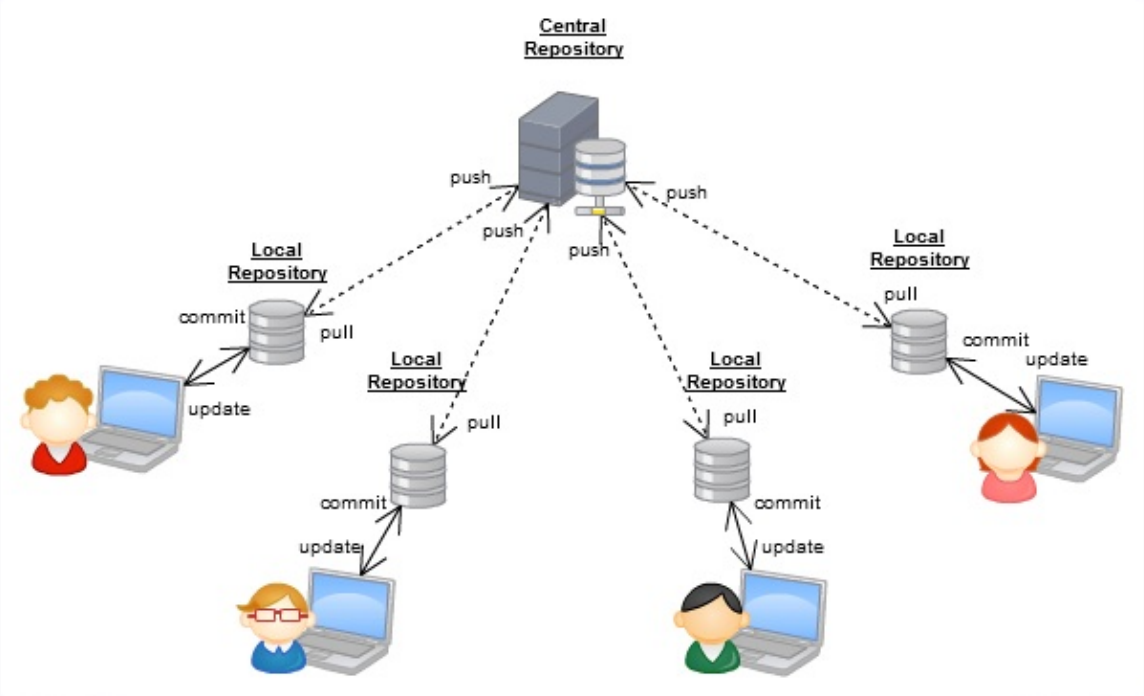
Uh, why would I want that?

You'll come to understand over time, but in many situations, utilizing a command line interface can be faster and can be more powerful than relying on the operating system's GUI. You'll get plenty of exposure to the command line on day one.

Git / GitHub

Because code files are ones in which multiple developers need to carefully build upon each other's work, Git offers a specialized set of strategies for orchestrating the collaboration. GitHub then takes all these collaborative actions and stores them online. In a way, GitHub serves as a sort of Dropbox for coders. It offers a central place for individuals or teams to

upload their code, to view revision history, and to make changes to a master set of files. You'll come to learn a lot about Git and GitHub in your first week of class. You will receive the link to your class-specific repository during orientation.



Homebrew

Homebrew is a Mac-specific toolkit that makes it easy to install, using the command line, a variety of applications. It can greatly simplify the installation process for various tools you'll be using in your development career.

Heroku / Heroku Toolbelt

While you may not feel ready yet, very quickly, you'll be creating complete websites on your computers. But once you have these sites made, how do you get them online for the world to see?

Hosting platforms like Heroku effectively serve as a dumping ground for web applications. These platforms are set up to take your web applications' code, to activate them, and to then assign them to a URL for other visitors to see. In a sense, they *host* your applications so that every internet user has access to them. You'll learn a lot about how this works towards the tail end of the course.

As it relates to the pre-work, you'll be installing the Heroku Toolbelt, which offers a set of easy-to-use tools for interacting with the Heroku platform online.

Node.js

Oh, Node! (Get the pun?)

Node.js is a JavaScript library that we'll be using extensively for the back-end of our applications. Don't worry if that doesn't make much sense yet. We'll be spending effectively half the program working on Node. You'll become very proficient in its use and in its utility by the end of the course.

MAMP

MAMP is a popular *stack* of tools that is used by many web developers. In our case, we'll install it as a back-up, in case your machine runs into any issues with MySQL. (It happens. Tools are finicky sometimes.)

Collect Your Tools!

And that's it!

It's time to collect your tools and begin. As you'll see in the links below, we have guides for both Mac and Windows users on the process for getting setup. Follow the instructions closely and do your best with the information you have. (Yes, we know there is a lot to install.)

Just one bit of advice! As future coders, you will frequently be looking at documentation to install and to utilize unfamiliar tools. Resist the urge to stop and ask, "Am I doing this right?" Instead, just take your best stab at it. This is an important attitude to start developing *now*. A lot of what you'll be exposed to over the next six months will be unfamiliar. Don't be phased by it, and don't get bogged down by it. Trust your instincts! We'll make sure that anyone who is lost gets the help they need on Day One.

Good luck! Make sure your tools are extra pointy.

Assignment (Required):

- Get Yo' Tools Installed on Windows
- Get Yo' Tools Installed on Mac

Copyright

Assignment #5 - Get Yo' Tools Installed (Windows)

Overview

In this assignment, you will be installing all of the required tools and software necessary for the class. We've got a lot to install, so buckle in and get ready!

Before You Begin

Make sure you sign up for these services; you'll need all of them throughout the course.

- LinkedIn: https://www.linkedin.com
- GitHub: https://github.com
- StackOverflow: http://stackoverflow.com

Don't just create logins. Job recruiters often scour these sites in search of job candidates; make sure you provide your headshot and your contact info on all three services.

P.S. Don’t forget to outline your skills and your work experience on LinkedIn.

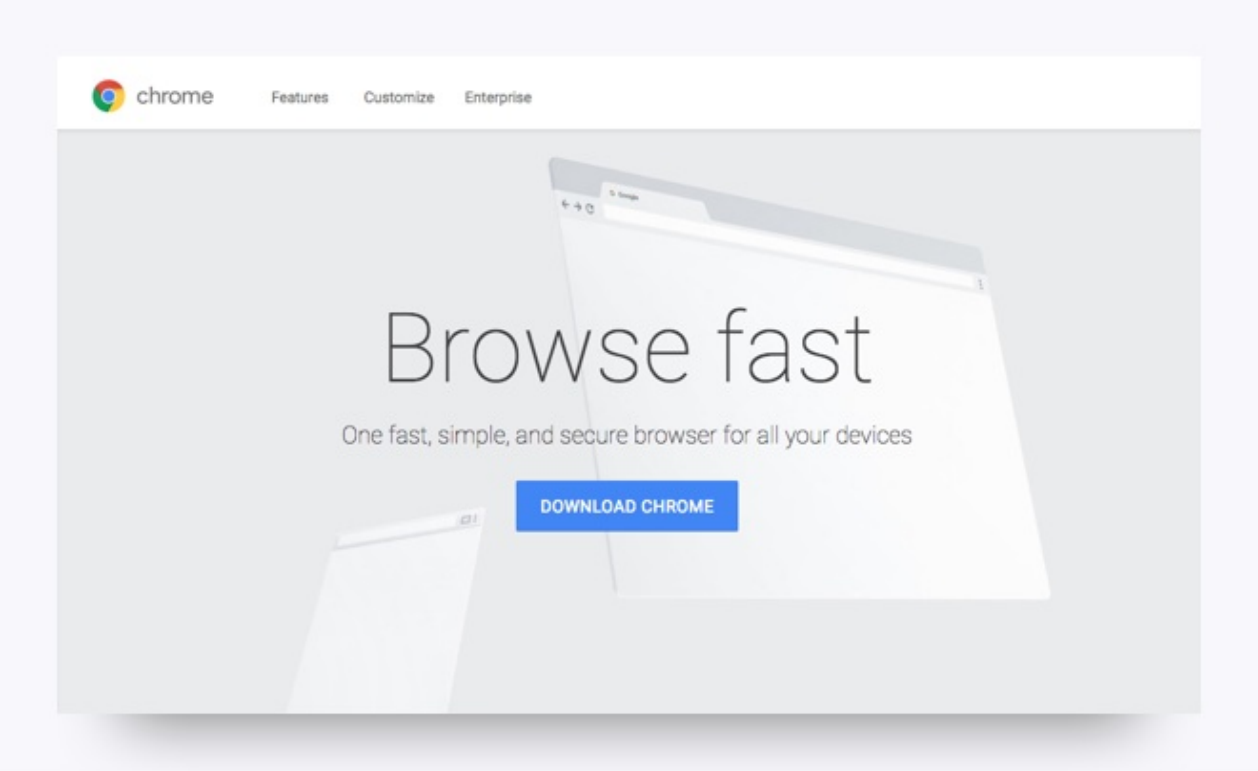
Also, make sure to accept the invite for your section on Slack as well. You will receive the link to your class-specific channel during orientation. If you haven't received an email from us yet with the section's domain name, contact your Student Success Manager to assist you.

Da Big Installation Enchilada

The rest of this assignment will walk you through the specific steps associated with installing each of the tools you'll need. Follow the instructions closely!

Google Chrome

During this course, consider Chrome *the* web browser; it comes loaded with tools for quickly editing the web pages you’ll create.



1. If you don’t already have Chrome installed, visit their download page.
2. Download, open, and run through Chrome’s installation file.

Screencastify

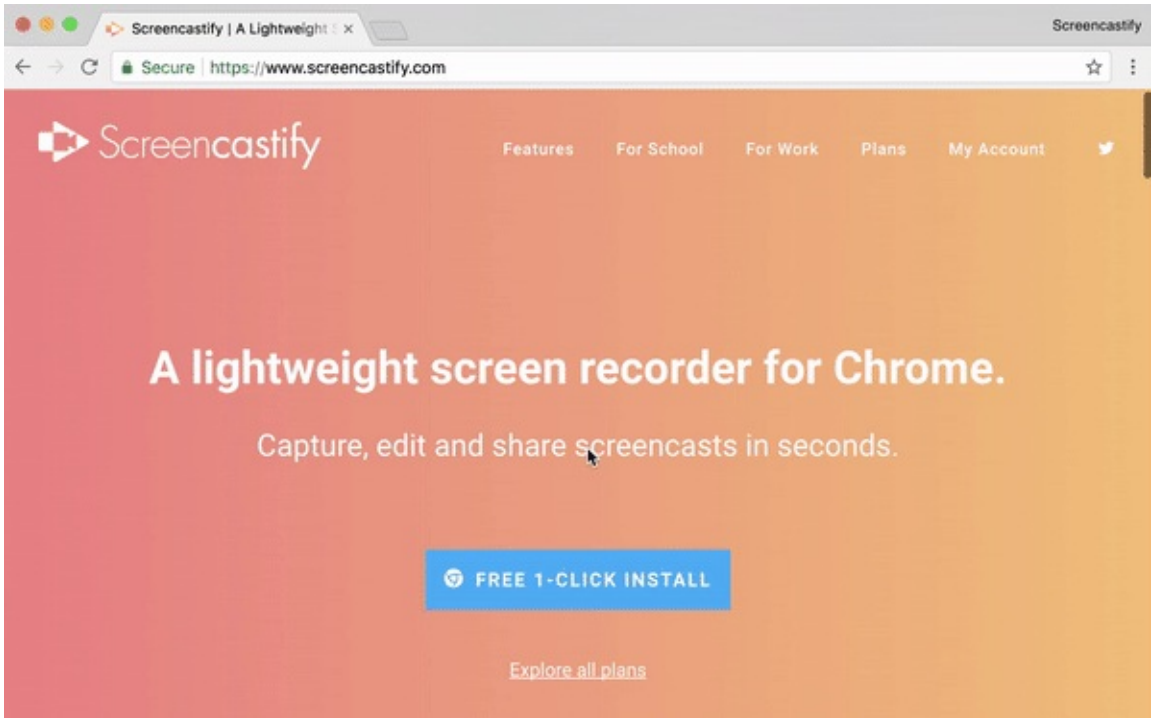
There are two places where you can install the Screencastify extension:

1. The Screencastify Website.
2. The Chrome Web Store.

Note: You must be using Google Chrome to install and use Screencastify

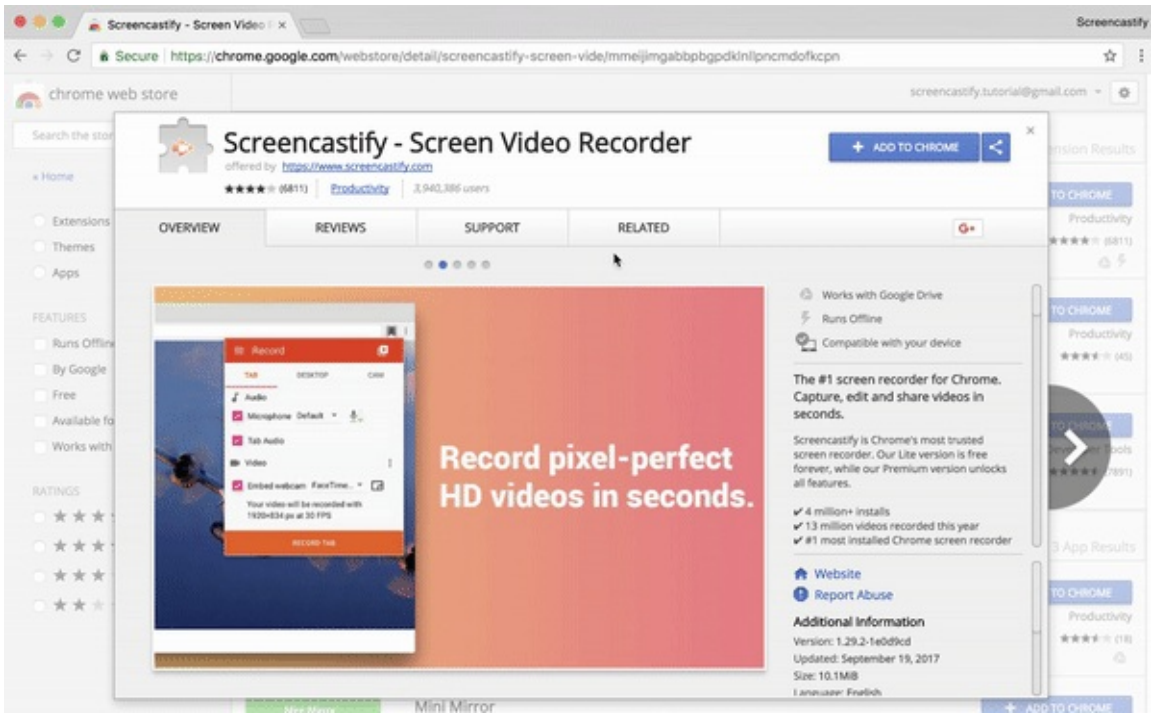
Install Screencastify from their Website:

- Go to the homepage and click the blue "**Free 1-Click Install**" button. You'll see a confirmation box confirming that you want to install the extension. Click "**Add extension**". Screencastify will then begin downloading.

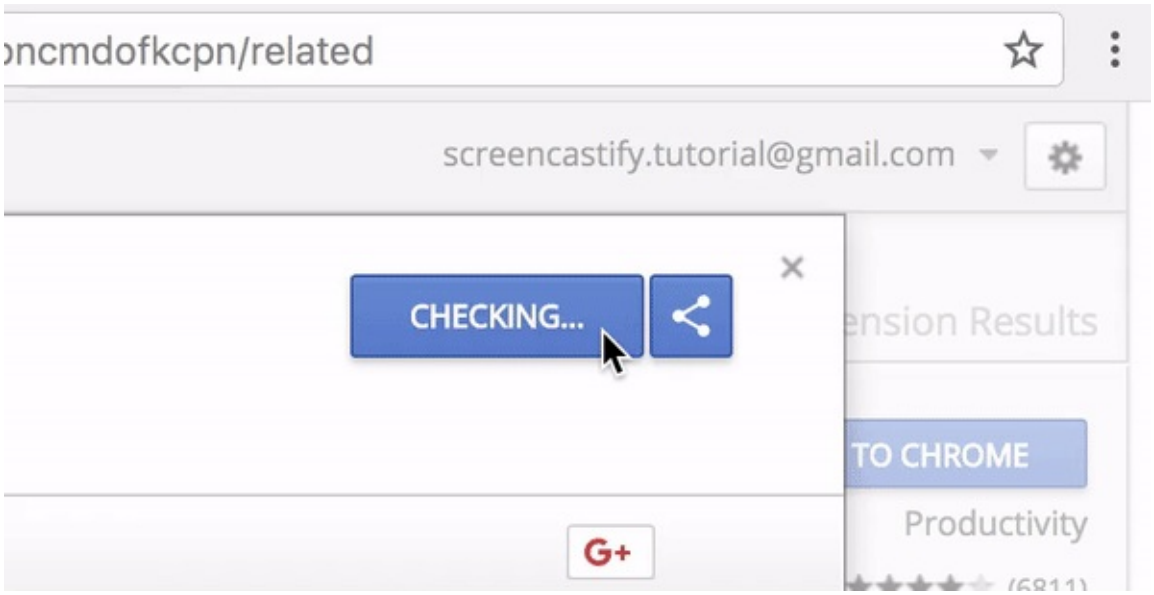


Install Screencastify from the Chrome Web Store:

- Click here to view their listing in the Chrome Web Store, then click "**Add to Chrome**". You'll see a confirmation box confirming that you want to install the extension. Click "**Add extension**". Screencastify will then begin downloading.



- Once Screencastify is finished recording, their grey film strip icon will appear in the top right-hand corner of your browser. Click the icon to get started.

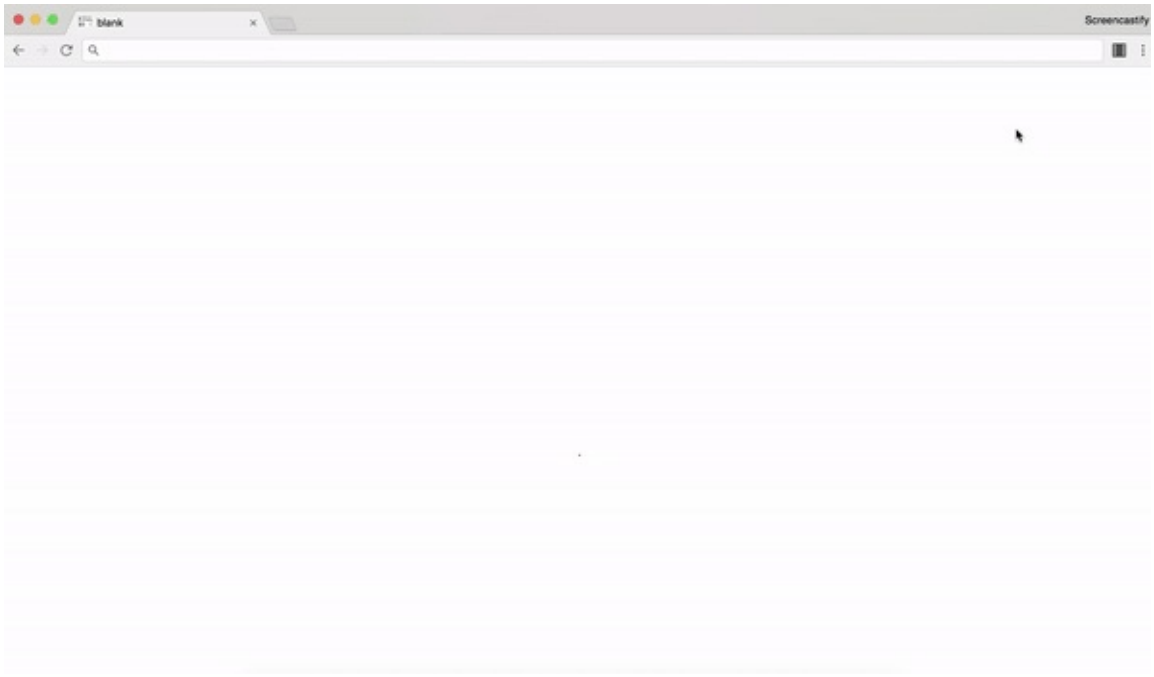


Set up your webcam and microphone:

The first time you click on the Screencastify extension icon, you'll be asked to give Screencastify permission to access your webcam and microphone. They ask for these permissions so that you're able to narrate over your recordings and embed / record your webcam.

Note: While you're required to allow access to your camera and microphone in order for Screencastify to function properly, you don't need to include audio or webcam video in your recordings. Screencastify never records audio or video unless you expressly ask it to.

- When you click on the grey film strip icon, you'll be taken to a page where you can allow camera and microphone access.
- Click the "**Setup Camera Access**" button. Chrome will confirm once more that you'd like to allow microphone and camera access. Click "**allow**".



- And that's it! From now on, you'll be able to narrate over your recordings and record your webcam whenever you'd like.

Troubleshooting: If Screencastify is unable to detect your microphone, click here. If it's unable to detect your webcam, click here.

Connect your Google Drive to Screencastify:

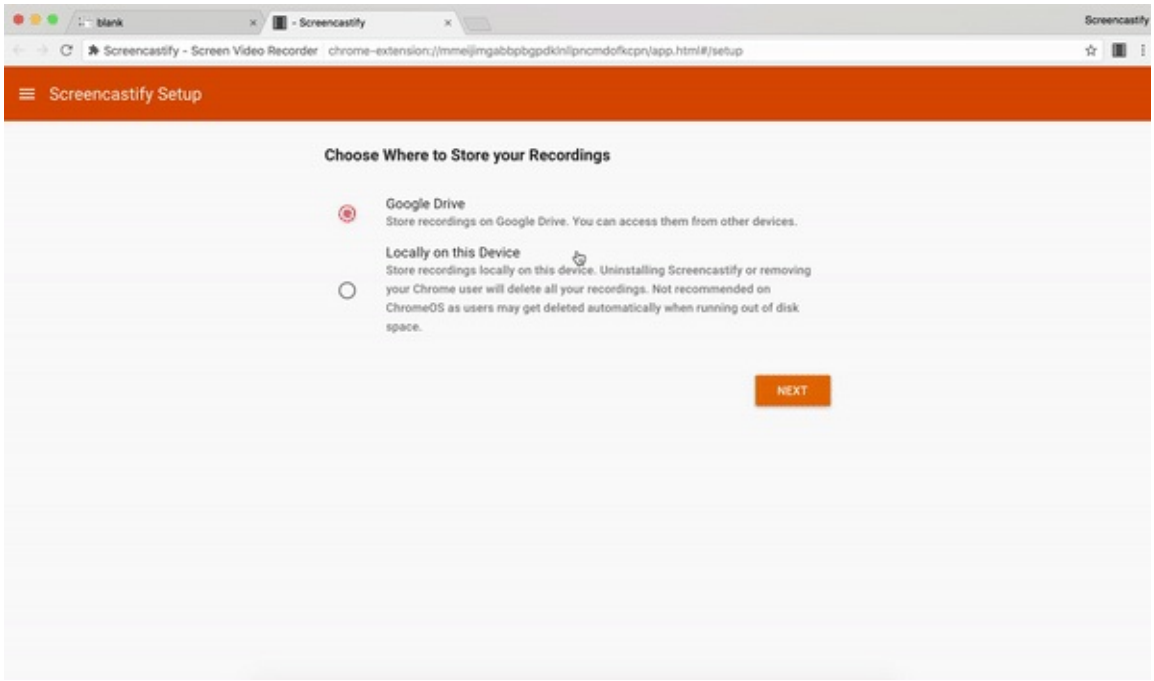
When you connect your Google Drive to Screencastify, all of your recordings will automatically save to your Google Drive in a folder titled " Screencastify". Google Drive, along with many other useful apps come free with a Google Account. If you don't already have one, go create one!

Depending on how long a particular recording is, this process could take a few minutes to complete after you end a recording.

Note: Screencastify highly recommends connecting your Google Drive to Screencastify and saving all of your recordings there vs. locally on your computer. It's much safer.

The first time you open the Screencastify extension, you'll be asked where you want to store your recordings (right after you allow camera and microphone access).

- Make sure that "**Google Drive**" is selected, then click "**Next**". You'll then need to sign in with your Google account and allow Screencastify permission to access your Google Drive. You can sign in with **any** Google-based account, not just a @gmail.com email.



Now, all of your videos will automatically save to your Drive after you finish recording.

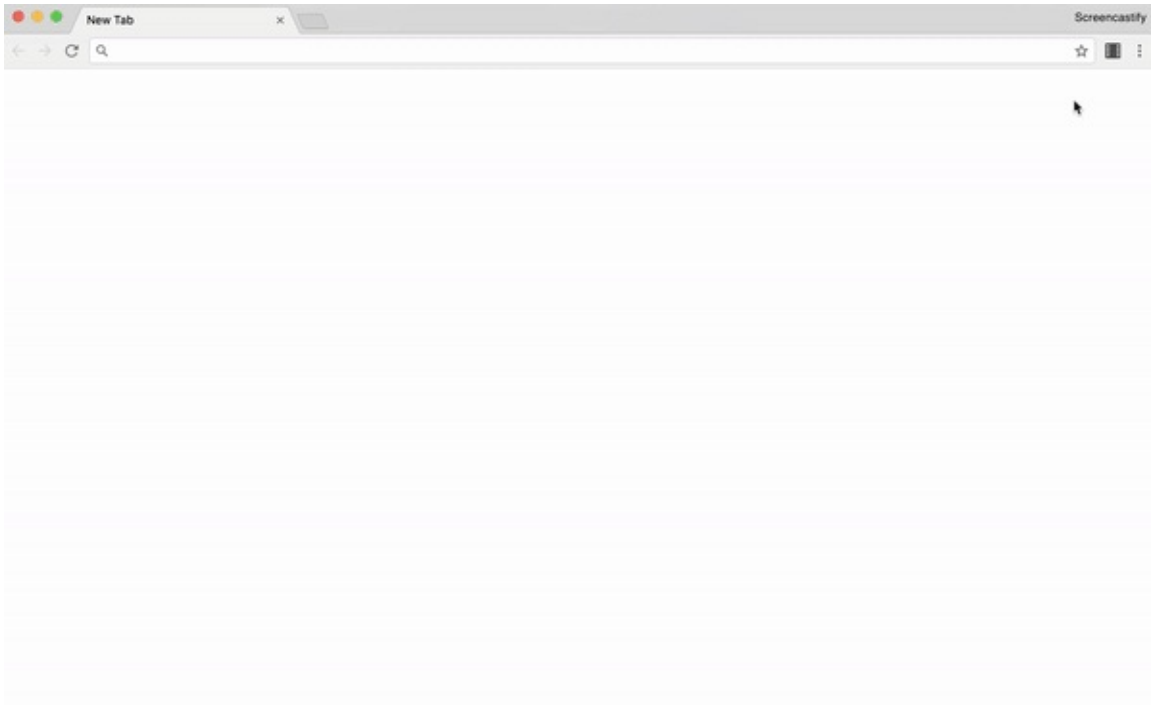
As soon as you finish your first recording, a new " Screencastify" folder will be created in your Drive. All of your recordings will be saved in this folder.

The videos you will see in Your Recordings library in Screencastify are synced with this "Screencastify" folder in your Drive. This means:

- If you delete a video in Screencastify, it will also be deleted from your Drive
- If you move a video out of the "Screencastify" folder in your Drive, it will no longer be accessible from Your Recordings in Screencastify
- If you rename a video in Screencastify, it will also be renamed in your Drive

Note: Screencastify never views, modifies, stores, or in any way interacts with the other files in your Google Drive.

You can always change the default storage location for your recordings, even after initial setup. Simply open the extension menu and click "**Options**". The first thing you'll see is an option to select where you want to store your recordings.

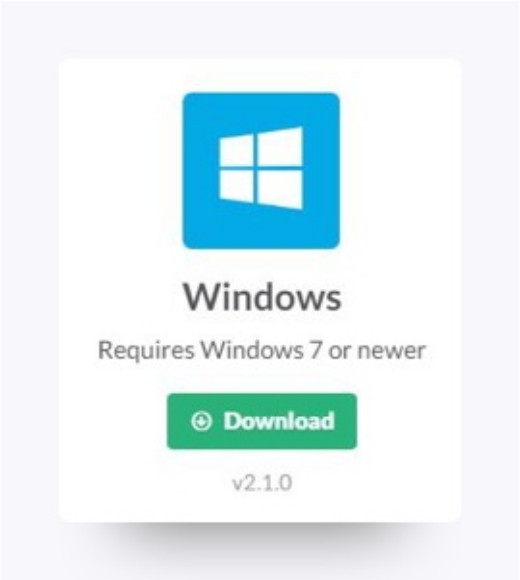


Important Resource: Want to know how to record, save to your Google Drive and share your recordings? Check it out here!

Slack

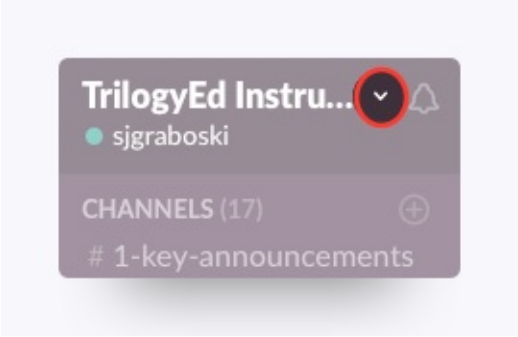
You'll be messaging your instructor, your TAs, and your fellow classmates with this business-centric chatting app. The teaching and career staff will post some of their most important announcements here so set this program up as soon as you can. You will receive the link to your class-specific channel during orientation.

1. **If you don't have the Slack app yet,** go to https://slack.com/downloads. Select "Windows" to download the installation file, and then open the program.

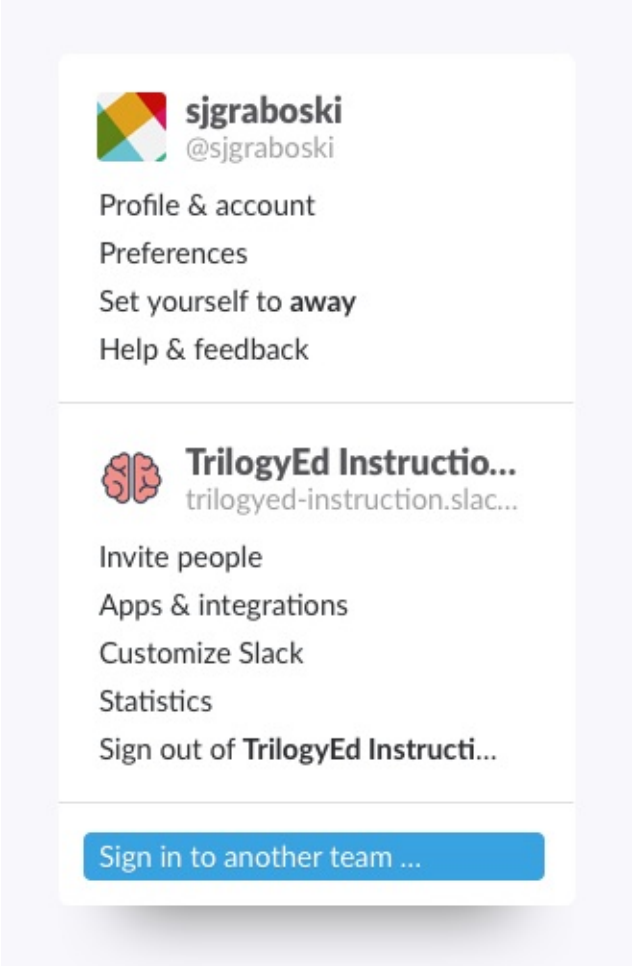


2. **If you already use the Slack app,** you just need to add our channel to your application.

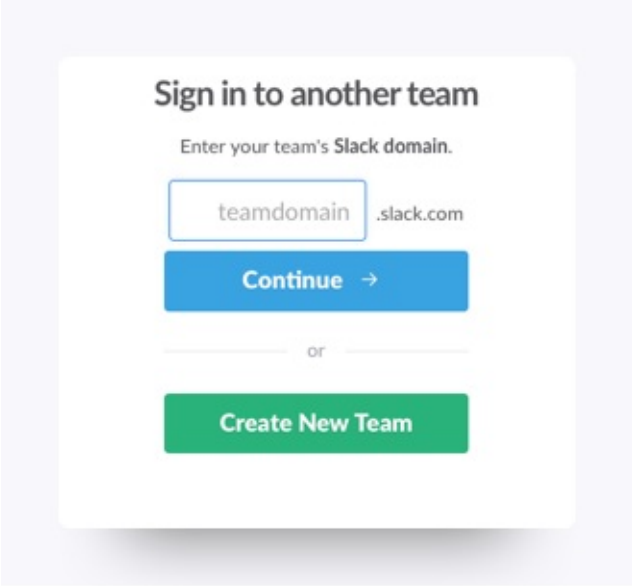
- Click the header of your current Slack Channel.



- Then select “Sign in to another team …”



3. As you run through the guide, make sure you do the following:
    - Enter in the domain we gave you for Slack.



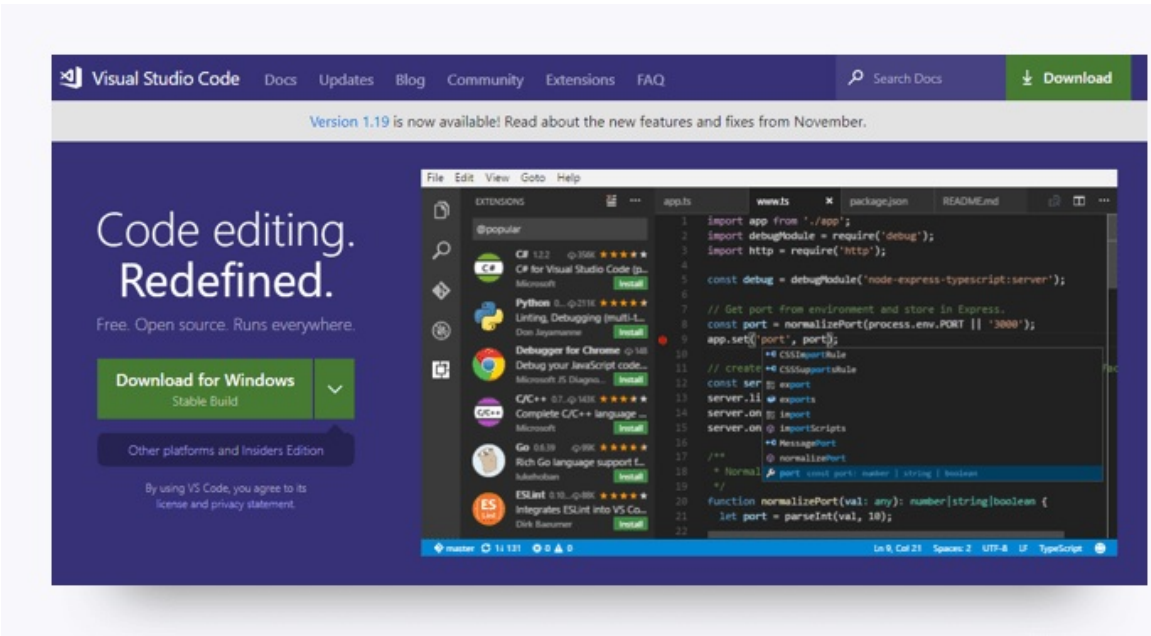
- Enter in the email with which we invited you, as well as your password, when prompted.

4. When you see the chatroom, you're finished.

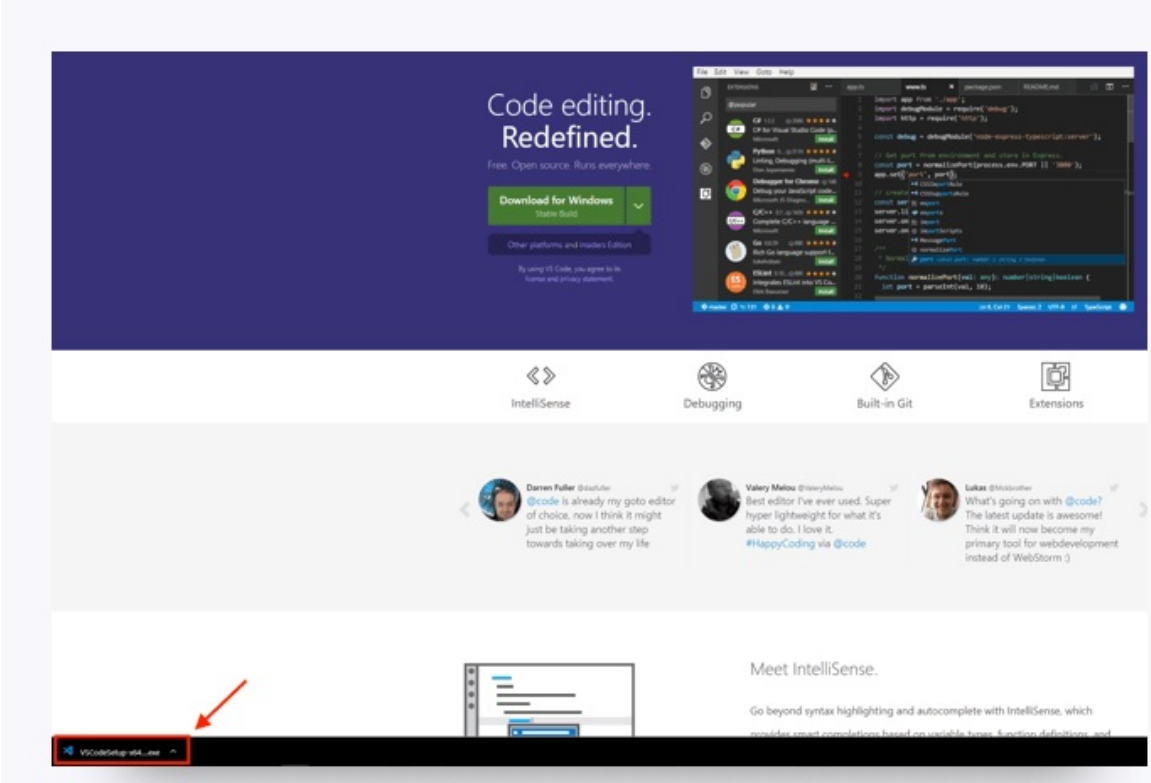
Visual Studio Code

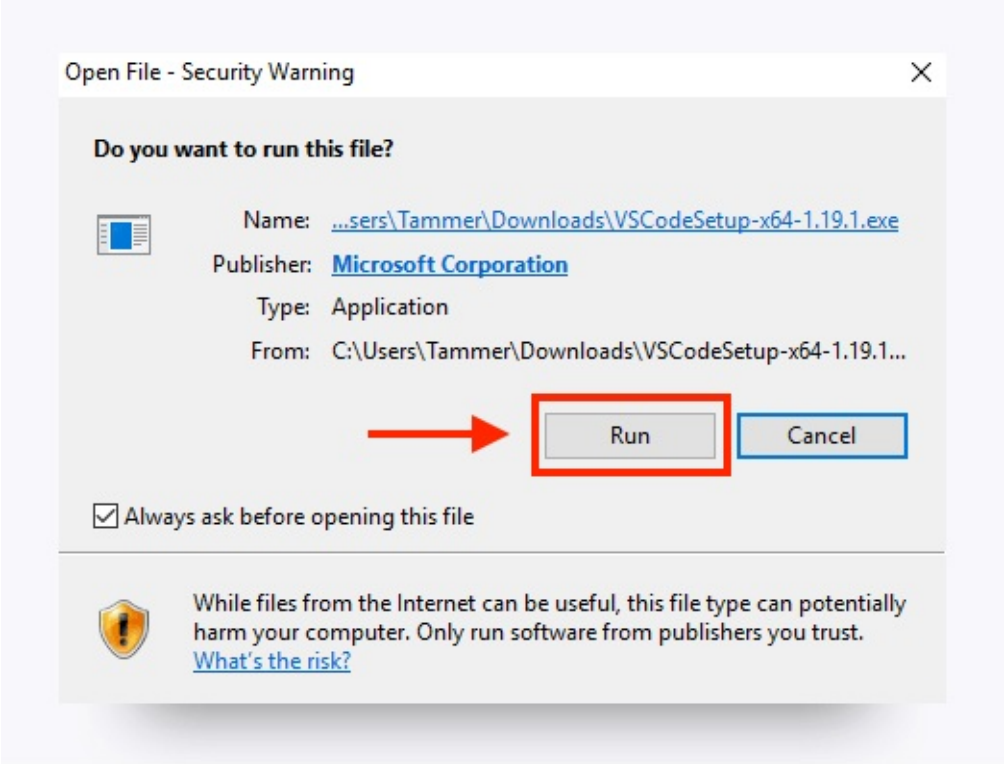
This text editor features a variety of indispensable tools and customizable components that will help you code your websites and apps.

1. Download the appropriate version for your operating system.

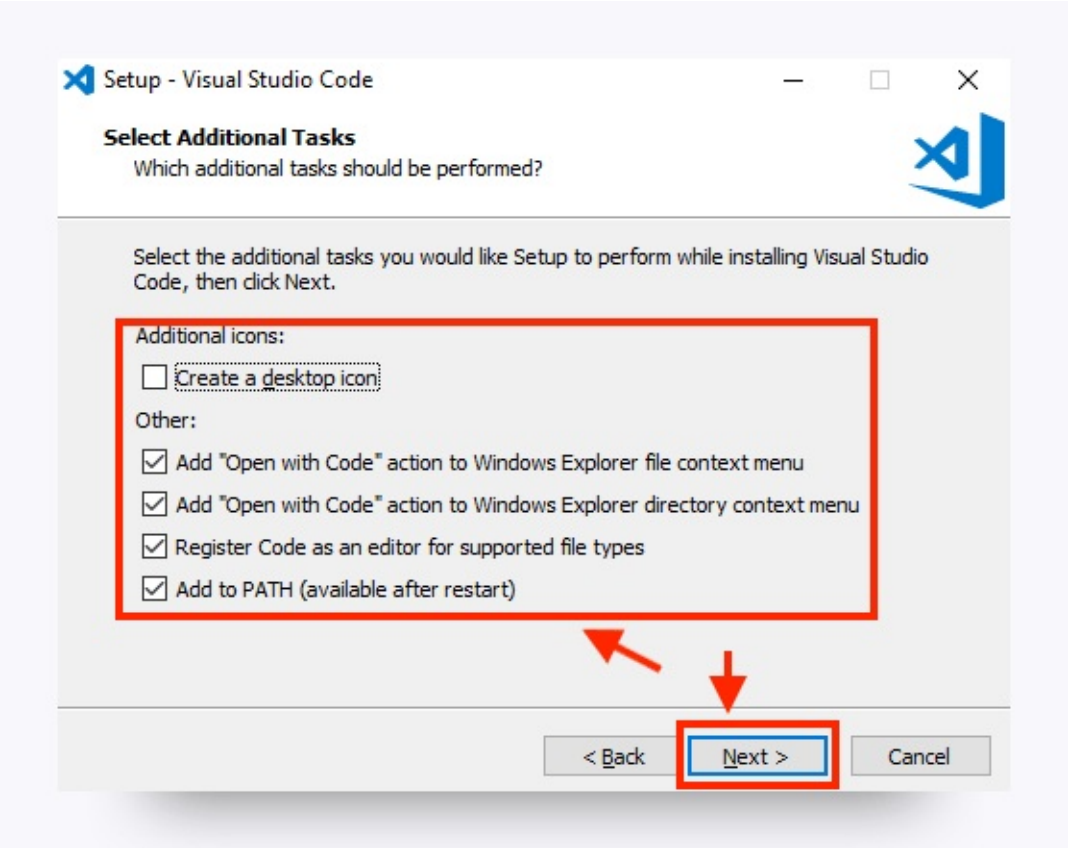


2. Once the download is complete, open and run the installer file.





3. Go through the prompts presented in the installer. When you reach the following screen, select the following settings.

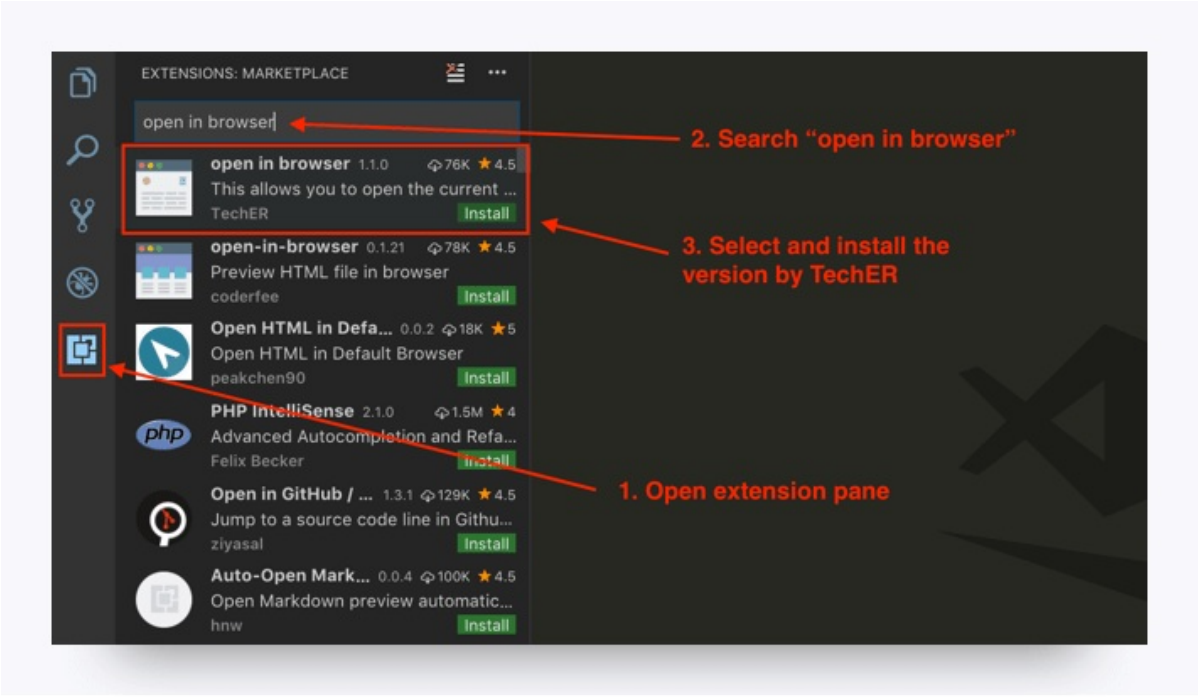


Note: Once the installation is complete, you should be able to access Visual Studio Code from your start menu.

Install "Open in Browser" Extension

With this extension, you can open HTML files in your web browser from the Visual Studio Code editor.

1. Open Visual Studio Code.
2. Open the extensions pane and search for "Open in Browser".
3. Select the version written by "TechER" and install.

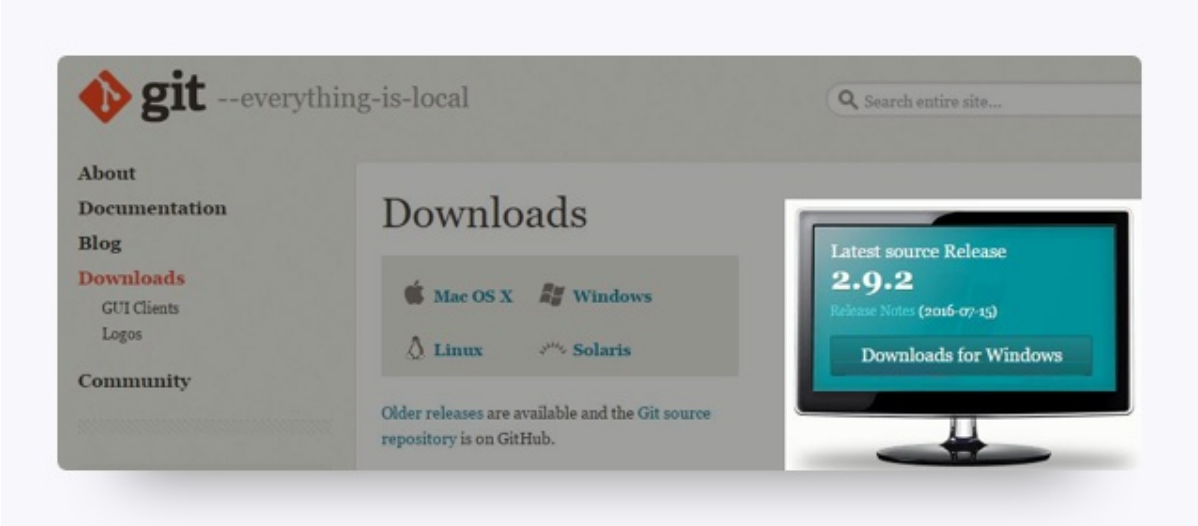


Git & Git Bash

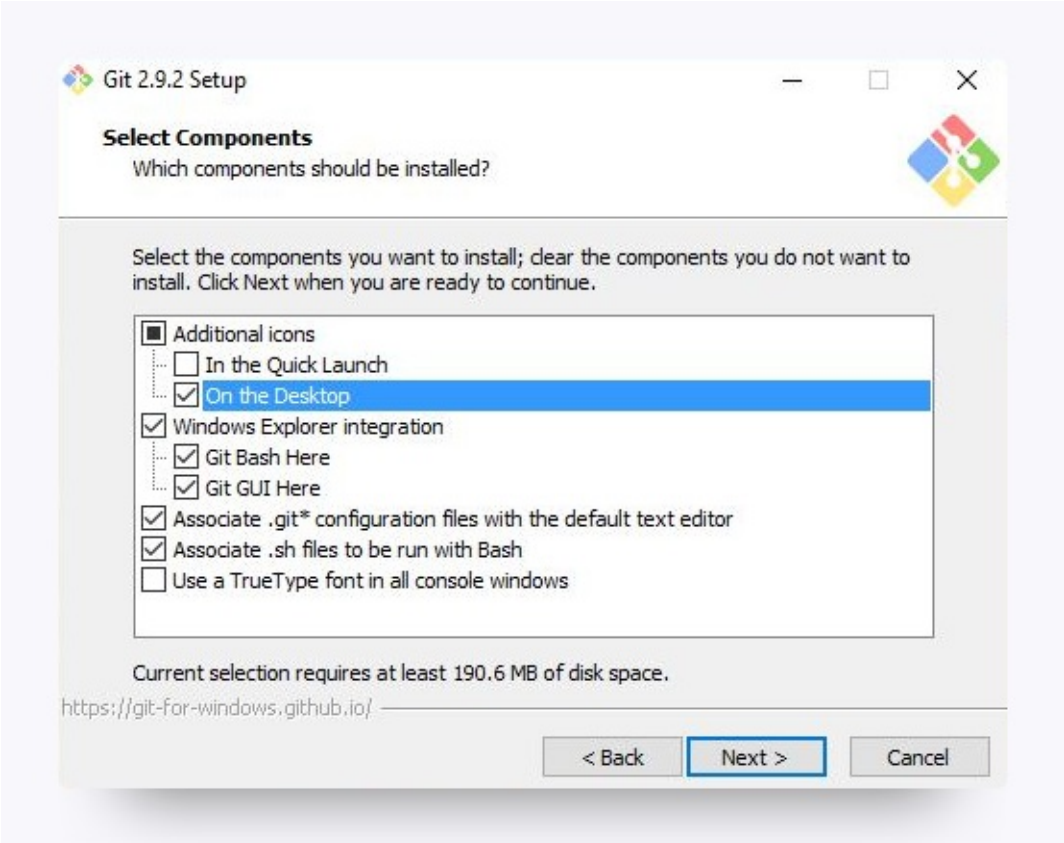
Coders depend on this tool for version control; the process of logging the development of programs and applications. This comes in handy during collaborative programming, when teams of programmers change, add, and remove code throughout a project’s directory; this process would be chaotic without Git.

The installation also includes Git Bash, or Bash for short. You'll be using this command line terminal throughout the course and during the rest of these instructions.

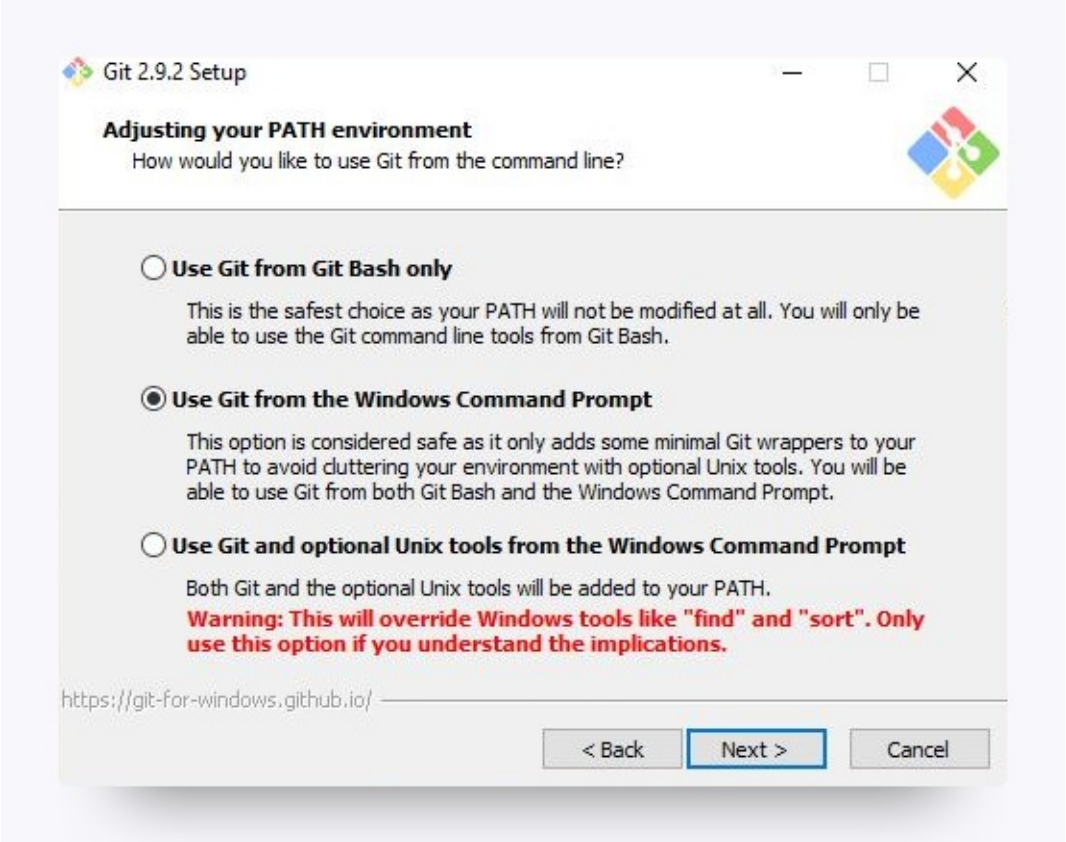
1. Go to Git download page: https://git-scm.com/downloads. Click on the download for your computer.



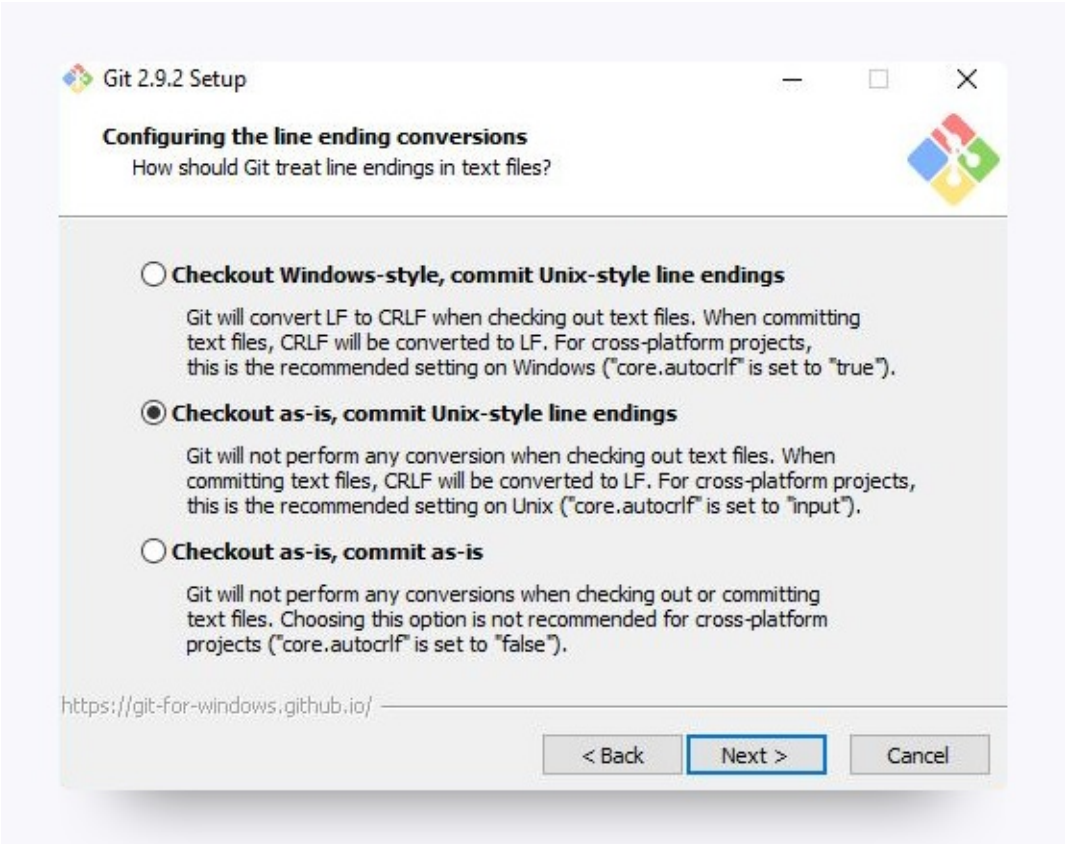
2. Run through the installation file. Make sure you check off the right boxes as shown in these four images.
    - Save Git to the desktop (this should save Git Bash to your desktop too).



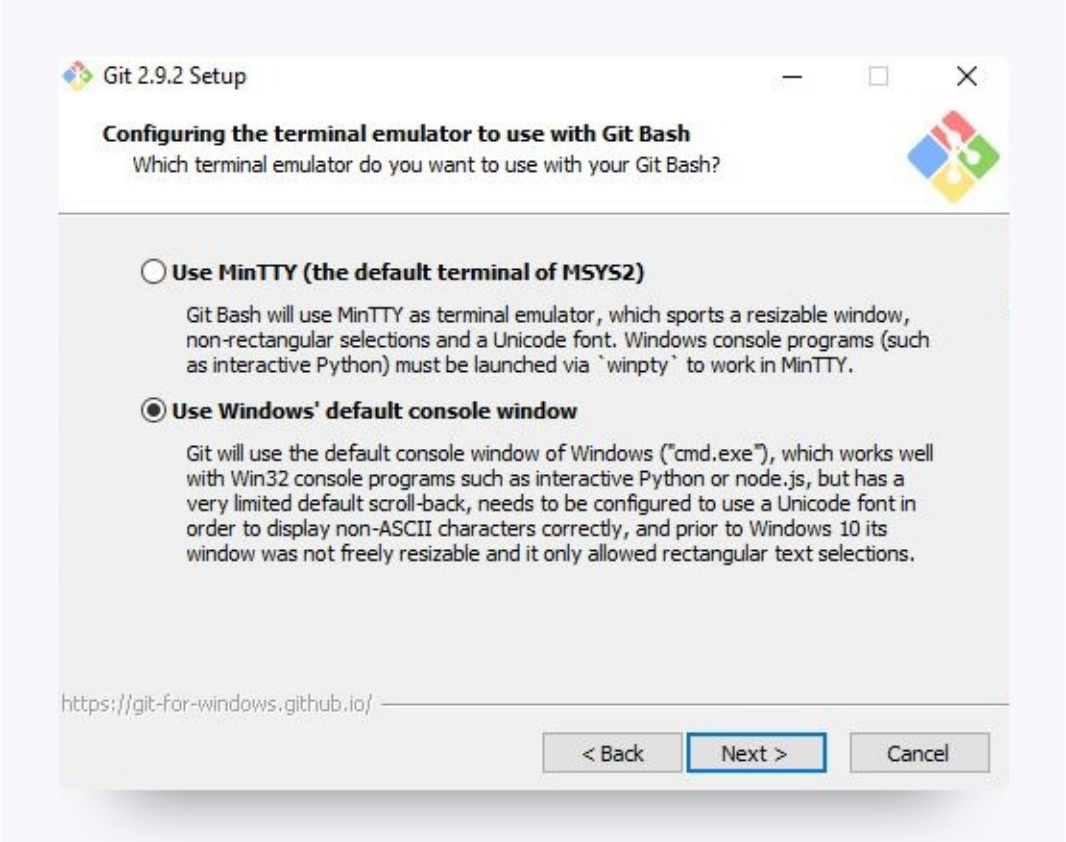
    - Use Git from the Windows Command Prompt.



- Checkout as-is.



- Use Windows' default console window.



Heroku Toolbelt

This tool lets developers deploy their web apps to the cloud, allowing anyone with the right addresses to access their creations.

1. First, you need to sign up for a free Heroku account: https://signup.heroku.com/.
2. Go to https://toolbelt.heroku.com. Download the installer.
3. Go through the install guide. Then open Command Prompt on your computer (not Bash).
    - Command Prompt, or cmd.exe, comes preloaded on Windows operating systems, but it will be located in different locations, depending on your version of Windows. Use your OS's search feature for Command Prompt, and it should pop up shortly.
4. Type `heroku` into the command line, and then press `enter`. If `heroku` doesn't work for you, type `heroku login` instead. When prompted, enter the credentials from your Heroku account, and then close Command Prompt.

Node.js

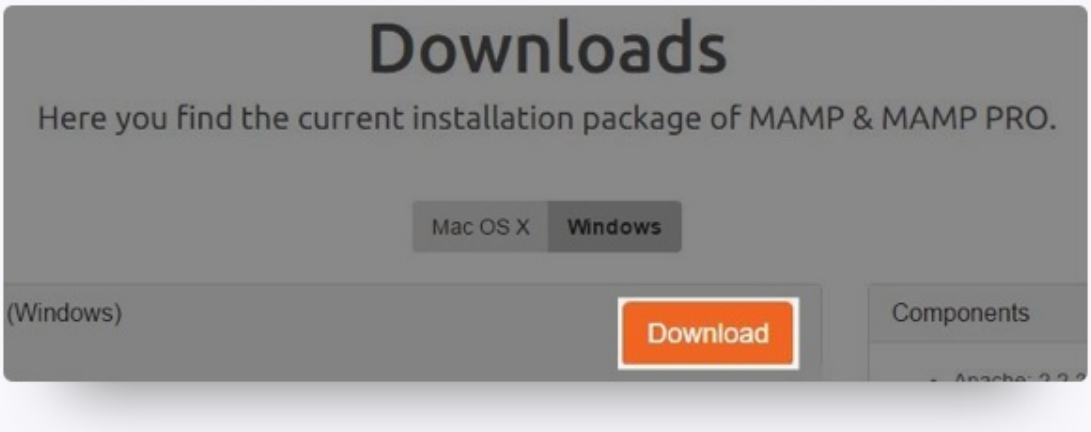
This runtime environment has quickly become the standard for coding back-end programs. Your code will run on Node throughout most of the course.

1. Go to the Node.js site: https://nodejs.org/en. Click the download button and run through the installation file.

MAMP

This tool lets developers run local servers for their back-end-reliant web apps. In other words, you can power some of your more complex sites without an internet connection.

1. Go to https://www.mamp.info/en/downloads/. Select Windows and click download.



2. Run through the installer.
    - When prompted, make sure you deselect MAMP PRO; that's a paid service, and

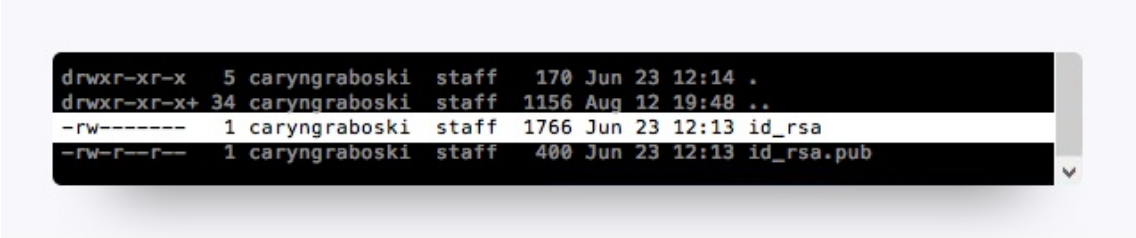
you won’t need it.

SSH Key

Generating SSH keys allows developers to interface with certain remote services without having to constantly type out login information. You're going to set up an SSH key for GitHub.

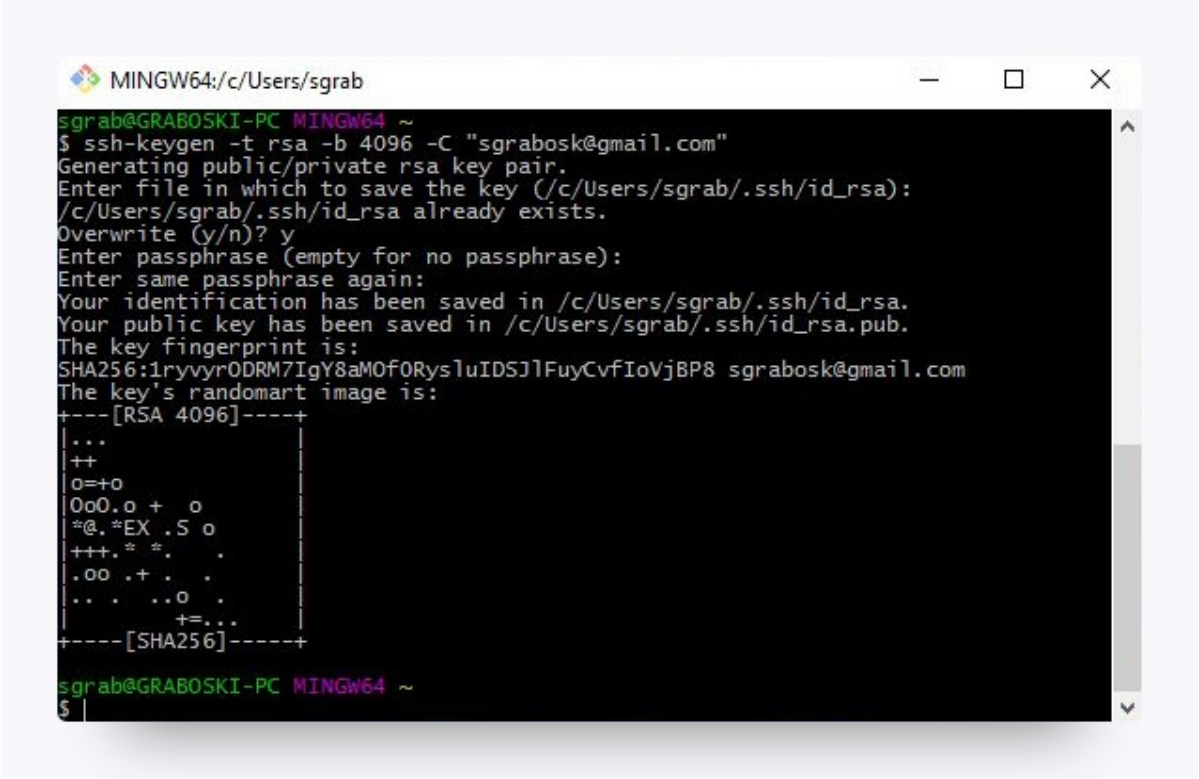
Without a key, you won’t be able to push your code to GitHub without entering a password each time; trust us, that would be as irritating as needing a key to open every door in your home.

1. If you haven't signed up for a GitHub account yet, you'll need to do so before moving on with these steps. Visit https://github.com.
2. Open up Bash.
3. We need to set up SSH keys. First, let’s make sure you don’t already have a set of keys on your computer. Type this into your Bash window **(copying and pasting will not work)**:
    - `ls -al ~/.ssh`
    - If no keys pop up, move onto step 4.
    - If keys do pop up, check that none of them are listed under `id_rsa`, like in this image:



        - If you do find a key with a matching name, then you can either overwrite it by following steps 4 to 6, or you can use the same key in steps 10 and beyond. Be advised that you'll have to remember the password tied to your key if you decide not to overwrite it.
4. Type in this command along with your email to generate your keys
    - `ssh-keygen -t rsa -b 4096 -C "YOURGITHUBEMAIL@PLACEHOLDER.NET"`
5. When asked to enter a file to save the key, just hit `enter`.
    - Also enter a passphrase for your key.
    - Note: You shouldn’t see any characters appear in the window while typing the password.

6. When you're finished, your window should look like this:

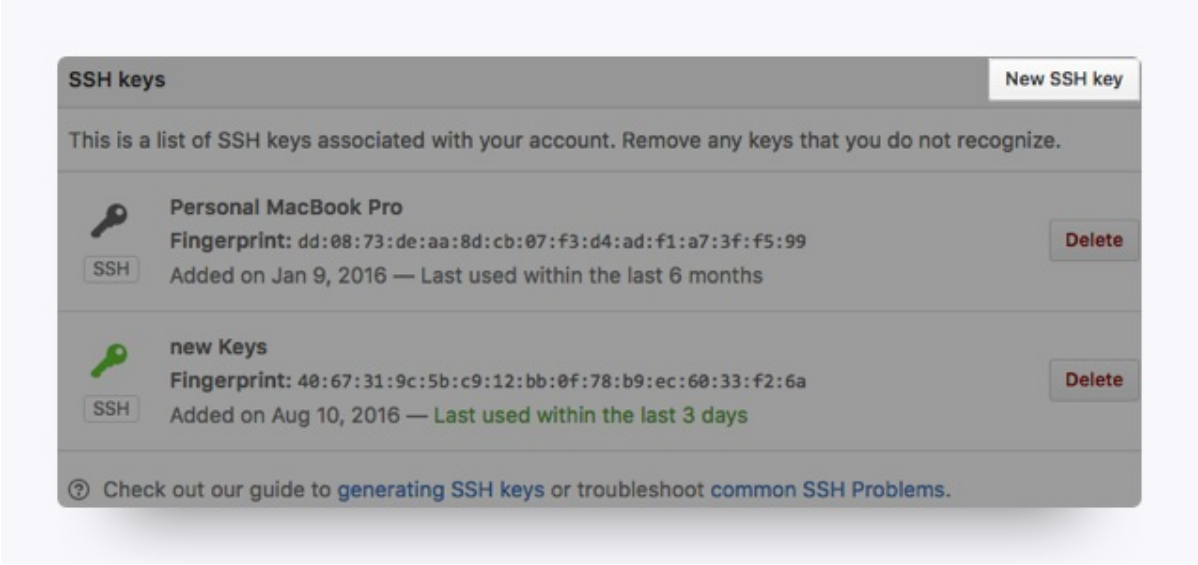


7. For the next step, we need to use a tool called ssh agent to link our key with our machine. Let's test whether ssh-agent is working. Run this command in Bash:
    - `eval "$(ssh-agent -s)"`
    - If your Bash window looks like the below image, move onto the next step.

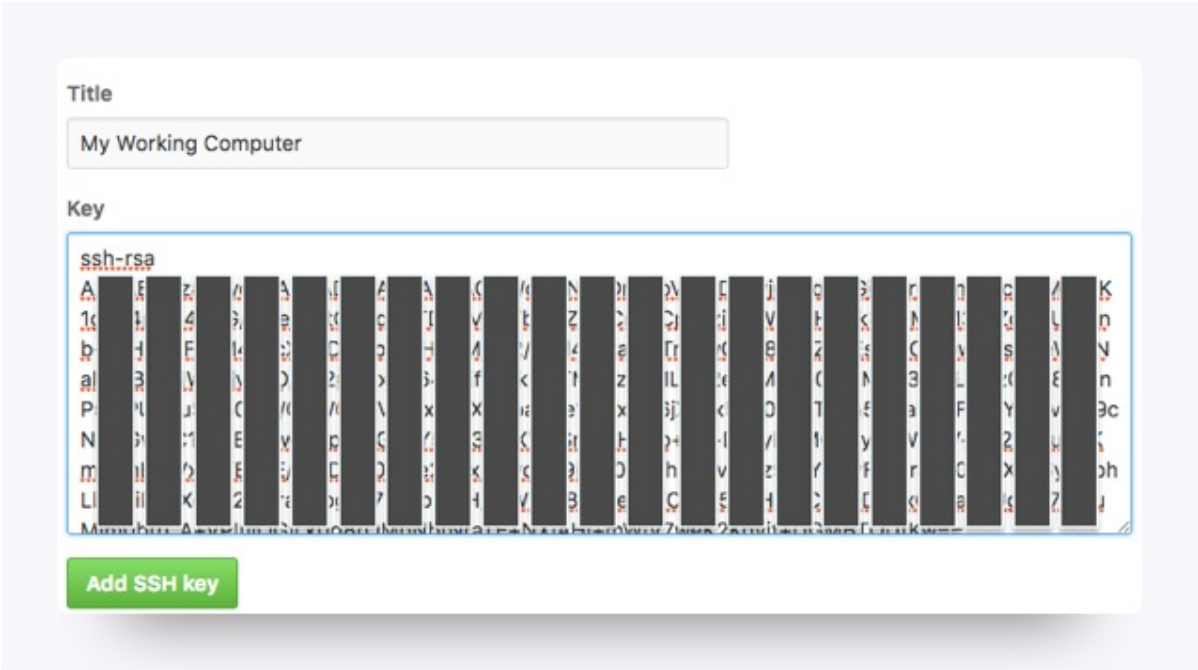
8. Now run this command:
    - `ssh-add ~/.ssh/id_rsa`
9. When prompted for a passphrase, enter the one associated with the key.
    - If you've forgotten this password, just create a new one, starting with step 4.
10. We need to add the key to GitHub. Copy the key to your clipboard by entering this command:
    - `clip < ~/.ssh/id_rsa.pub`
    - You shouldn't see any kind of message when you run this command. If you do, make sure you entered it correctly.

- Do not copy anything else to your clipboard until you finish the next instructions. Otherwise, you'll have to repeat this step again.

11. Go to https://github.com/settings/ssh. Click the "New SSH Key" button.

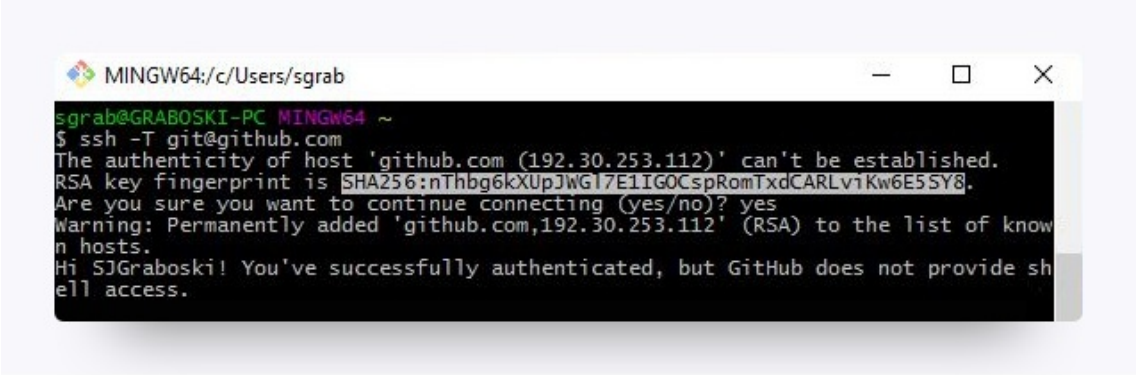


12. When the form pops up, enter a name for your computer in the Title input. In the Key input, paste the SSH key you copied in step 10.



13. Now we just need to add GitHub to your computer's list of acceptable SSH hosts. Go back to your Bash window. Type in this command: `ssh -T git@github.com`

- You should see an RSA fingerprint in your window. Only enter "yes" if it matches the one highlighted in the image below.



Amaze-Balls!

If you got through all the installations, give yourself a pat on the back! Installations are never fun, but just like taxes, ya gotta do them.

Be sure to take a break before continuing with the rest of the pre-work.

Copyright

Assignment #5 - Get Yo' Tools Installed (Mac)

Overview

In this assignment, you will be installing all of the required tools and software necessary for the class. We've got a lot to install so buckle in and get ready!

Before You Begin

Make sure you sign up for these services; you'll need all of them throughout the course.

- LinkedIn: https://www.linkedin.com
- GitHub: https://github.com
- StackOverflow: http://stackoverflow.com

Don't just create logins. Job recruiters often scour these sites in search of job candidates; make sure you provide your headshot and your contact info on all three services.

P.S. Don’t forget to outline your skills and your work experience on LinkedIn.

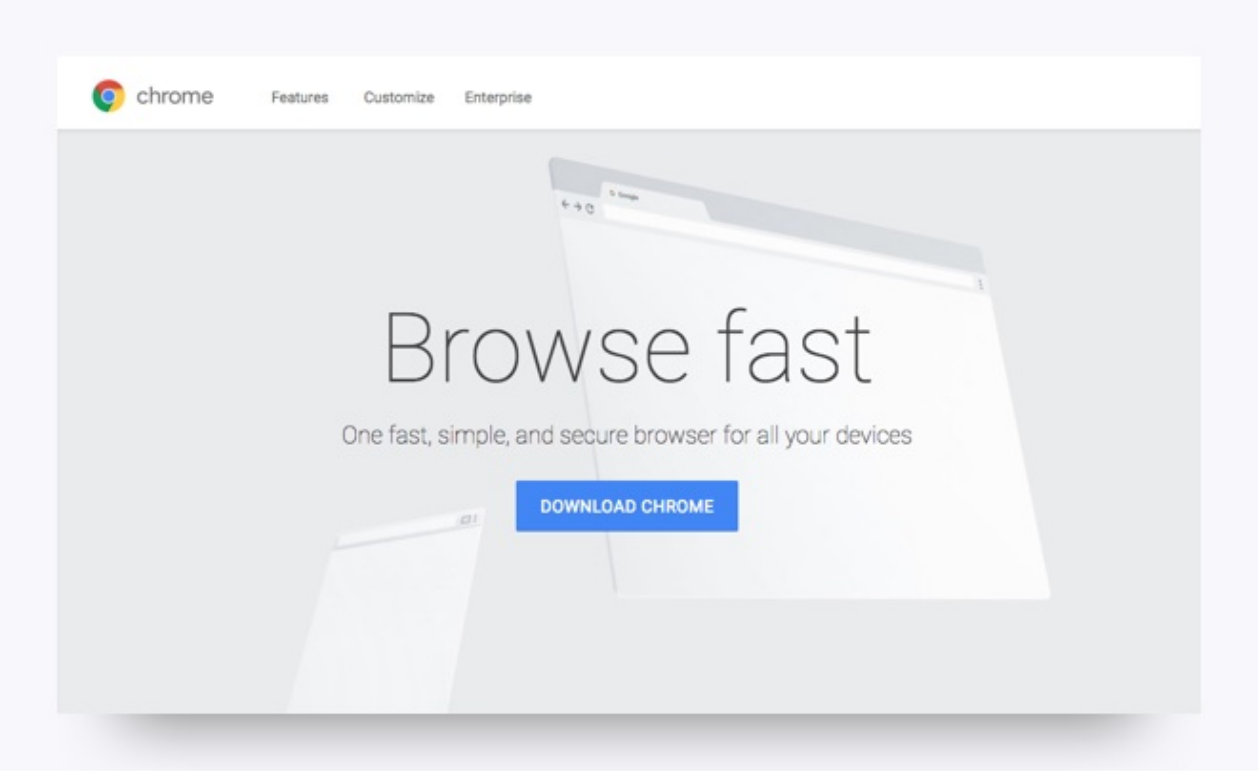
Also, make sure to accept the invite for your section on Slack as well. You will receive the link to your class-specific channel during orientation. If you haven't received an email from us yet with the section's domain name, contact your Student Success Manager to assist you.

Da Big Installation Enchilada

The rest of this assignment will walk you through the specific steps associated with installing each of the tools you'll need. Follow the instructions closely!

Google Chrome

During this course, consider Chrome *the* web browser; it comes loaded with tools for quickly editing the web pages you’ll create.



1. If you don’t already have Chrome installed, visit their download page.
2. Download, open, and run through Chrome’s installation file.

Screencastify

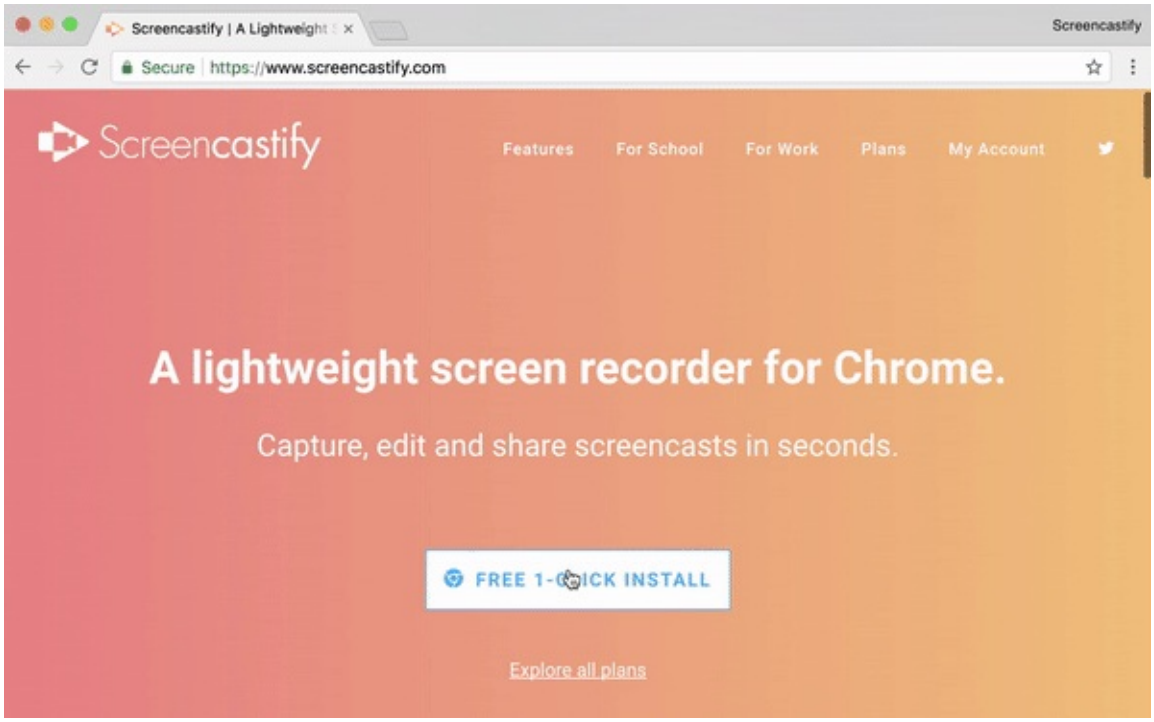
There are two places where you can install the Screencastify extension:

1. The Screencastify Website.
2. The Chrome Web Store.

Note: You must be using Google Chrome to install and use Screencastify

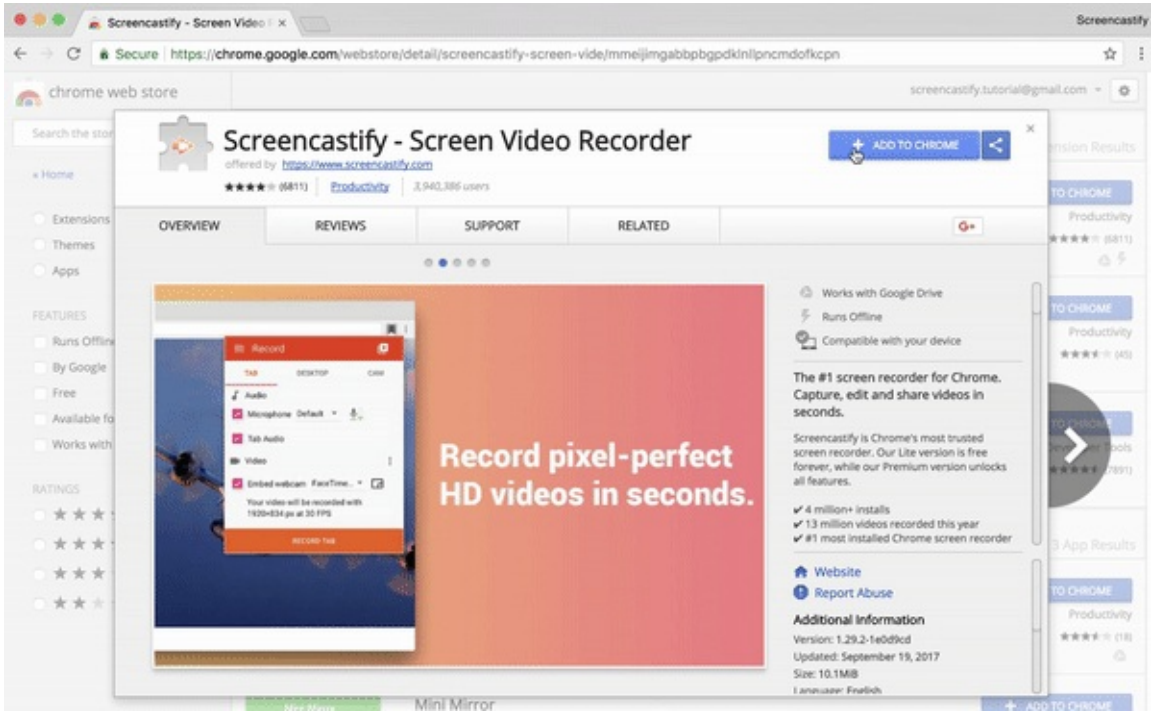
Install Screencastify from their Website:

- Go to the homepage and click the blue "**Free 1-Click Install**" button. You'll see a confirmation box confirming that you want to install the extension. Click "**Add extension**". Screencastify will then begin downloading.

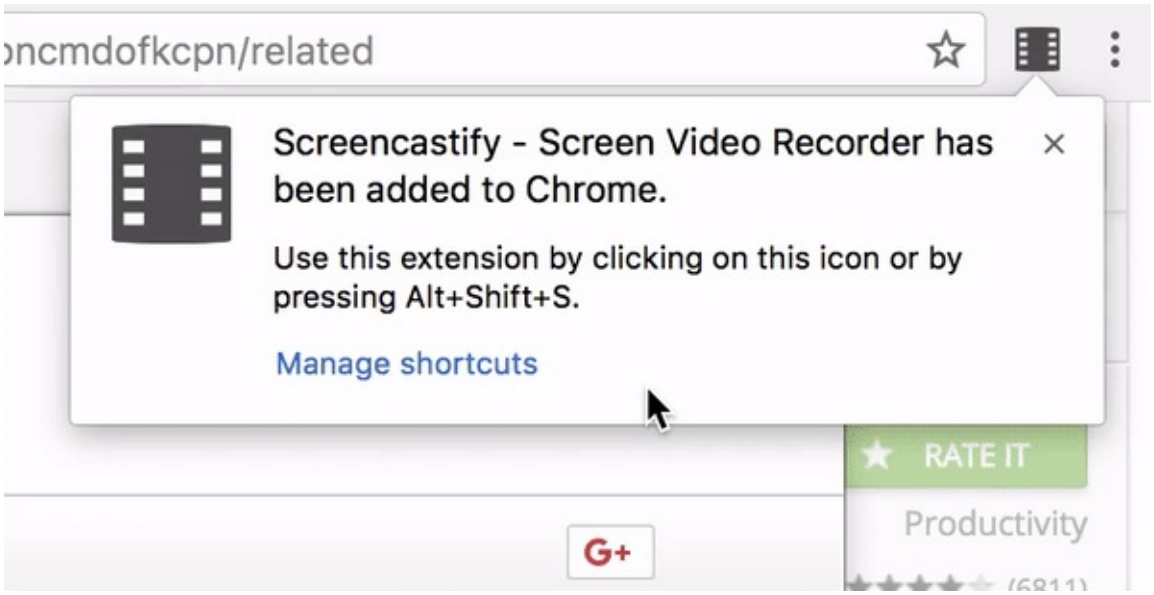


Install Screencastify from the Chrome Web Store:

- Click here to view their listing in the Chrome Web Store, then click "**Add to Chrome**". You'll see a confirmation box confirming that you want to install the extension. Click "**Add extension**". Screencastify will then begin downloading.



- Once Screencastify is finished recording, their grey film strip icon will appear in the top right-hand corner of your browser. Click the icon to get started.

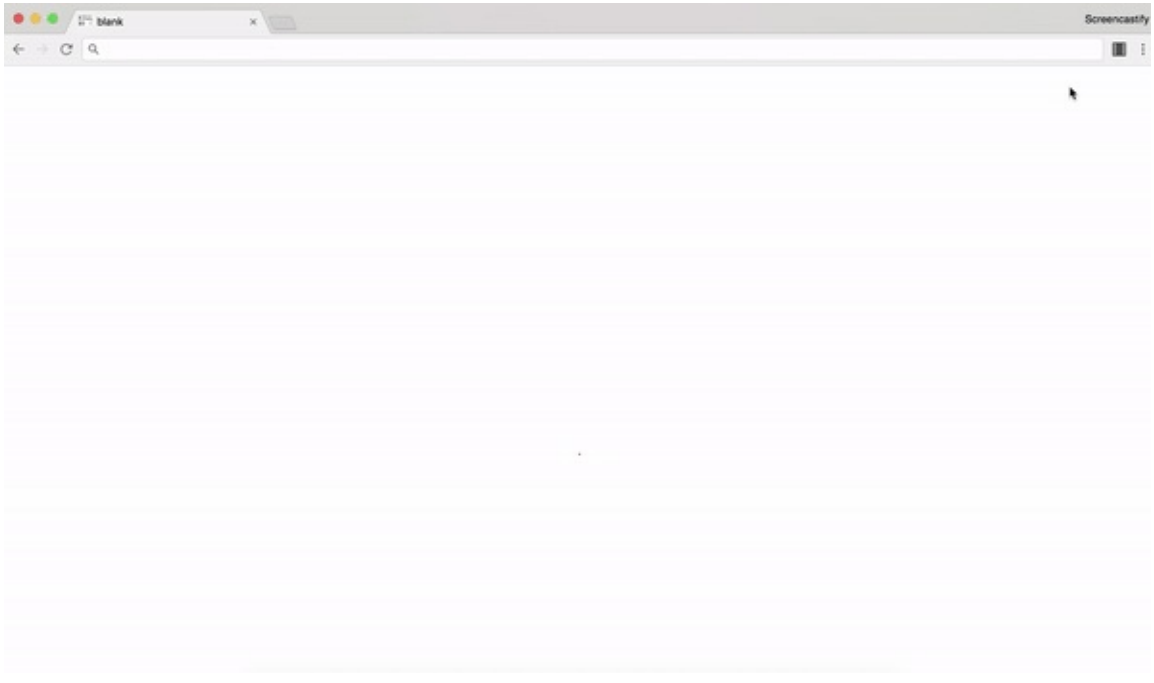


Set up your webcam and microphone:

The first time you click on the Screencastify extension icon, you'll be asked to give Screencastify permission to access your webcam and microphone. They ask for these permissions so that you're able to narrate over your recordings and embed / record your webcam.

Note: While you're required to allow access to your camera and microphone in order for Screencastify to function properly, you don't need to include audio or webcam video in your recordings. Screencastify never records audio or video unless you expressly ask it to.

- When you click on the grey film strip icon, you'll be taken to a page where you can allow camera and microphone access.
- Click the "**Setup Camera Access**" button. Chrome will confirm once more that you'd like to allow microphone and camera access. Click "**allow**".



- And that's it! From now on, you'll be able to narrate over your recordings and record your webcam whenever you'd like.

Troubleshooting: If Screencastify is unable to detect your microphone, click here. If it's unable to detect your webcam, click here.

Connect your Google Drive to Screencastify:

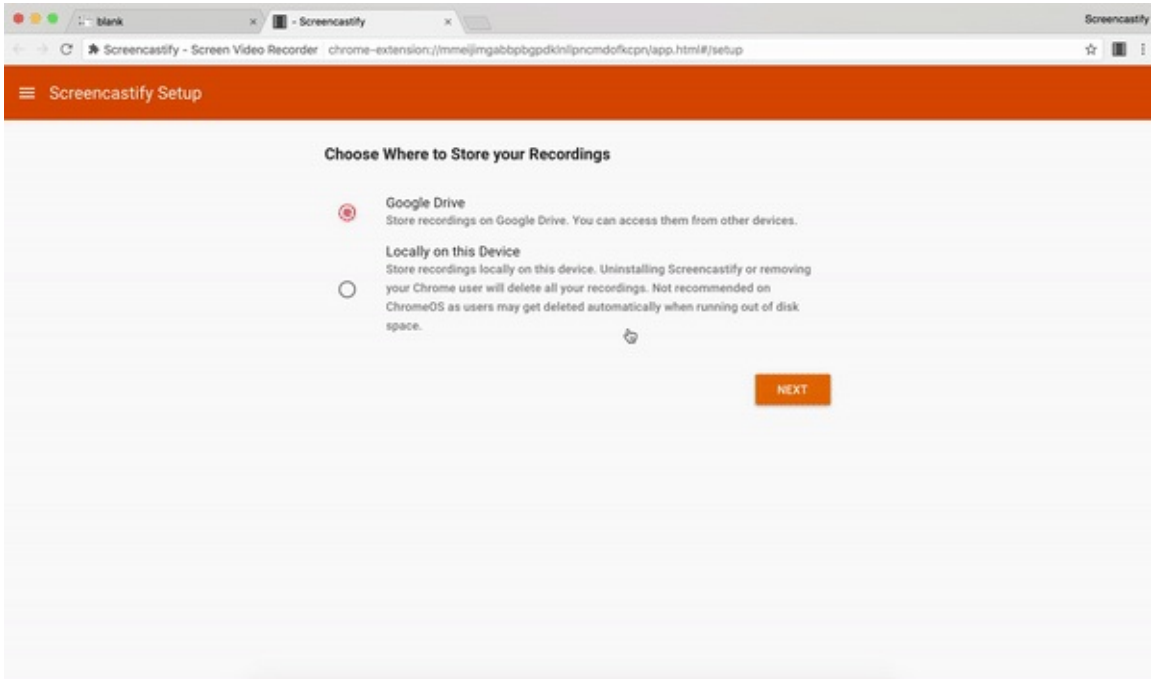
When you connect your Google Drive to Screencastify, all of your recordings will automatically save to your Google Drive in a folder titled " Screencastify". Google Drive, along with many other useful apps come free with a Google Account. If you don't already have one, go create one!

Depending on how long a particular recording is, this process could take a few minutes to complete after you end a recording.

Note: Screencastify highly recommends connecting your Google Drive to Screencastify and saving all of your recordings there vs. locally on your computer. It's much safer.

The first time you open the Screencastify extension, you'll be asked where you want to store your recordings (right after you allow camera and microphone access).

- Make sure that "**Google Drive**" is selected, then click "**Next**". You'll then need to sign in with your Google account and allow Screencastify permission to access your Google Drive. You can sign in with **any** Google-based account, not just a @gmail.com email.



Now, all of your videos will automatically save to your Drive after you finish recording.

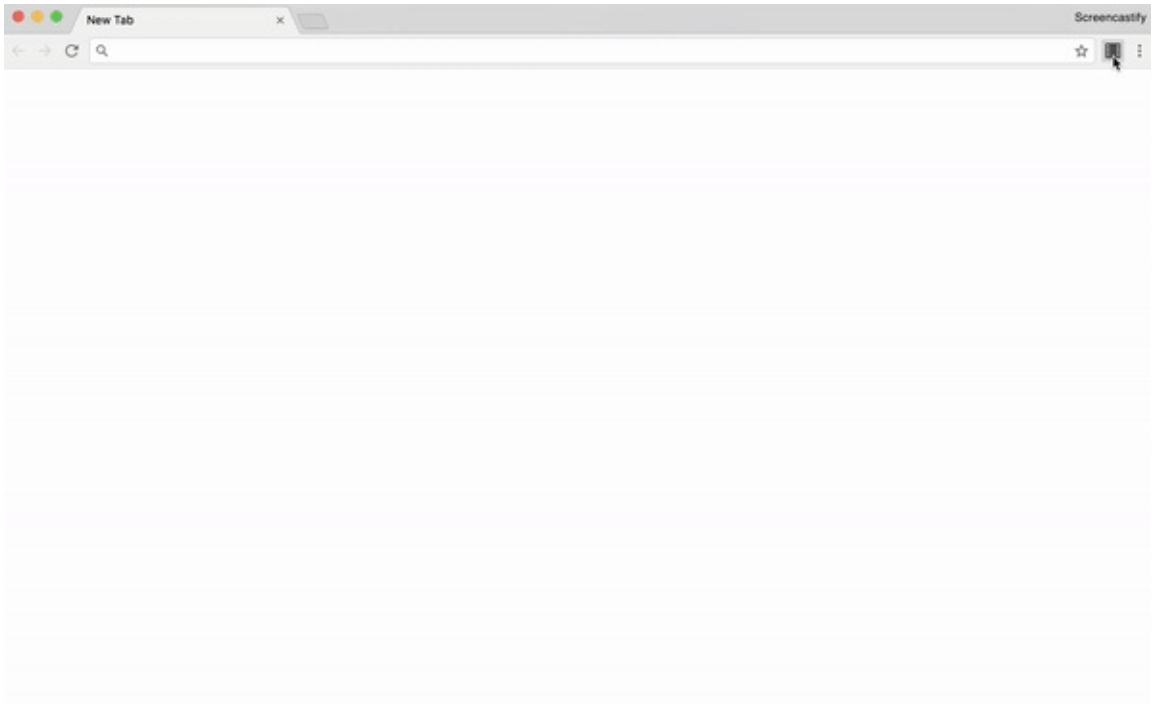
As soon as you finish your first recording, a new " Screencastify" folder will be created in your Drive. All of your recordings will be saved in this folder.

The videos you will see in Your Recordings library in Screencastify are synced with this "Screencastify" folder in your Drive. This means:

- If you delete a video in Screencastify, it will also be deleted from your Drive
- If you move a video out of the "Screencastify" folder in your Drive, it will no longer be accessible from Your Recordings in Screencastify
- If you rename a video in Screencastify, it will also be renamed in your Drive

Note: Screencastify never views, modifies, stores, or in any way interacts with the other files in your Google Drive.

You can always change the default storage location for your recordings, even after initial setup. Simply open the extension menu and click "**Options**". The first thing you'll see is an option to select where you want to store your recordings.

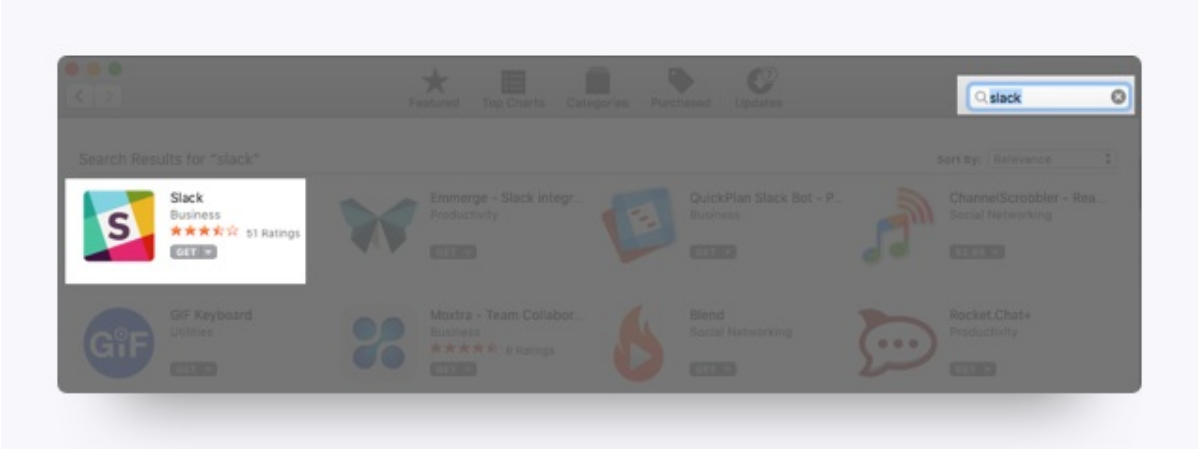


Important Resource: Want to know how to record, save to your Google Drive and share your recordings? Check it out here!

Slack

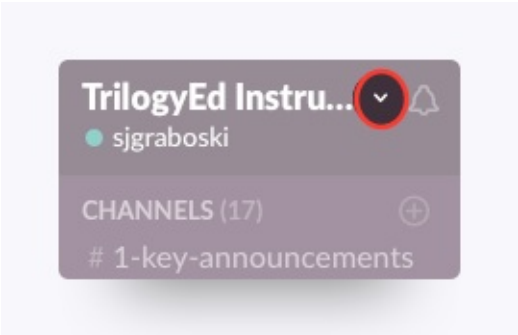
You'll be messaging your instructor, your TAs, and your fellow classmates with this business-centric chatting app. The teaching and career staff will post some of their most important announcements here, so set this program up as soon as you can. You will receive the link to your class-specific channel during orientation.

1. **If you don't have the Slack app yet,** search for Slack in your Mac's App store, and then click the Get button under the app's listing. Click the button again when it displays "install."

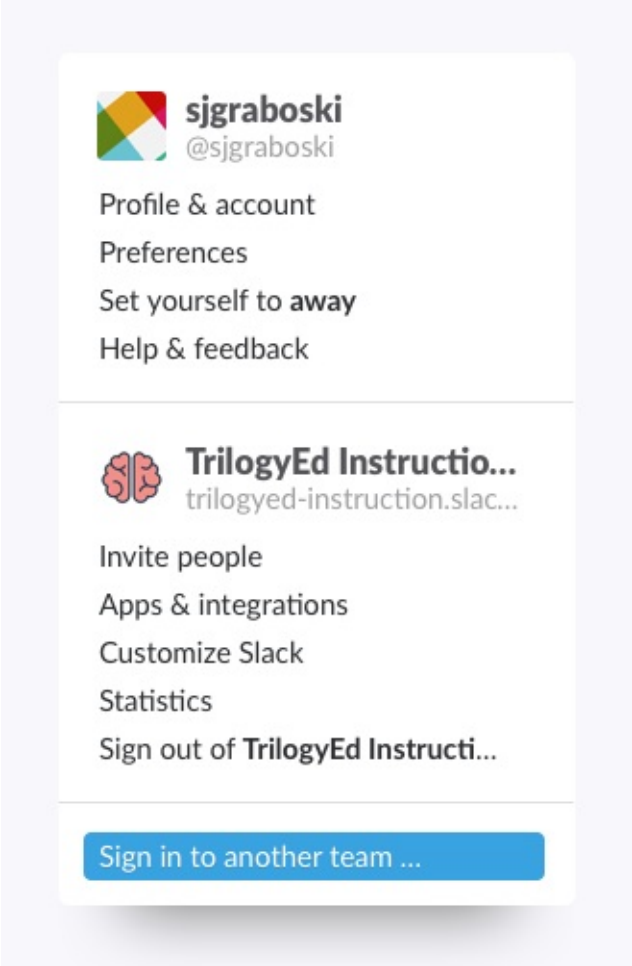


- When the app finishes installing, open it and move on to step 2.

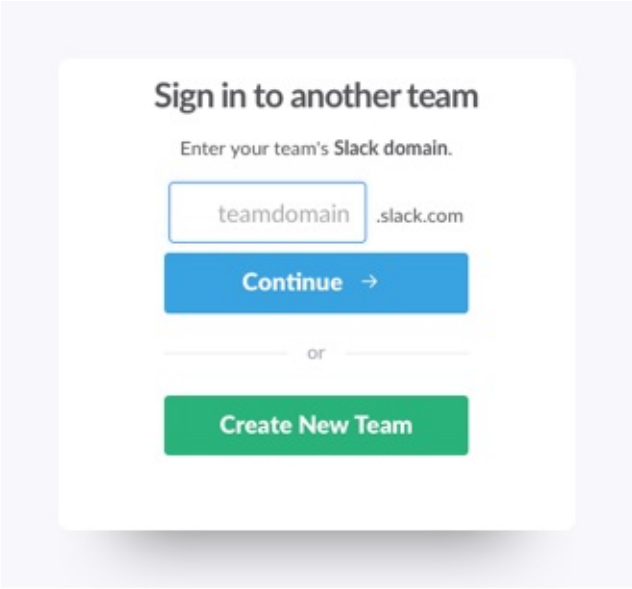
2. **If you already use the Slack app,** you just need to add our channel to your application.
   - Click the header of your current Slack Channel.



   - Then select “Sign in to another team …”



3. As you run through the guide, make sure you do the following:
    - Enter in the domain we gave you for Slack.

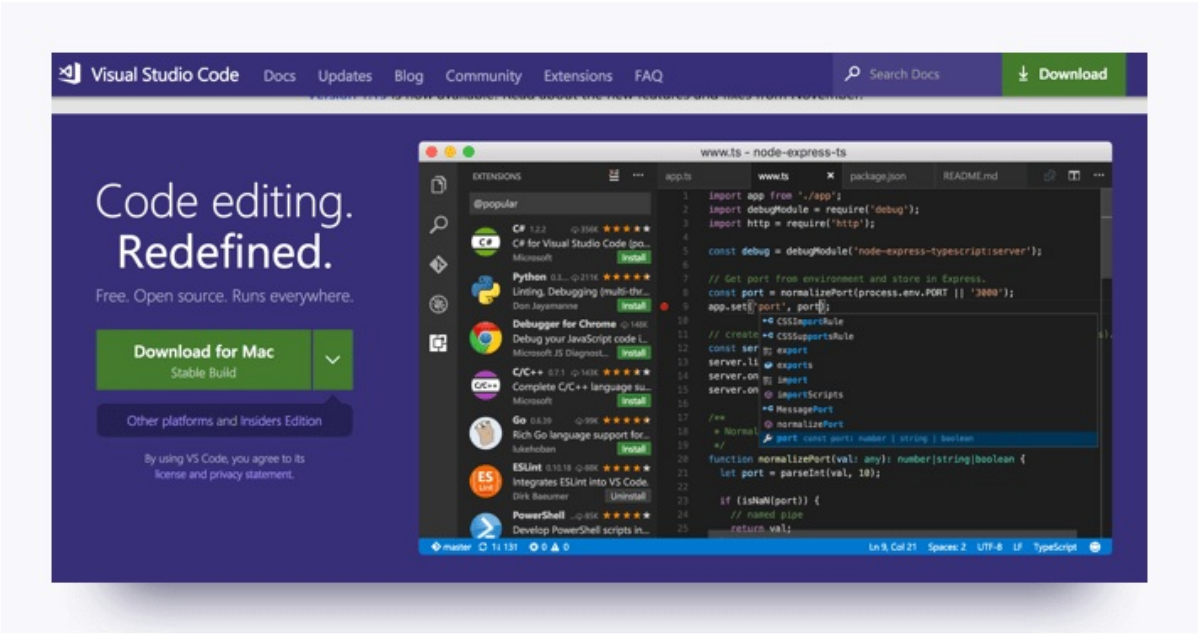


    - Enter in the email with which we invited you, as well as your password, when prompted.

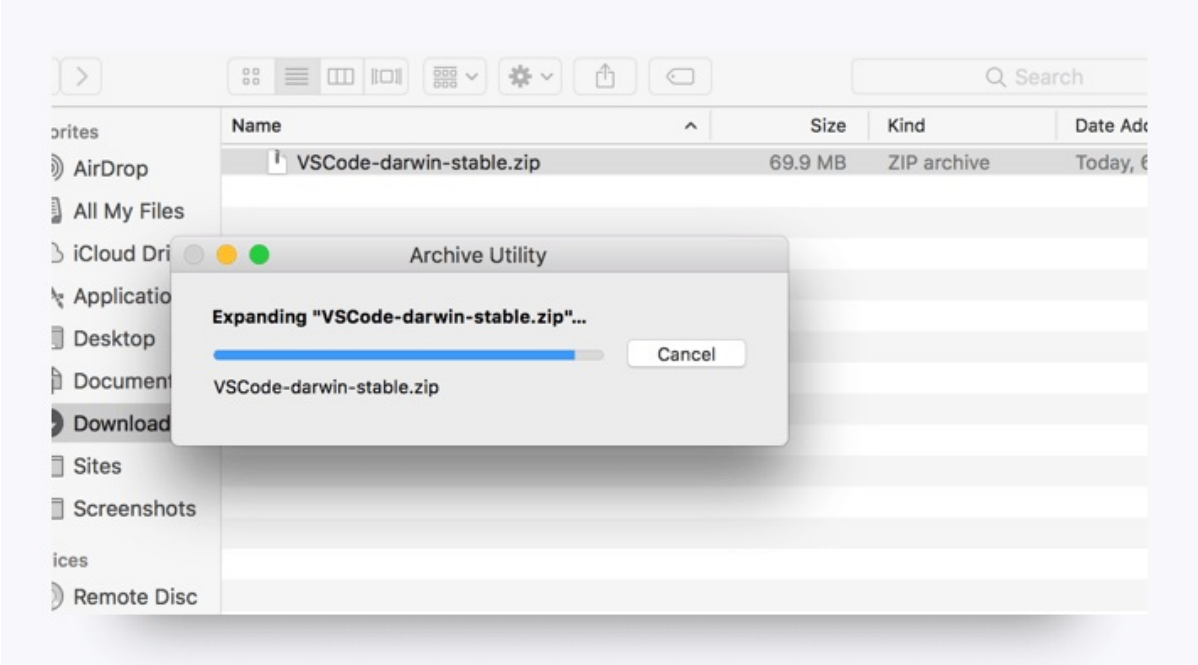
Visual Studio Code

This text editor features a variety of indispensable tools and customizable components that will help you code your websites and apps.

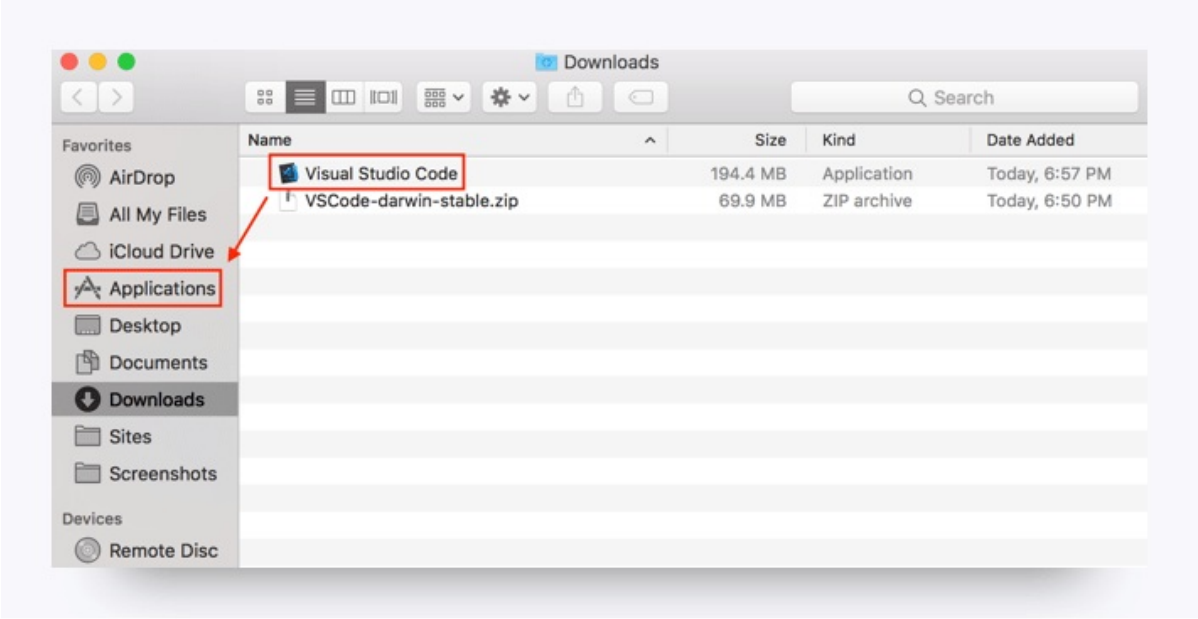
1. Download the appropriate version for your operating system.



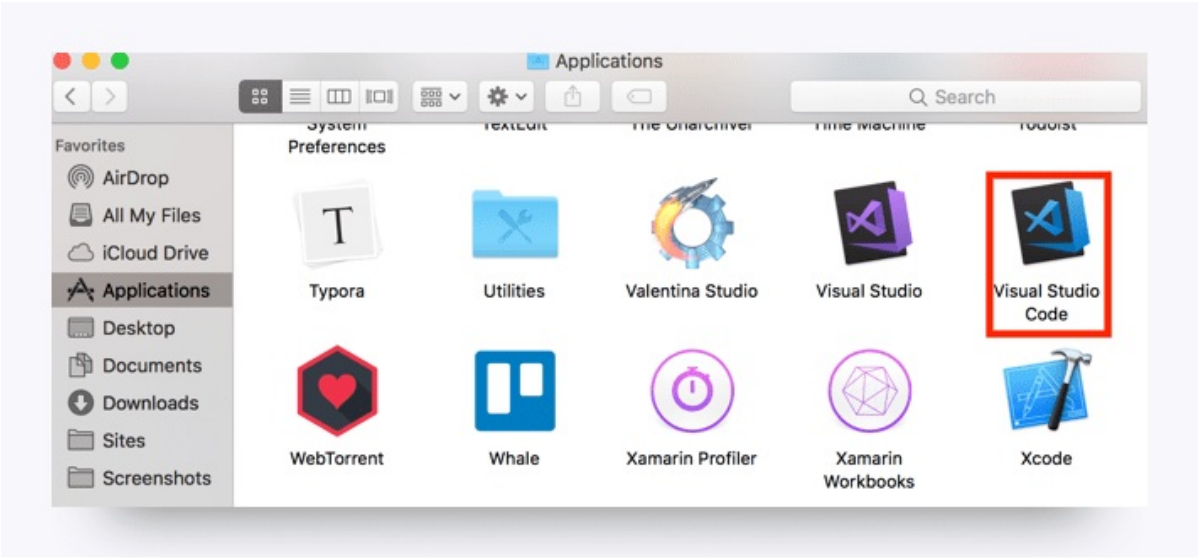
2. Unzip the file.



3. Drag the resulting file to your Applications folder.



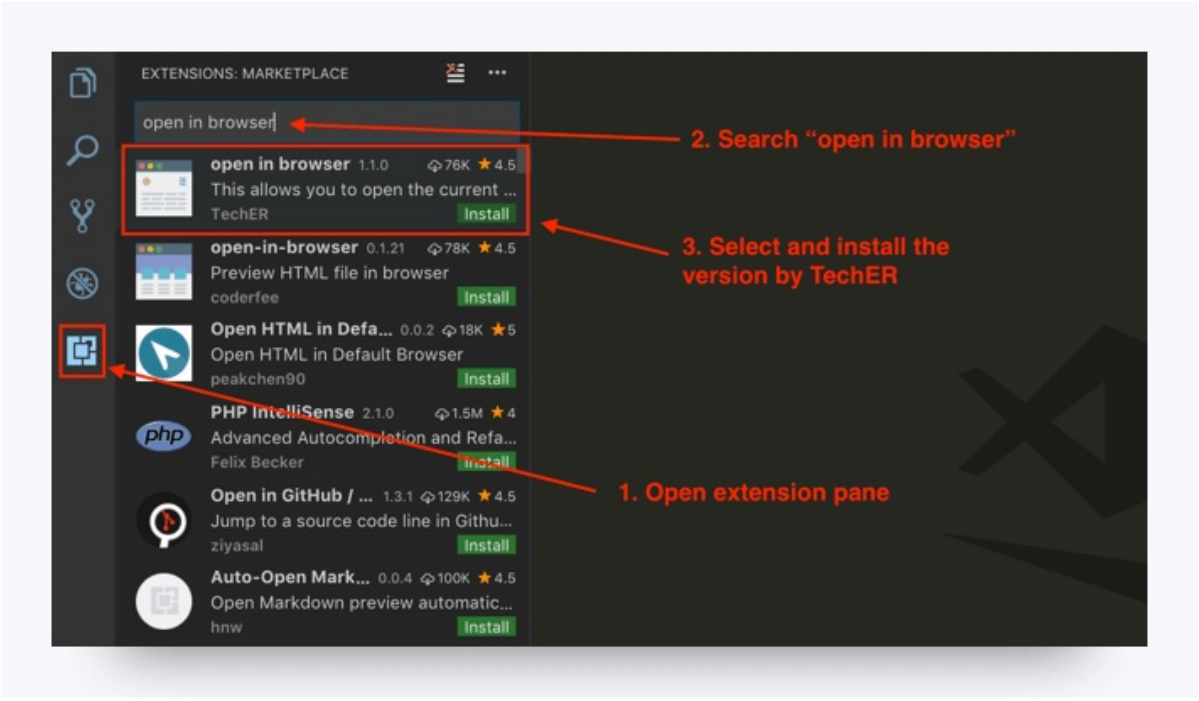
4. You should now be able to access Visual Studio Code from your Applications folder.



Install "Open in Browser" Extension

With this extension, you can open HTML files in your web browser from the Visual Studio Code editor.

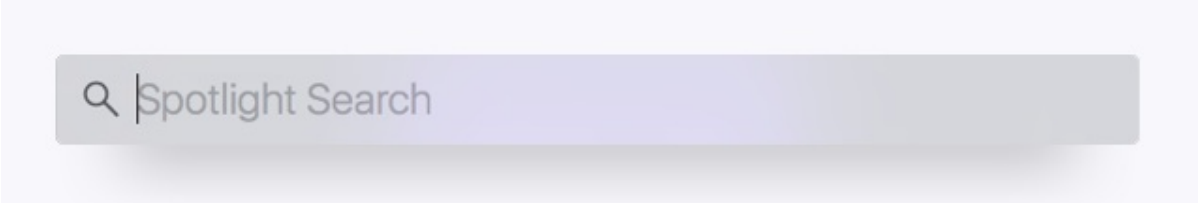
1. Open Visual Studio Code.
2. Open the extensions pane and search for "Open in Browser".
3. Select the version written by "TechER" and install.



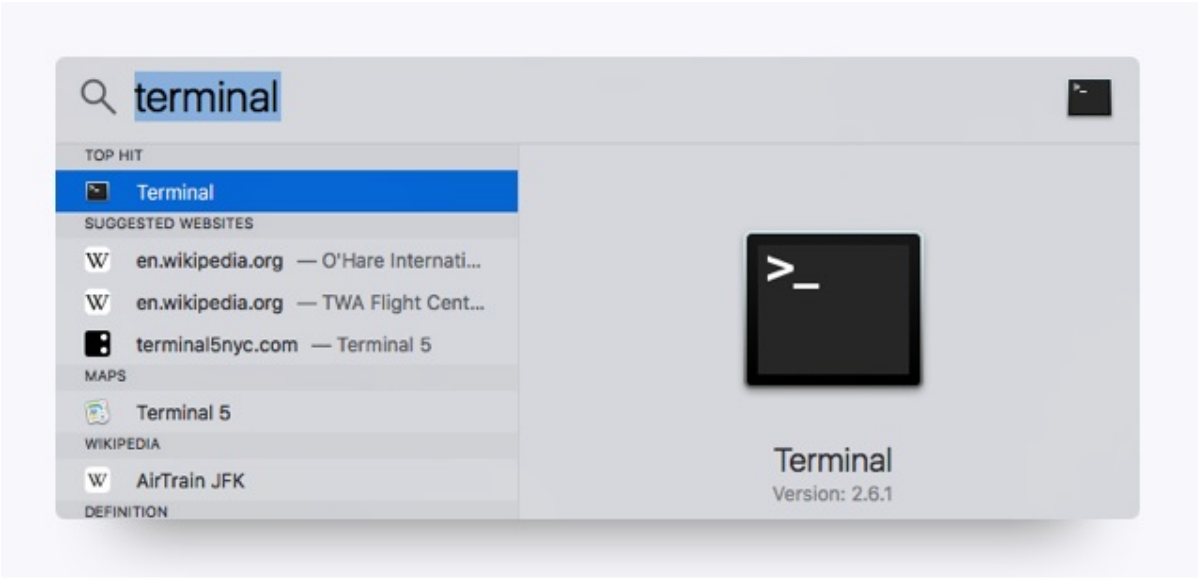
Terminal

You'll be entering your command line code through this interface. Since you're on a Mac right now, you already have it! Just follow these steps to open the program.

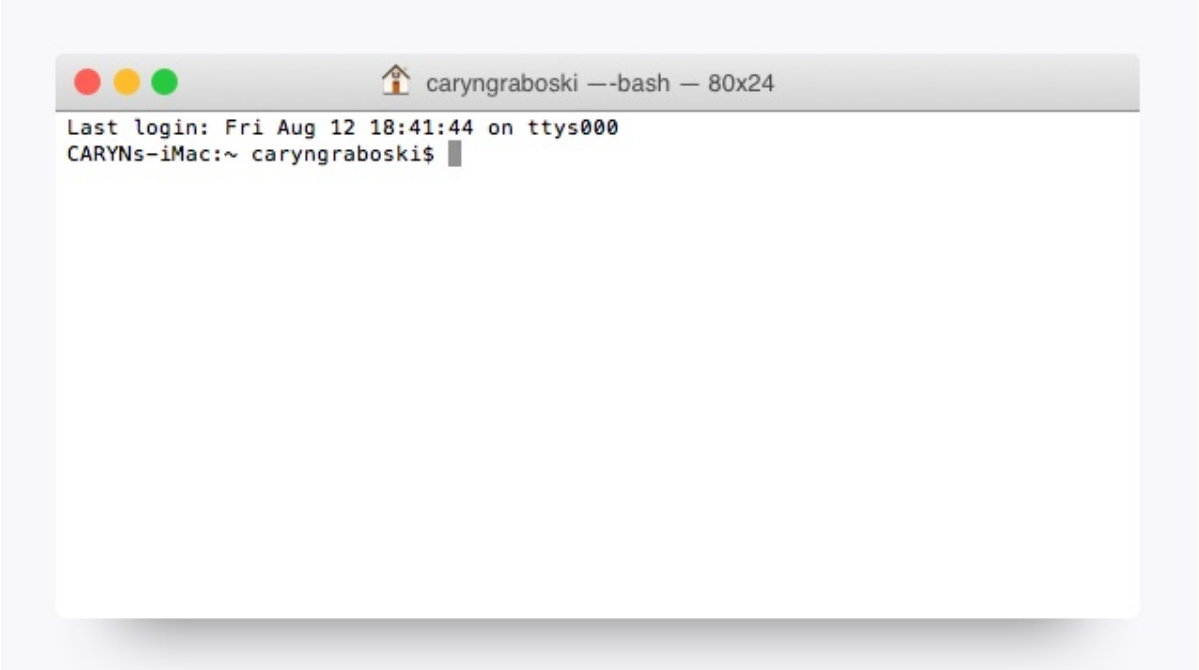
1. Press `command` + `spacebar` to open up Spotlight Search.



2. Type “Terminal” into the search, and then hit enter.



3. Keep this window open; you’ll need it for the next steps.



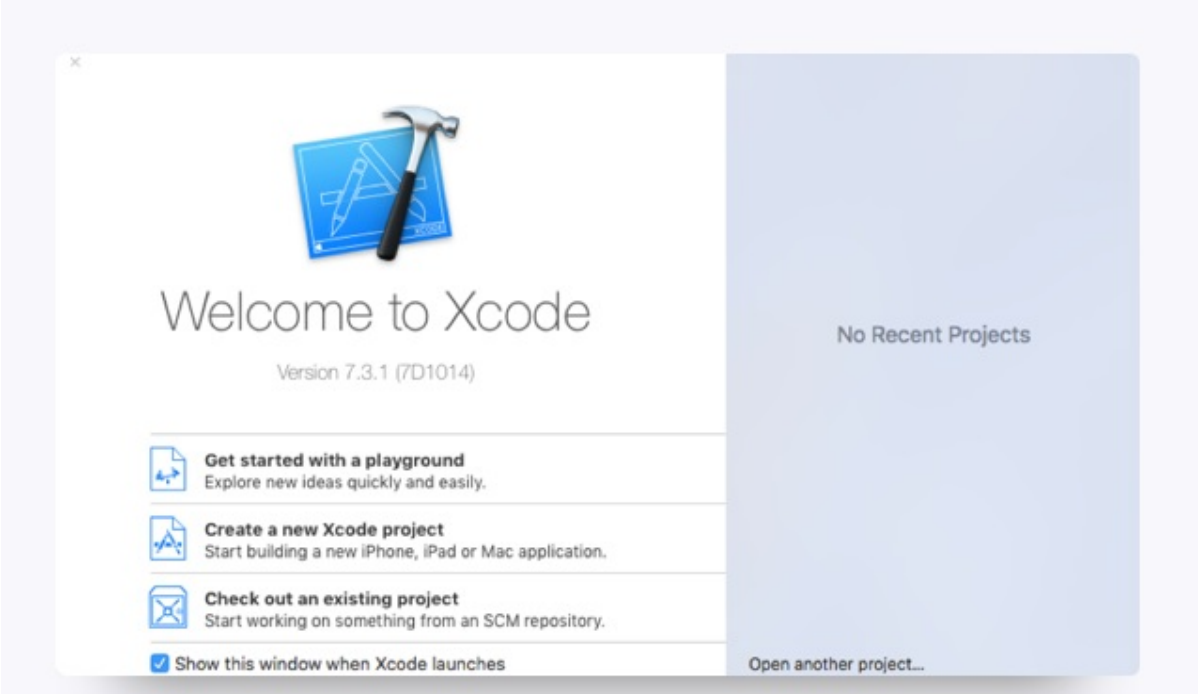
Xcode

Xcode is a development suite exclusive to the Mac. While we're going to use Visual Studio Code in this class, installing Xcode will set up some of the other required bootcamp programs. This includes the always-handy Git—coders depend on this tool for logging the development of programs and applications.

1. Open up the App Store on your Mac machine, and then search for Xcode. Download and install it.

2. Once Xcode finishes downloading, open the program. Agree to the terms of service and wait for the remaining components to install.

3. When you see the window below, you're done! Move onto the Heroku Toolbelt.

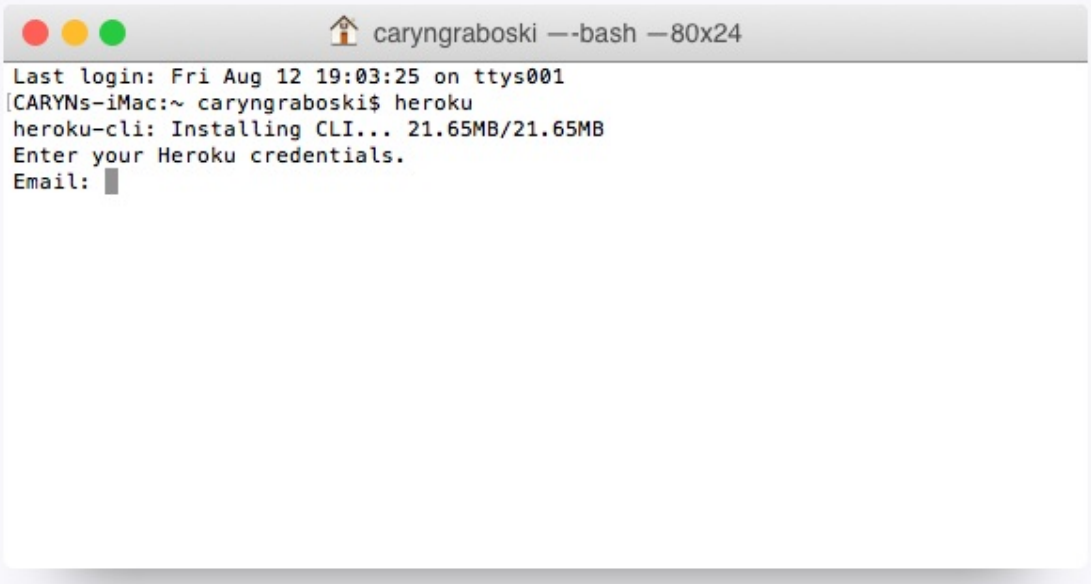


Heroku Toolbelt

This tool lets developers deploy their web apps to the cloud, allowing anyone with the right addresses to access their creations.

1. First, you need to sign up for a free Heroku account: https://signup.heroku.com/.
2. Then go to https://toolbelt.heroku.com. Download the installer.

3. Go through the install guide. Then open Terminal. Type `heroku` into the command line, and then press `return`. If `heroku` doesn't work for you, type `heroku login` instead.



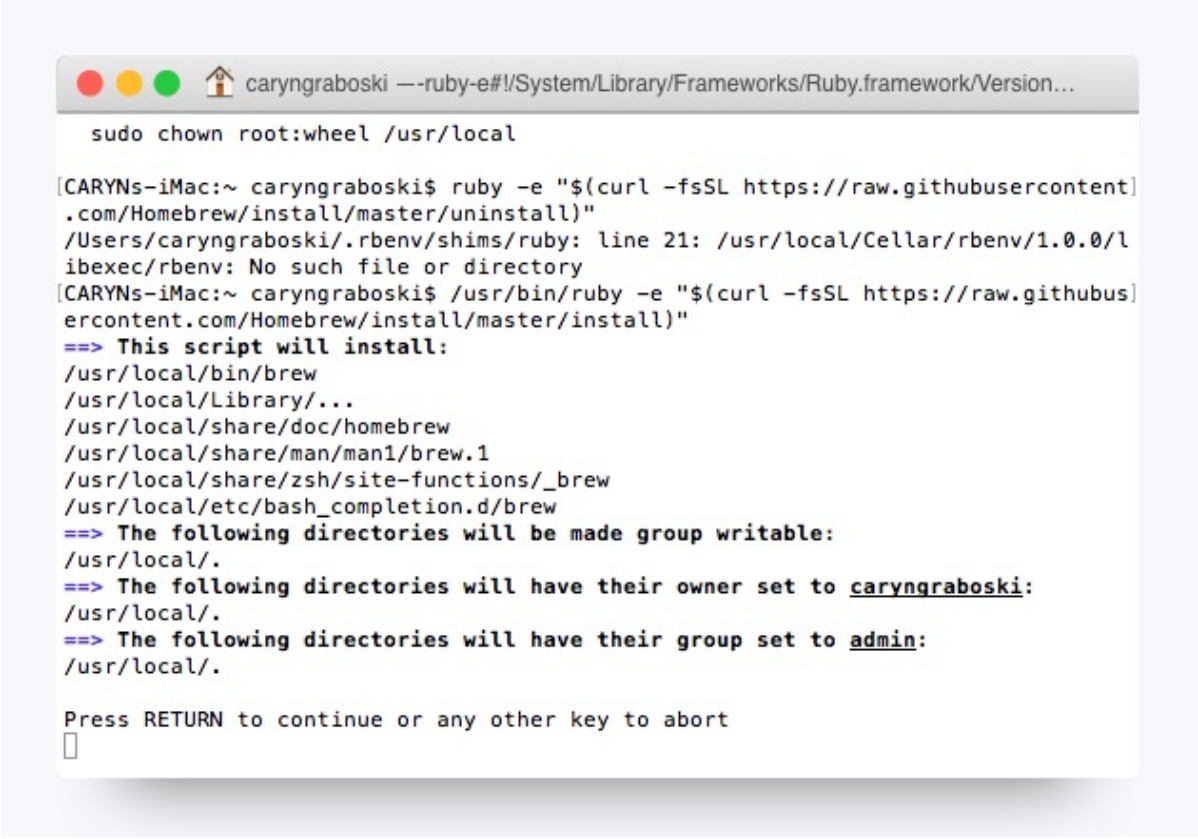
4. When prompted, enter the credentials you used when you signed up for your Heroku account.

Homebrew

This tool makes it a cinch to install new programs and libraries in your Terminal window.

1. Go to http://brew.sh. Copy the script listed under “Install Homebrew.”

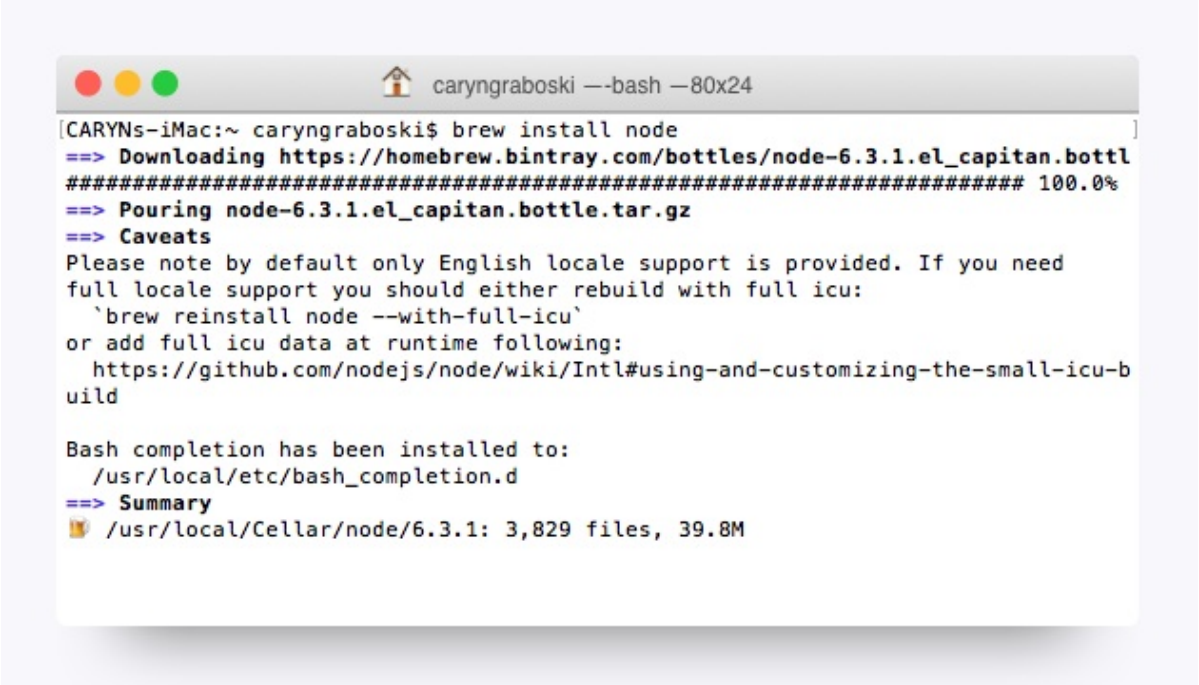
2. Paste the script into your Terminal window. Press the `return` key when prompted.



Node.js:

This runtime environment has quickly become a standard for coding back-end programs. Your code will run on Node throughout most of the course.

1. Just run this command in your Terminal window: `brew install node`

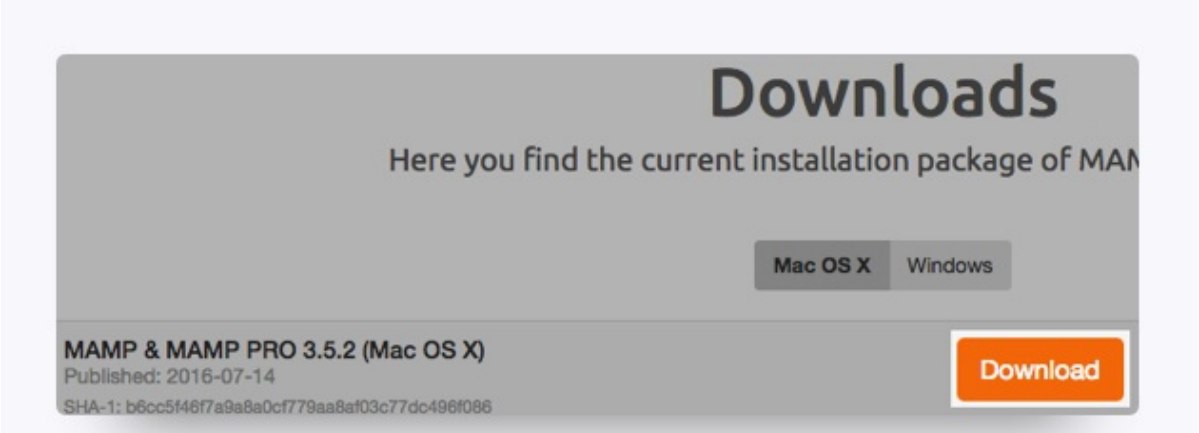


   - Note: make sure you have brew installed before you run this.
2. Type `node -v` in Terminal, and hit `return`. If Terminal returns a version number, then you've successfully installed Node.

MAMP:

This tool lets developers run local servers for their back-end-reliant web apps. In other words, you can power some of your more complex sites without an internet connection.

1. Go to https://www.mamp.info/en/downloads/. Select Mac and click download.



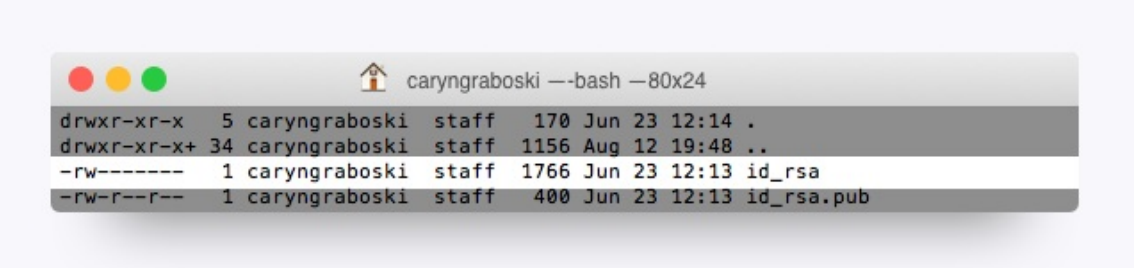
2. Run through the installation.

SSH Key

Generating SSH keys allows developers to interface with certain remote services without having to constantly type out login information. You're going to set up an SSH key for GitHub.

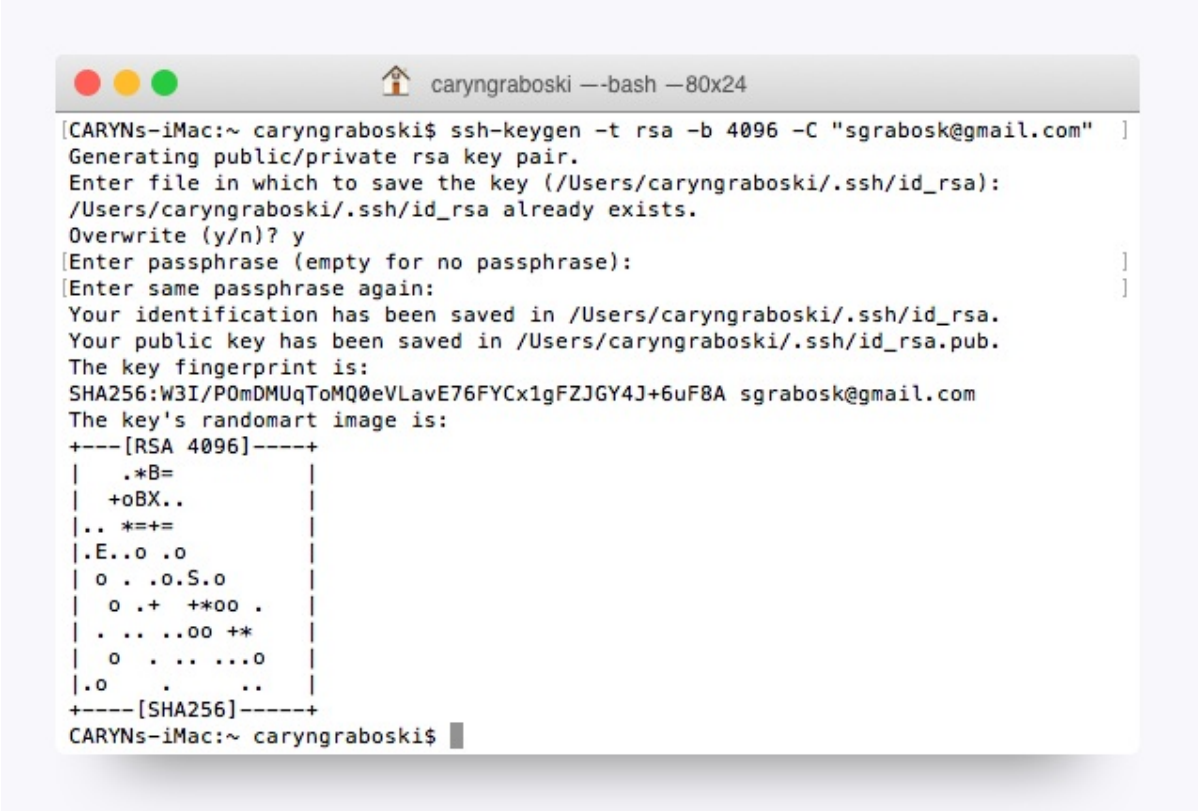
Without a key, you won't be able to push your code to GitHub without entering a password each time; trust us, that would be as irritating as needing a key to open every door in your home.

1. Sign up for an account on https://github.com.
2. Open up Terminal.
3. We need to set up SSH keys. First, let's make sure you don't already have a set of keys on your computer. Type this into your Terminal window **(copying and pasting will not work)**:
    - `ls -al ~/.ssh`
    - If no keys pop up, move on to step 4.
    - If keys do pop up, check that none of them are listed under `id_rsa`, like in this image:

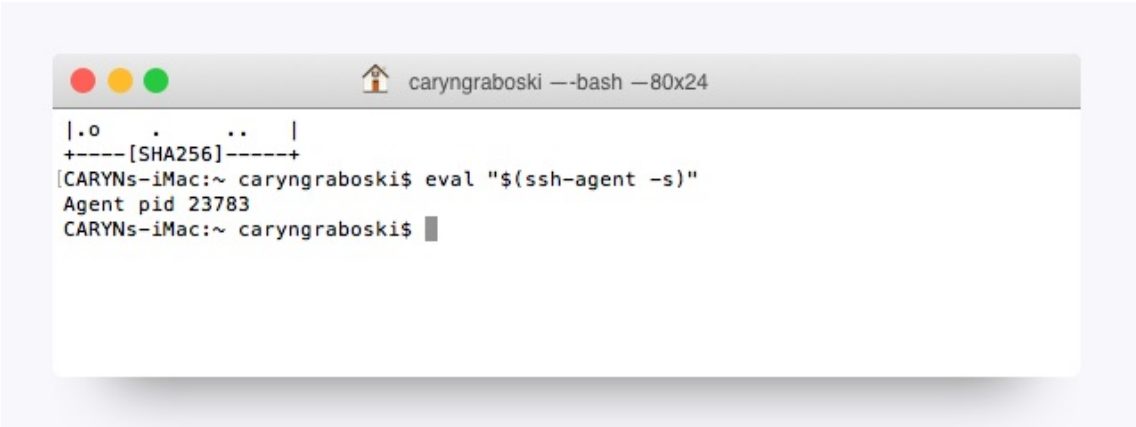


        - If you do find a key with a matching name, then you can either overwrite it by following steps 4 to 6, or you can use the same key in steps 10 and beyond. Be advised that you'll have to remember the password tied to your key if you decide not to overwrite it.
4. Type in this command along with your email to generate your keys:
    - `ssh-keygen -t rsa -b 4096 -C "YOURGITHUBEMAIL@PLACEHOLDER.NET"`
5. When asked to enter a file to save the key, just hit `return`.
    - Also enter a passphrase for your key.
    - Note: You shouldn't see any characters appear in the window while typing the password.

6. When you're finished, your window should look like this:

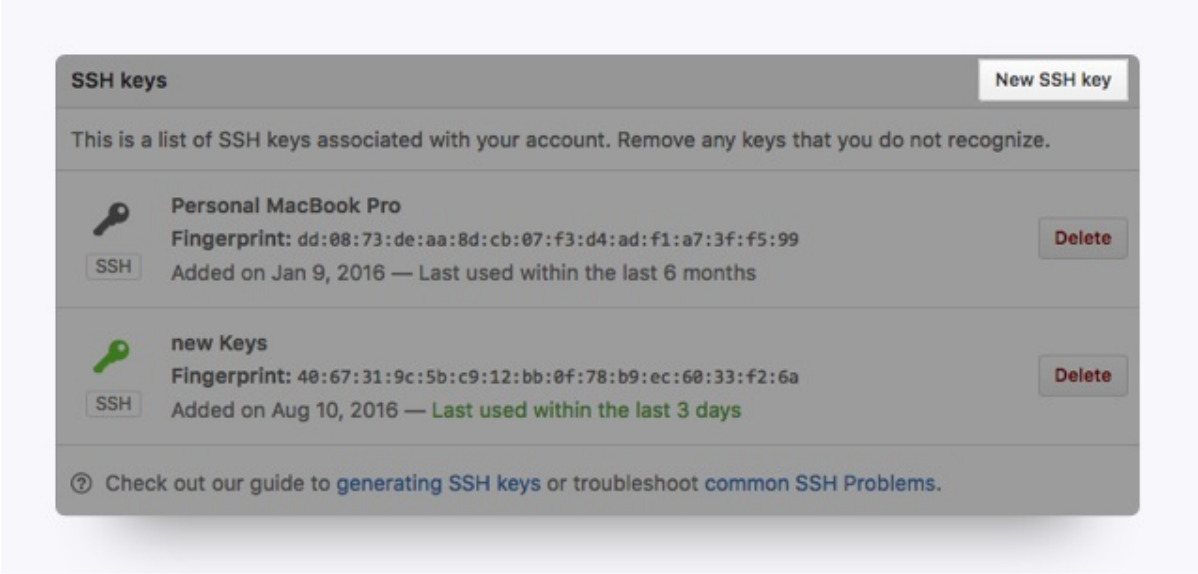


7. For the next step, we need to use a tool called an ssh agent. Let's test whether that's working on your machine. Run this command in Terminal:
   - `eval "$(ssh-agent -s)"`
   - If your Terminal window looks like the image below, move onto the next step.

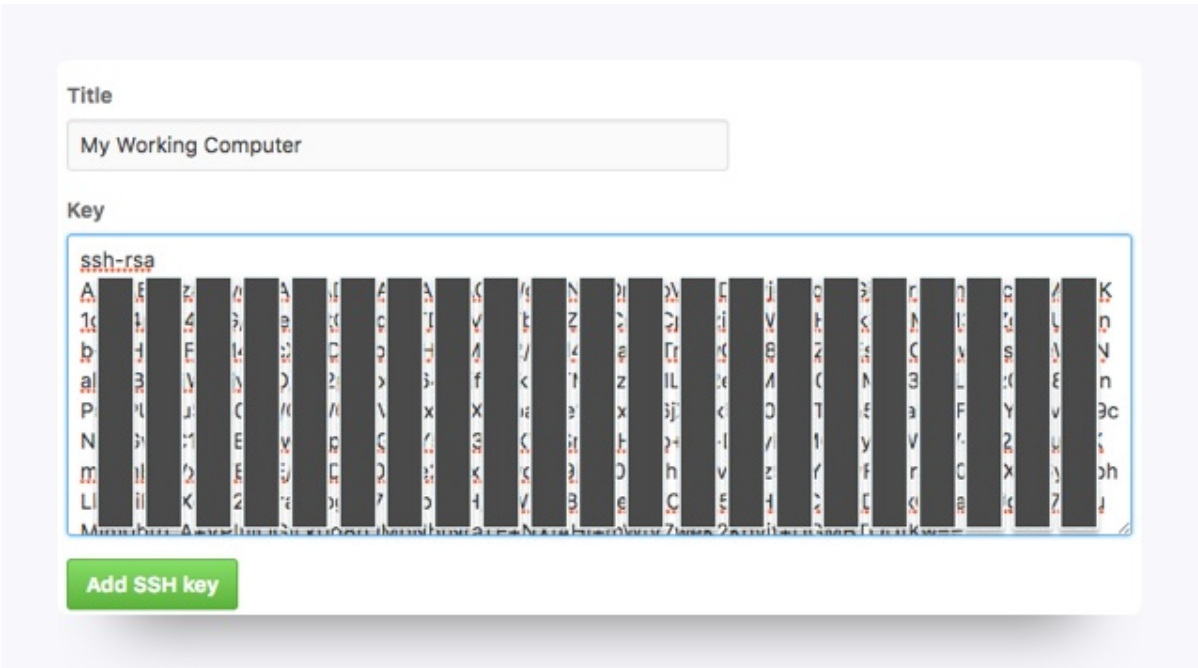


8. Now run this command:
   - `ssh-add ~/.ssh/id_rsa`
9. When prompted for a passphrase, enter the one associated with the key.
   - If you've forgotten this key, just go through step 4 to create a new one.

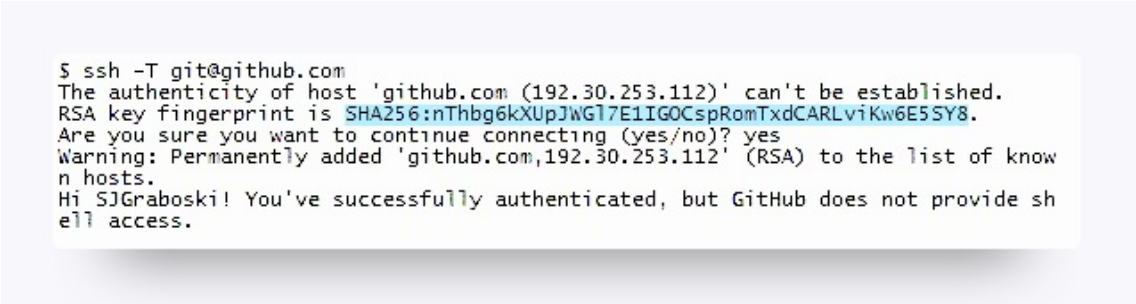
10. We need to add the key to GitHub. Copy the key to your clipboard by entering this command:
    - `pbcopy < ~/.ssh/id_rsa.pub`
    - You shouldn't see any kind of message when you run this command. If you do, make sure you entered it correctly.
    - Do not copy anything else until you finish the next steps. Otherwise, you'll have to enter the copy command again.
11. Go to https://GitHub.com/settings/ssh. Click the "New SSH key button."



12. When the form pops up, enter a name for your computer in the Title input. In the Key input, paste the SSH key you copied in Step 10.



13. Now we just need to add GitHub to your computer's list of acceptable SSH hosts. Go back to your Terminal window. Type in this command: `ssh -T git@github.com`
    - You should see an RSA fingerprint in your window. Only enter "yes" If it matches the one highlighted in the image below:



Amaze-Balls!

If you got through all the installations, give yourself a pat on the back! Installations are never fun, but just like taxes, ya gotta do them.

Be sure to take a break before continuing with the rest of the pre-work.

Copyright

Screencastify - Record, Save to your Google Drive and Share Recordings

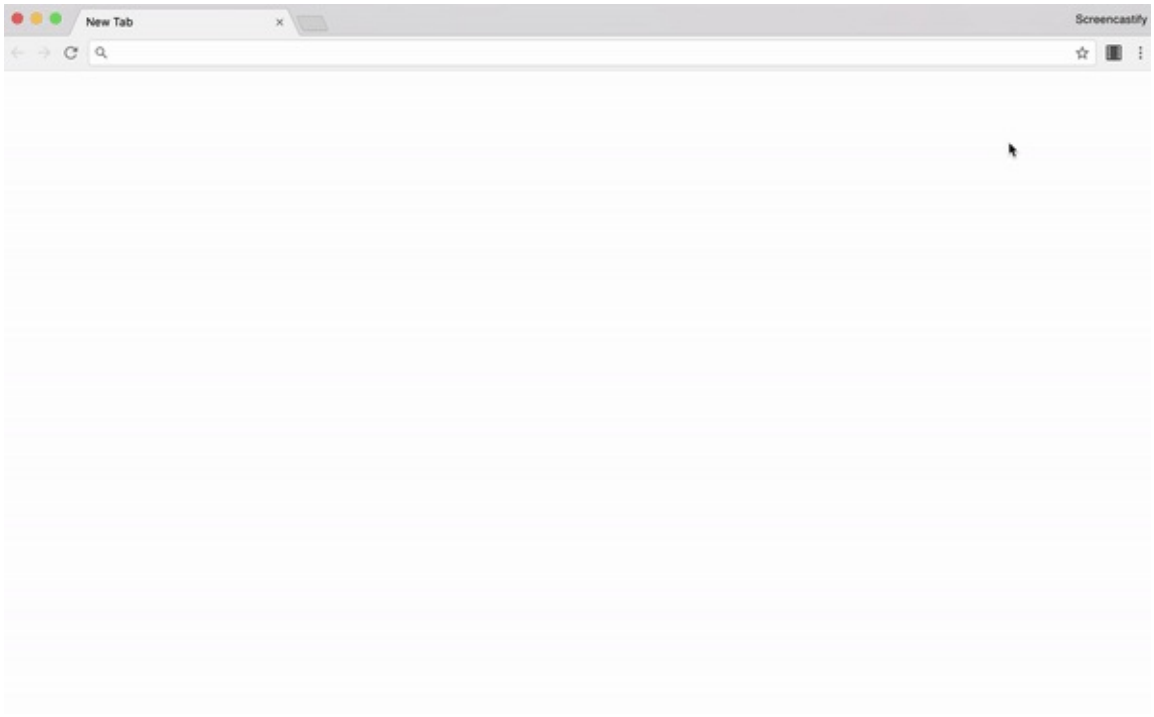
Before You Begin

Before doing this, you should have already installed Google Chrome, installed the Screencastify Extension and created a Google account. You will be saving recording to your Google Drive - a free app that comes with a Google account. If you haven't installed Chrome or the Extension yet, head over to the Machine Ready section, choose the installs link for your machine at the bottom of the page and get them installed.

Starting A Recording

1. Click the Screencastify extension icon.
2. Choose what you want to record: your browser tab, entire desktop, or webcam.
3. To narrate over your recording using an internal or external microphone, check the box next to Microphone.
4. To capture the audio originating from within your browser tab or application (e.g. from a video being played), check the box next to Tab Audio or System Audio.
5. Embed your webcam (optional - tab and desktop recordings only).
6. Click **RECORD**. You'll see and hear a countdown, and then your recording will begin.

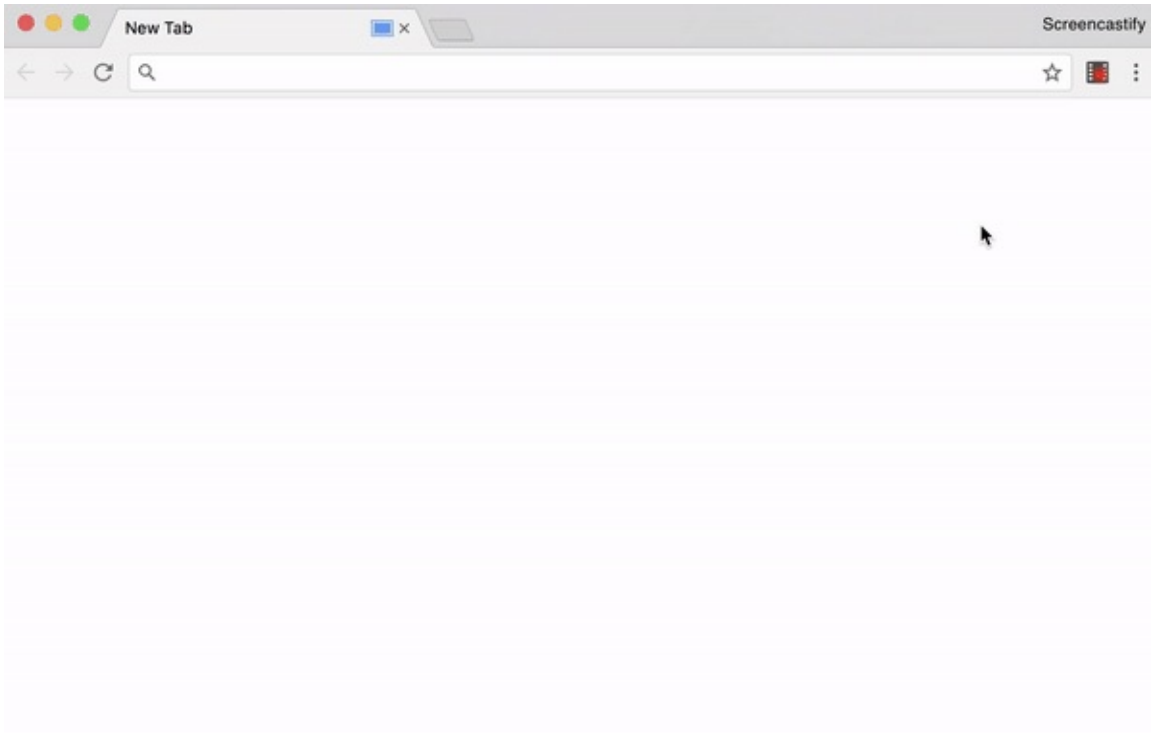
Note: *When Screencastify is recording, a red dot will appear in your extension icon.*



Ending A Recording

1. To end a recording, click the extension icon again (or use a keyboard shortcut) and click **END RECORDING**.
2. When you end a recording, you will automatically be taken to the recording's Video Page, where you can edit, save and share your recording.

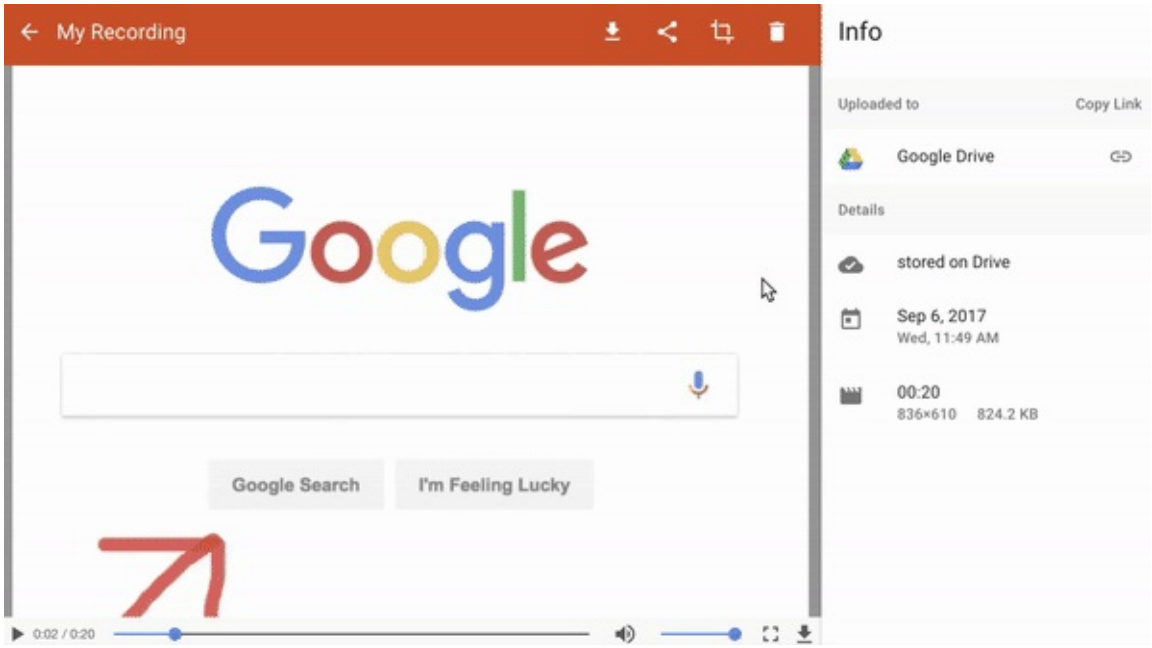
Tip: You can also pause and resume a recording.



Save a Recording to your Google Drive

After connecting Screencastify to your Google Drive (highly recommended and you should have already done this in the Machine Ready installs section), every recording you make will automatically be saved to your Google Drive in a folder called "Screencastify".

You can confirm that your recording has successfully been saved to your Google Drive in the right sidebar of the Video Page.



This process could take a few minutes, depending on how long your recording is.

Important: if you delete a recording in Screencastify, it will also be deleted from your Drive. If you move a video out of the "Screencastify" folder in your Drive, you will no longer see it in Your Recordings in Screencastify.

Note: Your recording may appear blurry in your Google Drive immediately after uploading because Google processes the low-res version first. The full-resolution version of your video will be available within 15 minutes.

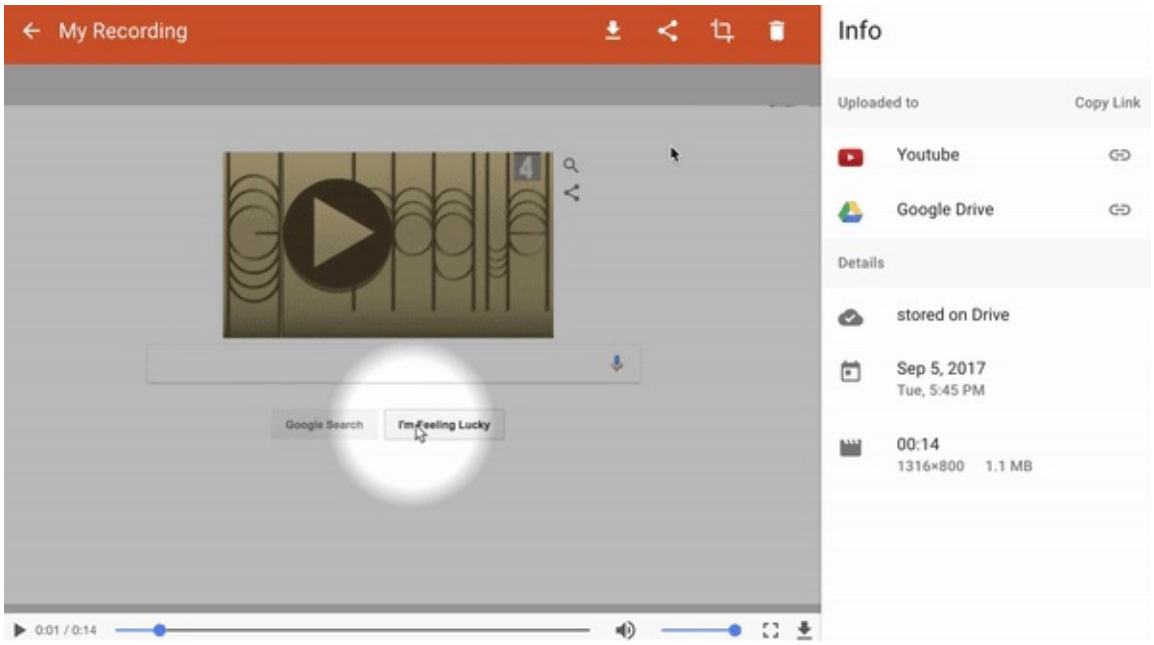
Share a Recording with a Google Drive Link

If your Screencastify account is connected to your Google Drive (highly recommended and you should have already done this in the Machine Ready installs section), you can grab a link to your recording and share it wherever you'd like.

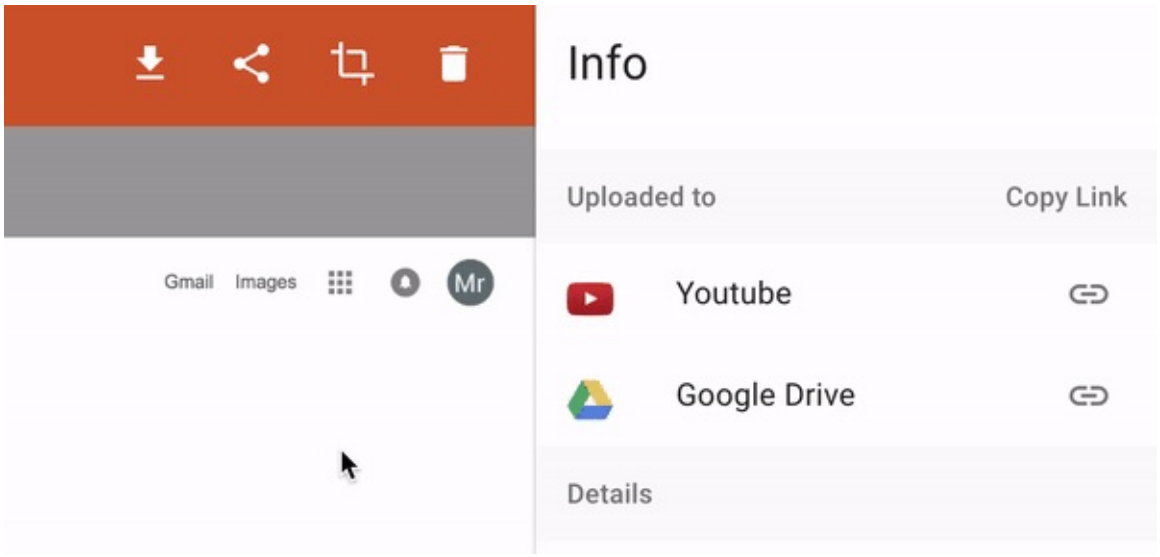
There are two ways to get the Drive link for a recording. The first:

1. From the Video Page, click the **Share** icon o open sharing options.
2. Select "**Google Drive**".
3. Click **GET LINK** to generate and reveal the shareable link.
4. Select your desired privacy settings for the recording.

5. Click **COPY LINK** to copy the link to your clipboard.



The second way to get the Drive link for a recording is by simply clicking the **Paperclip** icon next to "Google Drive" in the sidebar of the *Video Page*. The shareable link to the recording will automatically be copied to your clipboard.



Tip: If your Screencastify account is connected to your Google Drive, all of your recordings will automatically be saved in a folder called "Screencastify" in your Drive.

If you make the entire "Screencastify" folder shareable, then anybody who has a link to that folder will be able to view all of your past and future recordings.

Copyright

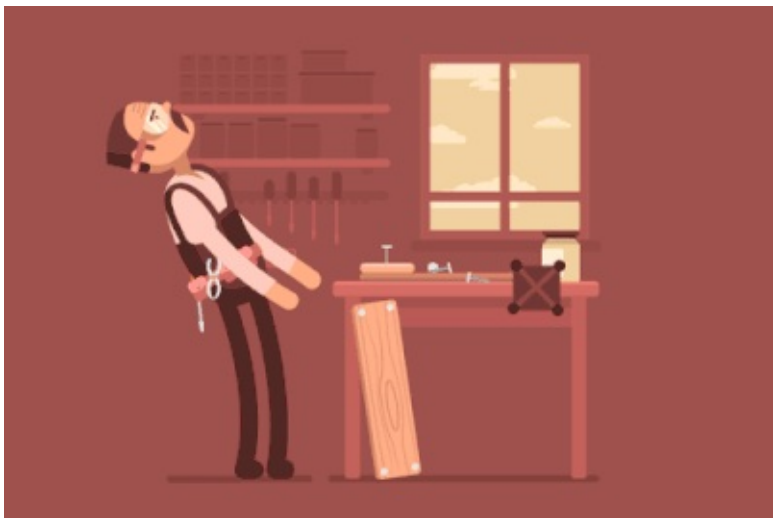
Module #6 - Sort of Code with Pseudocode (Required)

In this module, you'll be undertaking an introductory dive into the world of programming logic.

Be warned! This module is one of the most important in the Pre-Work Curriculum, and it is also one of the most challenging. Push yourself to put in the time and effort necessary to complete it fully.

Why is Programming So Hard?

One of the biggest reasons that programming is such a challenging profession is because fundamentally, computers are very, *very* dumb. Much of the complex logic and layered thinking that we humans take for granted would be completely bewildering to even the most complex of computers.



Along this vein of thought, the art of *coding* and *programming* is principally focused on taking complex ideas and on breaking them down into simple instructions that a computer, a machine, or a browser can interpret. Learning to code, thus requires proficiency in the two components necessary to translate ideas into *machine-language*.

1. The first is that every coder must learn the *syntax* that a computer understands. Syntax is sort of like the grammar and punctuation of machine-speak. It's like knowing that every sentence has a subject and a predicate or like knowing that every sentence begins with a capital letter and ends with a period. In the same way, programming languages come with their own *rule-sets*. These rule-sets may specify a special meaning for brackets or for symbols—or for a particular method for utilizing code across

different files. Throughout the course, you'll be learning a *ton* of syntax. It will seem tricky at first, but over time, it will feel like second nature. In truth, syntax will quickly become the least of your worries.

2. Instead, your real challenge will be associated with the second proficiency: converting your ideas into *computerthink*. You see, unlike people, computers are limited in their ability to work with abstract, with vague, or with general ideas. Instead, complex problems need to be broken down into small, discrete blocks of logic.

Think Like a Computer

Take for instance the following challenge:

How do you make a peanut butter and jelly sandwich?

Perhaps you might make a list that looks like the following:

1. *Get bread, peanut butter, and jelly from fridge.*
2. *Get utensils and a plate.*
3. *Slather peanut butter and jelly.*
4. *Combine smeared breads.*

This is all fine and dandy... but really, there's more to it than this.

Take Item #3, for instance: *Slather peanut butter and jelly*. A human might understand this instruction, but a computer or a machine would also need to be told the following:

- *How many times do you slather?*
- *What direction do you slather?*
- *What amount do you slather?*
- *What utensil do you use to slather?*
- *Which do you slather first?*

As you will find out, learning to code often requires one to think far more slowly than we are often accustomed. Instead of rushing through minor details, your computer will force you to slow down and to reason out your thoughts in a sometimes-nauseating number of steps.

But What About the Future?

Unfortunately, until a breakthrough in artificial intelligence emerges, you and I are stuck with our dumb computers. This is good news for us as developers, however, because it guarantees that the skills we possess will be in demand for a good time to come.

In the remainder of this module, you will be shown a video exposing you to the logical building blocks used in almost every programming language (definitely watch this!). You will then take this newfound understanding to create a simple game using the visual programming language Scratch. Don't let the cartoony interface and silly cat logo deceive you. You can build some powerful things with Scratch, and this activity will throw you for a doozy.

Video (Required)

- Introduction to Programming Logic

Assignment (Required):

- Scratch That!

Additional Reading:

- How to Tackle Something You Have No Idea How to Do

Supplemental Resources:

- Getting Started with Scratch
- Scratch Help Guide

Copyright

Assignment #6: Scratch That!

Overview:

In this assignment, you'll be channeling your inner child and creating a simple game using Scratch, a free and simplified programming language.

Your goal is to build a game that functions like any of the following examples:

- A Game of Cat and Mouse (Challenge)
- Get the Cheesy Puffs! (Recommended)
- Button Click Exercise (Too easy...)

While you have three options, we *strongly* encourage you to complete the *Recommended* or *Challenge* activities.

And yes... we know you haven't been taught Scratch. But if fifth graders can learn it on their own, then so can you. This is your first *real* challenge as a developer. Take the time to independently learn what it takes to build these applications. The logic that underlies these games is *very* similar to the logic you'll be dealing with in JavaScript so treat this as a serious endeavor. As a baseline, you can expect the recommended or the challenge assignment to take up to 15 hours of time if you are completely new to programming.

Before You Begin:

Go to https://scratch.mit.edu/ and create a new account. Then take a few minutes to watch the provided tutorials and to browse a few of the projects others have built. As you will quickly discover, you can build some amazing things with Scratch.

Instructions:

1. Watch the demos again and choose an assignment. Again, we *strongly* suggest you complete the *Recommended* or *Challenge* activities.
    - A Game of Cat and Mouse (Challenge)
    - Get the Cheesy Puffs! (Recommended)
    - Button Click Exercise (Too Easy...)
2. Make a rough outline that pseudocodes all aspects of the game. In your outline, consider answering the following:

    - What are the rules of the game?
    - How does that translate into code?
    - Will we need to use loops? if/else statements?
    - What variables will we need?
    - Under what circumstances will our variables change?
3. Once you have a rough outline, begin the process of *coding* it out in Scratch. Your final game must include the following functionality:
    - **A Game of Cat and Mouse**
        - Users can move the cat left and right with the arrow keys.
        - Users can press the spacebar to shoot a projectile.
        - Mice fall from the sky at random locations.
        - If a mouse reaches the bottom, the player loses a life.
        - If the player shoots a mouse, the mouse is deleted, and the player gets a point.
        - Once a player reaches 0 life, they lose.
    - **Get the Cheesy Puffs!**
        - Users can move the cat up, down, left, and right with the arrow keys.
        - If the player touches the cheesy puffs, their score goes up by one, and the puffs move to a random location.
        - When the player reaches a score of 10, a victory image is shown, and the game ends.
    - **Button Click Exercise**
        - A variable that starts the game at 0.
        - A button that increases the value of the variable by 1.
        - A button that decreases the value of the variable by 1.
        - A button that resets the value of the button back to 0.
4. Once you finish, create a new folder titled "Scratch Activity" in your Google Drive Pre-Work folder. Then, to that folder, upload a text file with a link to your functioning game. Be sure to make your project publicly sharable! You can test this by opening your game in Chrome's Incognito Window.

Bonus:

Find ways to add your own creative touch to things! As a suggestion, consider adding any of the following features:

- **A Game of Cat and Mouse**
    - Random *power-ups* that drop from the sky and change the cat's projectile.
    - Alternative enemies with different properties from the mouse (e.g. dogs take two

hits to defeat).
    - Music that changes speed as the player advances to further rounds.
    - Etc...
- **Get the Cheesy Puffs!**
    - Make the cheesy puffs a moving target.
    - Add in obstacles that the user has to avoid.
- **Button Click Exercise**
    - Add more buttons that change your variable in different ways!

Helpful Hints:

- Be sure to check out the supplemental resources listed in the reading chapter to get a primer on Scratch programming fundamentals.
- Try to look at other games found on the website. Then *view* the code to get inspiration for your own project. (Borrowing from the work of other developers in the open-source community is a great habit to get into, as you'll be doing the same repeatedly as a web developer.)
- Don't get discouraged if you can't complete this activity all the way. Submit what you were able to accomplish and be proud of what you scrapped together. What you will learn, time and again, in this bootcamp, is that perfection doesn't come easy and that sometimes it doesn't come at all.
- Stay motivated! This assignment is very much in line with the sorts of challenges you will face in class. At times, this will feel frustrating and pointless, but don't cheat yourself out of a good learning opportunity.

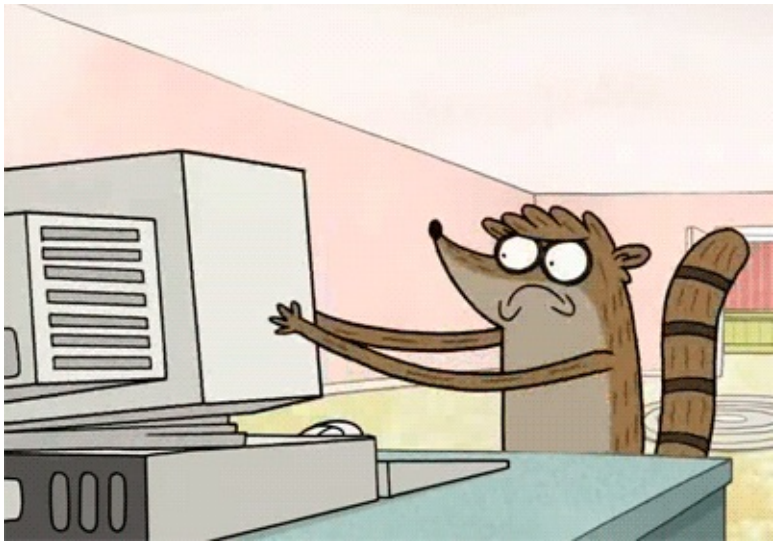
Good luck!

Copyright

Module #7 - Where's the $@#*'ing Error? (Required)

A Fact of Life

A great coder once said, *"$@#$ my life!!! Why isn't this working!?!?"*



In fact, nearly every coder, at some point, has said something like this. You see, despite what many people think, coding is far from the effortless work of writing line after brilliant line of new features and visual effects.

Instead, it is often a slow and cumbersome process—one that will resist every effort to make it work as you expect. In fact, you should already brace yourself for the reality of spending hours (if not days) chasing down missing commas, misplaced parenthesis, and diffuse cases of capitalization.

And despite what many of you may be thinking, this reality is one that will follow you throughout your career. Whether you remain a green-eyed *n00b* forever or whether you become a seasoned system architect at Google, know that errors will be a fact of life. Thus, your job (and your professional career) will partly pivot around your ability to not only create ideas and to code them out, but it will also rely on your ability to fix things when they fail to work as expected.

Let us repeat this again *together*: **Errors will be a fact of life.**

Do not let them discourage you. Do not let them bring you down. Do not let them make you quit.

Instead, relish *every* instance in which you fix something broken, and celebrate the fact that you have entered the fray as a professional developer.

But I want to be the very best. To catch them is my test....

And for that you should be applauded!

Error-catching is an important part of every developer's role. The faster you catch bugs and resolve them, the sooner you can get back to creating applications and building out your clients' latest requests.

The process of fixing broken code is called *debugging*, and just as with all problems in life, there are right ways to deal with coding issues, and there are wrong ones. In this course you will develop a fleet of strategies for debugging issues, but in this pre-work module, we'll be giving you a small taste of what it's like to work through perplexing code errors.

How to Debug

Tip #1: Start Small

Let's start with the wrong way.

Often, new developers, in facing an instance of unworking code, immediately assume that "I must have done *everything* wrong. Let me start over (or give up)."

This is a great way to waste a lot of time, but it's not a great way to learn or to get code working.

Instead, when facing an issue, always try to isolate your attention to the lines of code closest to the feature that is broken. If clicking that button doesn't trigger a pop-up like you expected, look closely at the code associated with the button. Check your syntax carefully. Check how you've named things. Check your parenthesis. Check if your capitalization is off.

Start small and center your attention on what's broken. *Then* move outward to the farther stretches of the code.

Perhaps the button isn't broken after all, but the pop-up is. Perhaps you forgot to save. Perhaps you aren't even looking at the right file. Don't be so quick to assume there is a gigantic problem with your code. Sometimes it's the smallest thing keeping you from a working solution.

Tip #2: Reference Working Code

Many new developers have this tendency to think *I'm cheating if I look back at old code.*

This thought couldn't be further from the truth! Instead, as a developer, think of yourself as a curator of references. In working with many examples of code (as you will in this class), you are continually building a bank of *good code*. Whenever you face a task or an issue, you can always reference this *Bank of Good Code* to remember how to solve a previous problem.

This is how professional developers work as well. They constantly search through their own historic code, and those of others, to quickly tackle their issues at hand. Over time, certain bits of code become second nature... but this takes time and takes many hours of practice. Don't be hard on yourself if you aren't there yet. Memorizing will take time.

But let's get back to debugging. Say we have the following instance of a broken button:

Broken Code: Up Button

```
$(".upButton").on(click, function(){
    $(".captainplanet").animate({top:"-=200px"}, "normal");
});
```

The above code was written in the hopes of moving a Captain Planet image on button click. Yet, despite our best efforts, the code isn't moving Captain Planet like we expected. Instead, he just sits there.

Now we could hunt and peck through the code at random...

or we could pull an example of working code like the one we have here:

Working Code: Down Button

```
$(".downButton").on("click", function(){
    $(".captainplanet").animate({top:"+=200px"}, "normal");
});
```

Take a moment to compare the code between these two snippets. Can you spot the error? (Really try here!)

Hopefully, you will have noticed the missing quotation marks around the word "click". Now even if you've never been exposed to JavaScript before, it should be clear that having access to the working code can help you spot the issue.

This is the value of *good code* references.

Tip #3: Keep Your Code Clean

The third key to error-free living is to keep your code *aesthetically* clean. The concept of clean code will make more sense as you progress through the class, but in the meantime, take a look at the below two blocks of code:

Sloppy Code

```
$("#randomButton").on("click", function()
{for (var i=0; i<3; i++){alert(Math.floor(Math.random() * 25) + 1);}})
```

Clean Code!

```
// When randomButton is clicked...
$("#randomButton").on("click", function(){

    // Loop through 3 times...
    for (var i=0; i<3; i++){

        // And create an alert with a new random number between 1 and 25.
         alert(Math.floor(Math.random() * 25) + 1);
    }

});
```

Believe it or not, both these blocks function *exactly* the same. However, to a developer needing to maintain or to improve the code, the latter is obviously preferable.

What you will find in this course is that you can often *get by* with creating sloppy code that is poorly indented, that is lazily organized, and that is lacking any comments. But know that YOU are the primary recipient of the pain to come. Disorganized code is harder to read and is harder to debug. Errors that are easy to spot when code is well-indented and is well-structured, become nightmares to find when it is not.

Take, for instance, the following *broken* (but very realistic) code:

Sloppy Broken Code

```
for (var i=0; i<=5; i++)
    {if(i==5)
{console.log("Yay! My favorite #5!!")}
else(){    console.log("I don't like this number very much.")
}
```

It's hard to spot, but the code was missing a `}` towards the end of the else statement.

Now take a look at the same code—with the same error—but structured with good indentation:

Clean, Broken Code

```
// Loop through a set of numbers
for (var i=0; i<=5; i++){

    // If the number is 2
    if(i==5){
        // Print it out.
        console.log("Yay! My favorite #5!!");
    }

    // If the number is not 2
    else{
        // Print it out
        console.log("I don't like this number very much.");

}
```

Assuming you knew that every `{` must end with a `}`, this format makes the error much easier to spot!

Unfortunately, the example of poorly formatted code was taken from the pages of previous students. Remember, you've been warned! Don't repeat their mistakes!

Tip #4: Read the Debug Error

As you proceed through the course, you will be introduced to browser-based debugging tools like Google Debugger. These tools will help flag troublesome lines of code that are difficult to spot.

Now, we know what you're thinking: *Omg. I am so relieved. Those last exercises were tough!*

And truthfully, you are right! These debugging tools are incredibly powerful. However, they are only as powerful as the developer who utilizes them.

Take, for instance, the following broken code and the resulting output provided by the Google Debugger.

Broken Code

```
var myName = "Sam";
var favFood = "Green Eggs and Ham";

alert(myName + " loves " + favfood);
```

Google Debugger Output

```
quick.html:3 Uncaught ReferenceError: favfood is not defined
```

For someone new to coding, the error message may be just as indecipherable as the original code, but let's try to dissect it together.

For one, we can see that it specifies a line number of 3, which means that it *thinks* the error is on line 3: `alert...`. (We say *think* because sometimes it gets tripped up by complex code.)

Next, it tells us that the error has something to do with the entry `favfood` and with its not being "defined." We'll get into concepts like variables and definitions later, but for now, consider the difference in how the `myName` data and the `favFood` data look. If you look closely, you might notice that `favFood` is capitalized once as `favFood` and is not capitalized in another as `favfood`.

This minor oversight was picked up and was flagged by Google Debugger, and with a little bit of sleuthing, it could easily be resolved. Again, don't be afraid of error messages! They are fantastic clues to your solution.

Tip #5: Test Often

This is a highly effective *preventive* strategy for keeping bugs out of your code.

Often, when new developers are assigned a task, they attempt to write *all* of the code in one sitting. If, for instance, they are tasked with creating a game of tic-tac-toe, they attempt to create the layout, the logic, the button events, the rules for determining who won, and everything else.... all without testing a single piece of functional code.

Unfortunately, this is a recipe for a hopeless labyrinth of bugs. Instead, it's a better idea to take as minimalistic an approach as possible. As you code, get into the habit of making *modest* changes, of saving them, and of then immediately testing them in the browser. This

way, you isolate the location and the number of bugs to *only* the block of code on which you are working. If you try to bite off too much, errors will creep into numerous places—each having a cascading effect.

Again, this will make more sense as you start coding, but keep this advice in the back of your mind!

Tip #6: Get Help

Despite all the stereotypes, coding is an actually very collaborative line of work. In professional settings, coders are constantly in communication with one another and with the online communities of other developers.

As you begin your journey into Web Development, always remember that it's OKAY to ask for help.

Put in the hard work. Put in the long hours. But there is no shame in asking for a second pair of eyes. Sometimes, a fresh perspective is all it takes to make a breakthrough!

Tip #7: Practice, Practice, Practice

Last—but definitely not least—is the best tip of all: *Always be coding!*

The single best thing you can do to become a faster debugger (and better coder) is to simply code a LOT. In the beginning, you can be certain that a LOT of those hours will go into mindless hunting, but don't disparage yourself by thinking those are *wasted hours*. Instead, consider every hour you spend debugging to be valuable knowledge gained. The more time you spend studying where errors exist and where they don't, the better equipped you'll be to solve any problem you come across.

Time to get Coding (and Debugging)!

Speaking of practice, it's time to work on an exercise!

Check out the assignment below to try your own hand at debugging some HTML and JavaScript.

P.S. We know you don't know HTML or JavaScript yet! That's the point :P

Assignment (Required):

- The Boo Boos in Boo's Website

Additional Reading:

- The Art of Debugging
- Do You Spend More Time Coding or Debugging

Supplemental Resources:

- Getting Started with Google Debugger (You will learn this in class, but it never hurts to learn early)
- Stack Overflow (You will be ALL over this site)

Copyright

Assignment #7: The Boo Boos in Boo's Website

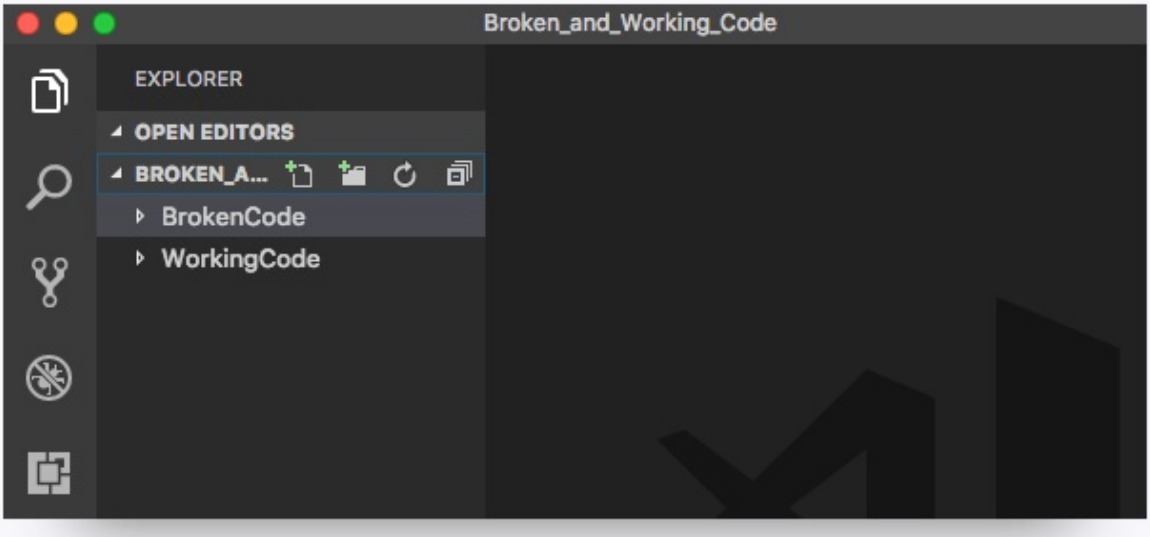
Overview:

In this assignment, you will be tasked with fixing Boo The Dog's website, using only the limited information you have and a working reference example. This is another challenging activity. Be prepared to put in 5+ hours on this activity if you are new to coding.

(Again, we know that you may be unfamiliar with HTML, CSS, and JavaScript -- but treat this like Where's Waldo. Don't get caught up in the fact that this is scary "code". Instead, treat this activity as a simple game of: "What's different?" Remember: Do your best here. This activity may take time, but it is very representative of your first few months in the program. The sooner you get comfortable spotting bugs, the less stressed you'll be as a web developer.)

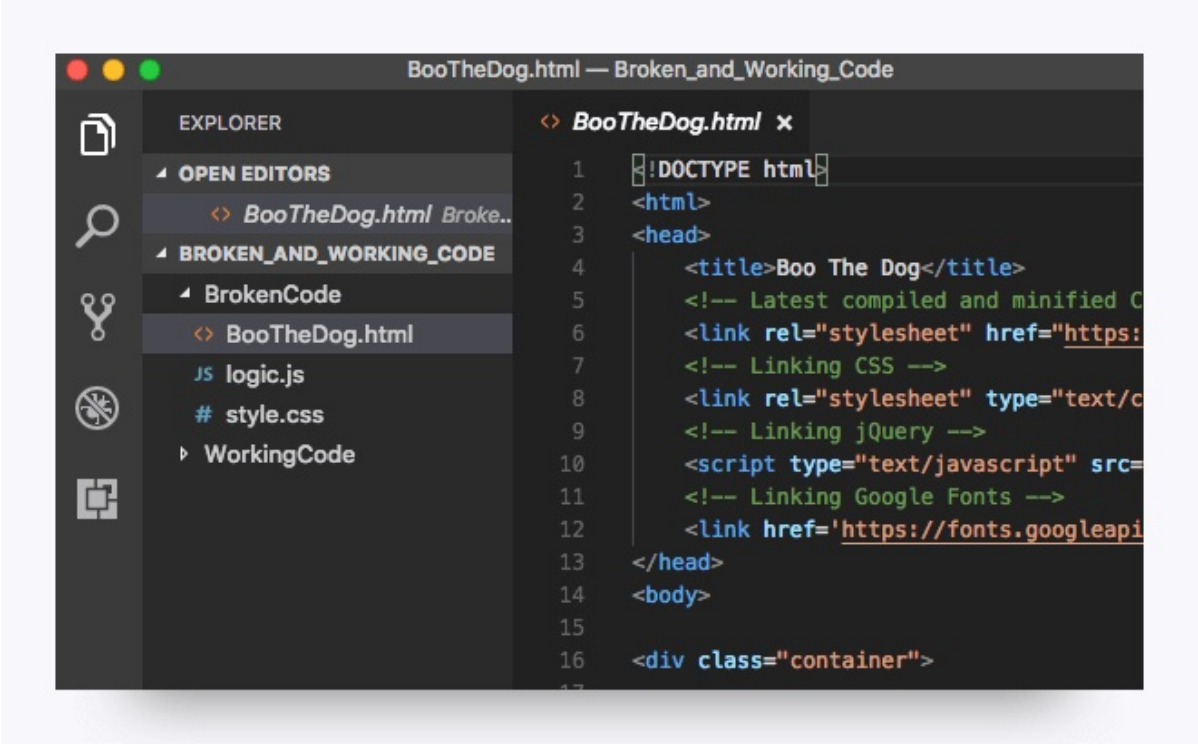
Before You Begin:

1. Download the complete code by visiting this link and clicking "View Raw".
2. Move this folder to somewhere accessible on your computer like the Desktop.
3. Then unzip the folder. You will see that it creates a folder titled `Broken_and_Working_Code` which holds two folders one titled `BrokenCode` and one titled `WorkingCode`.
4. Open up the Visual Studio Code editor.
5. Then drag the entire unzipped `Broken_and_Working_Code` folder into your Visual Studio Code editor. (Alternatively, you can click `File -> Open` and select the unzipped `Broken_And_Working_Code` folder.)
6. Then activate Visual Studio Code's explorer pane (if it isn't already active) by clicking `View -> Explorer`, or by clicking the file icon on the left sidebar. This will reveal a side bar allowing you to flip between the `BrokenCode` and the `WorkingCode`.



Instructions:

1. At this point you are all set to begin the exercise. In VSCode, select the file `GrumpyCat.html` in the `WorkingCode` folder to show the code. Then right click on the screen and click `Open in Browser`. This will show you the HTML page in your browser. Once in the browser, take a few moments to get familiar with how the page looks and functions. Click on the various buttons to understand the effect they have on the page.
2. Now return to VSCode and select the file `BooTheDog.html` in the `BrokenCode` folder. Use the same process to open this file in the browser. Once in the browser, take a few moments to get familiar with the page. You will quickly discover that each of the panels has something "broken" -- either buttons that fail to work or content that is failing to be rendered correctly.



3. For the remainder of this assignment, your task is to flip between the files in the `BrokenCode` folder and the `WorkingCode` folder to identify differences that may be the cause for the broken functionality.
4. As you identify issues, correct the code in the `BrokenCode` files and save those files. Then refresh the `BrokenCode` folder. If your changes were successful, you should see working functionality. If not... well, back to the drawing board!
5. Once you are done, copy your `BrokenCode` folder into your Pre-work Submission folder. Include in this folder a simple text file that briefly describes each of the errors you found.

Hints:

- This will be a challenging activity, no doubt! There are no shortcuts here. You need to dig in and to find the differences.
- Each panel has at least one error. Possible errors include missing commas, improper-element linking, and others. You'll need to use a discriminating eye to find each of the errors.
- As a starting point, take a few moments to fix the *aesthetic formatting* of the code in the `BrokenCode` folder (i.e. fix the indentation and line breaks between code). This will make things easier to compare.
- If you are advanced look into Google Developer Tools as a tool for helping here.

Copyright

Module #8 - HTML Hotness (Required)

Hooray! You made it! It's time to finally start coding. We'll begin our foray with the web's humble but ever-present powerhouse—HTML.

HTM... Huh?

HTML (Hypertext Markup Language) is one of the three cornerstone languages used on every webpage in existence. While the syntax might seem daunting at first, by the end of the course, you will find it simple, straightforward, and completely painless.

But what exactly is HTML?

In short, HTML handles the basic *markup* of a page. This means that HTML is responsible for the simplest aspects of our website—things like the following:

- What text elements are on the page?
- What images are on the page?
- Which element will come after which?
- Which text elements are our primary headings? Which are our secondary headings?

To draw a distinction, HTML won't be responsible for things like the following:

- Fanciful colors and layouts
- Snazzy effects on our page
- Complex user interactivity

As you will come to see, HTML effectively represents the bare *skeleton* of our webpage. We'll then use CSS and JS to add the fancy things like visual aesthetics, effects, and event-handling (like form submissions and database interactions).

Tag. You're it.

Every HTML document is made up of various pieces of contents wrapped in *tags*. These tags are most often represented by angle brackets (< tag >) with an associated tag name contained inside. We then insert our content in between an opening and a closing tag so that our browser will understand how to treat our content.

For instance, let's say I wanted HTML to style the following phrase in bold: "Coding Rocks!" To do this, I might write the below HTML.

Code:

```
<strong>Coding Rocks!</strong>
```

Visualized:

Coding Rocks!

What you can see in the above example is that we make use of the opening `<strong>` tag and the closing `<strong>` tag to wrap our content. The browser then interprets this HTML to understand—*Hey. The developer wants this phrase to be in bold.*

Don't worry, there are only a few dozen HTML tags out there, and after just a few weeks in the course, these will all become second nature.

Here are a few tags that you'll come across frequently:

- `title` - Aptly named, this tag defines the *title* of the website as shown on the webpage's tab.
- `head`, `body` - These tags help define the structure of the over-all webpage. In essence, `head` contains *invisible* matter that the browser uses to render the page correctly, whereas the `body` tag represents the actual content shown to the user.
- `h1`, `h2`, `h3`, `h4`, `h5`, `h6` - These tags represent what level of *heading* a given text block represents. Headings are exactly what they sound like—they are larger or more prominent elements of text on a page. They can be likened to *topic sentences* on a paper.
- `p` - This tag represents paragraphs or blocks of text. You'll use this tag extensively to wrap most of the text on your webpages.
- `strong`, `em` - These tags are used respectively to **bolden** or *italicize* a given text element.
- `br` - This tag is used to create a line of empty space between two blocks of content.
- `img` - This tag is used to display images on a page. The syntax is slightly different (see below), but you'll be walked further through its makeup during class.
- `a` - This tag (which stands for *anchor*) is used to create links to the same or to other webpages. Again, the syntax is slightly different, but you'll become comfortable using `a` tags as the course progresses.
- `ul`, `ol`, `li` - Lastly (for us), these tags represent unordered lists, ordered lists, and list items. In essence, these HTML elements represent bulleted lists of symbols or of numbers.

Hello, HTML

As with all things related to coding, there is no way to learn *without coding yourself*. So, let's roll up our sleeves and get started!

1. To begin, open up your Visual Studio Code editor.
2. Next, copy and paste the below code directly into your editor.

```
`<!DOCTYPE html>`
    `<html lang="en">`
    `<head>`
    `<meta charset="UTF-8">`

        <title>Hello World!</title>

    `</head>`

    `<body>`



        <h1>Panda Fan Site!</h1>

    <p>I LOVE PANDAS!!! I LOVE PANDAS!!! I LOVE PANDAS!!! I LOVE PANDAS!!! I LOVE PAND
AS!!!</p>

    <h2>Reasons I like Pandas</h2>

    <ul>

        <li>They are fuzzy</li>

        <li>They are cute<li>

        <li>They are adorable<li>

    </ul>

    <img src="http://images4.fanpop.com/image/photos/17800000/Cute-Panda-Cubs-Together
-pandas-17838800-450-324.jpg" alt="Cute Pandas">

    <br>

    <a href="http://www.pandafix.com/"PandaFix.com/>

`</body>`

`</html>`
```

3. Then select `File; Save As...`, and save the file as `My_First_Website.html` somewhere on your desktop.
4. Then right click on your Visual Studio Code window and click `Open in Default Browser`.
5. If all went well you should see the following page show up on your screen.

```
![Pandas](Pandas.png)
```

6. Rejoice!!! You just created your first HTML file.

But I just copied...

Yep. And that's what the next activity is for. It's time to work on an HTML file of your own!

But.... I wouldn't know where to start!

Sure you do!

Start by taking a look at the example you just copied. Modify the site to match what's being asked for. If there are any pieces that are still a mystery, do a little Google searching. Trust us... you will learn what it all means after the first week of class.

Assignment (Required):

- Mah Bands!

Additional Reading:

- W3 Schools - Intro to HTML
- If you know nothing about HTML, this is where you start

Supplemental Resources (Recommended):

- CodeCademy - HTML Basics
- Code School - Intro to HTML

Copyright

Assignment #8: Mah Bands!

Overview:

In this assignment, you'll pay tribute to your favorite band or musician by building a simple fan page. That's right, we're finally going to get our hands dirty, coding a website from scratch.

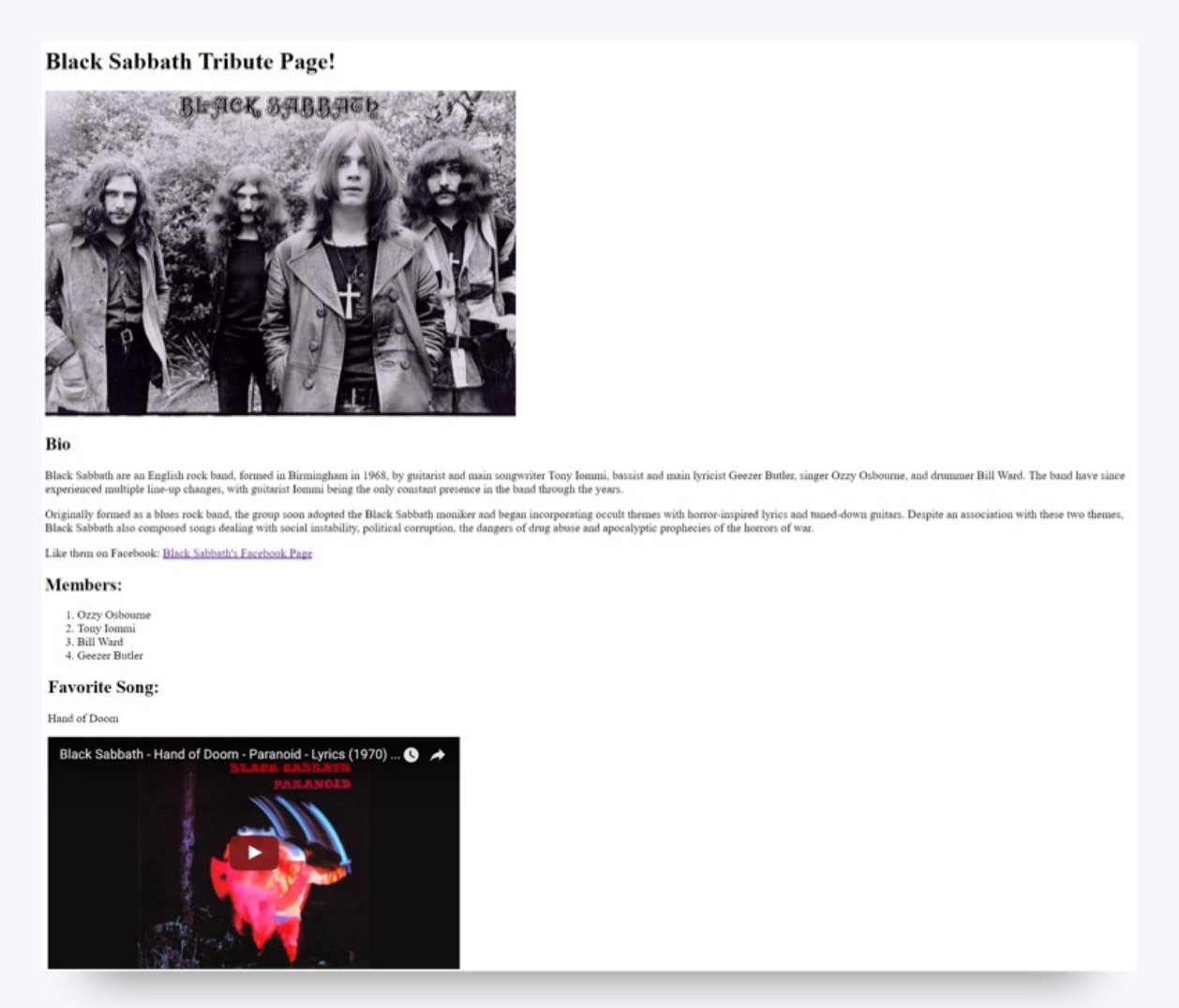
For this task, you will be using HTML to build out your page. HTML (Hyper Text Markup Language) is responsible for the underlying structure of a website. It might not look sexy, but where would you be without a skeleton?

Instructions:

1. Open VS Studio Code, create a new file, and then save it as `fanpage.html`. It's important to have the `.html` extension so that VS Code knows what type of file it is and knows how to read it.
2. Before you start adding content, create the structure for your page. Reference the example in the module chapter. Be sure to include the `!DOCTYPE` declaration and the `<html>`, the `<head>`, and the `<body>` tags where appropriate.
3. Add content to your page. Your page can be about whomever you want, but it must include the following requirements:

- Proper page structure:
  - `!DOCTYPE` declaration
  - `<html>`, `<head>`, and `<body>` tags
  - A `<title>` for the page
- At least one image (with an `alt` attribute)
- A link to another website
- An ordered or an unordered list (band members, favorite songs, albums, etc.)
- A bio section with `<p>` tags for formatting
- Use of `<h1>` and `<h2>` tags for heading
- A video of the band or musician playing
- Proper indentation

Your finished page should look something like this example:



Helpful Hints:

- Then to test, right click on the screen in VSCode and click `Open in Browser` to see how the page looks.
- If you get stuck, W3Schools has a wonderful series of tutorials on HTML: HTML Introduction.
- If you're still stuck... then Google should become your best friend. :-)

Good luck!

Bonus:

- Make your link(s) open in a new tab.
- Add a favicon to your site (the little icon that appears on the browser tab).
- Make a table.

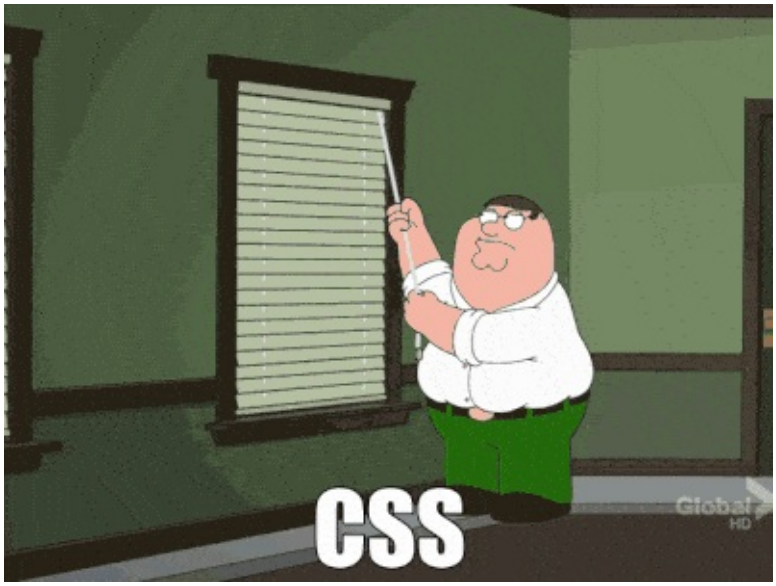
Copyright

Module #9 - See Me CSS (Required)

Now that you've had some practice creating basic HTML, it's time to take things to the next level. With the power of CSS, you can apply a wide variety of styles to your HTML and you can bring your pages to the next level of epic impressiveness.

CSS Soundbite

CSS (*Cascading Style Sheets*) is a *style sheet* language used for describing the *presentation* of a webpage. To continue our example from the previous module, if HTML is the skeleton of a webpage, then CSS is its fat, its skin, and its pinstripe suit. Whereas HTML is strictly concerned with the *markup* of webpages, CSS is focused on the colors, the aesthetics, and the visual layout instead. It works by hooking onto specific elements of an HTML page and by formatting them using any number of options (called *styles*).



Great! I love visuals!

Yeah... not so fast Van Gogh.

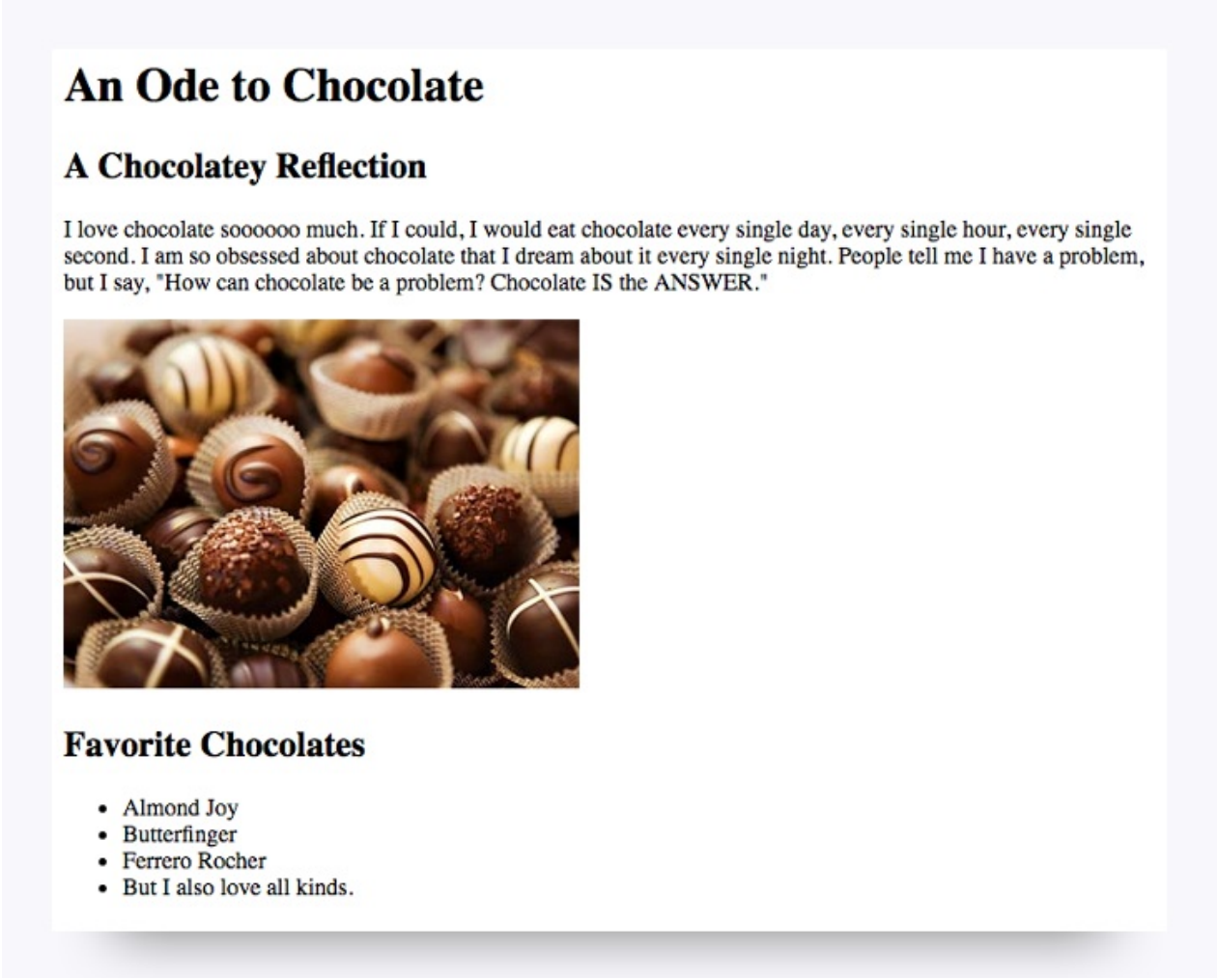
To the surprise (and horror) of many new web developers, jumping into CSS isn't as straightforward as might be expected. As is a common theme in web development, the process for formatting visual styles on a website requires an explicit level of detail and a precise command of the language. Again, it's worth remembering that creating web applications isn't a *drag and drop* process. Colors, aesthetics, fonts, and visual layouts each need to be *coded* in order for every browser to consistently render the page correctly.

A Chocolatey Dilemma

To begin our exploration of CSS, we'll need to create a starting HTML file. You can use the one below as an example.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Ode to Chocolate</title>
</head>
<body>

<h1>An Ode to Chocolate</h1>

<h2>A Chocolatey Reflection</h2>

<p>I love chocolate soooooo much. If I could, I would eat chocolate every single day, every single hour, every single second. I am so obsessed about chocolate that I dream about it every single night. People tell me I have a problem, but I say, "How can chocolate be a problem? Chocolate IS the ANSWER."</p>

<img src="http://stogiepress.com/blog/wp-content/uploads/2014/12/Chocolate-truffles.jpg" alt="chocolates">

<h2>Favorite Chocolates</h2>
<ul>
    <li class="almond">Almond Joy</li>
    <li class="butterfinger">Butterfinger</li>
    <li class="ferrero">Ferrero Rocher</li>
    <li class="all">But I also love all kinds.</li>
</ul>

</body>
</html>
```

If you copy this code into Visual Studio Code, save it as an HTML file, and then load it into browser, you should see a webpage that looks something like the below.



Not bad. But *sort of* boring.

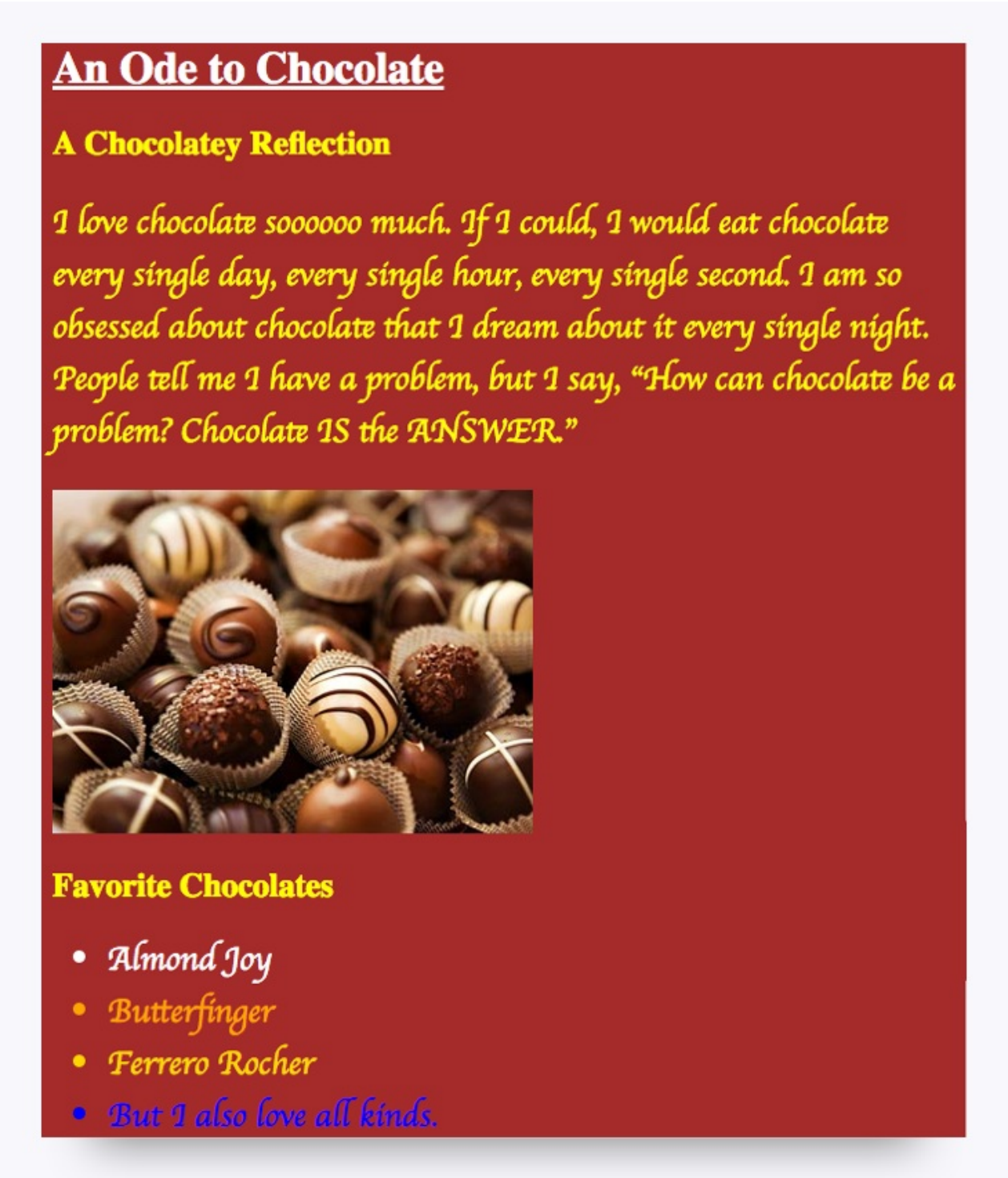
In fact, we might suspect that the author of this webpage would probably like a more chocolate-colored theme...

Enter CSS

And this is where CSS comes in! With a few extra lines of code, we can completely change the background color, the font sizes, and the font-colors of the website. Take a moment to add the below code to your HTML file. (It should fit right between the `</body>` and `</html>` tags of the previous example.)

```
<!-- .. Rest of Code.. -->
<!-- </body> -->

<!-- BEGIN CSS -->
<style>

    body {
        background-color: brown;
    }

    h1 {
        color: white;
        text-decoration: underline;
    }

    h2, h3, p {
        color: yellow;
    }

    p, li {
        font-size: 24px;
        font-family: cursive;
    }

    .almond {
        color: white;
    }

    .butterfinger {
        color: orange;
    }

    .ferrero {
        color: gold;
    }

    .all {
        color: blue;
    }

</style>
<!-- END CSS -->

<!-- </html> -->
```

Once we re-save our HTML file, the layout will look like the below.



Much chocolatey-er!

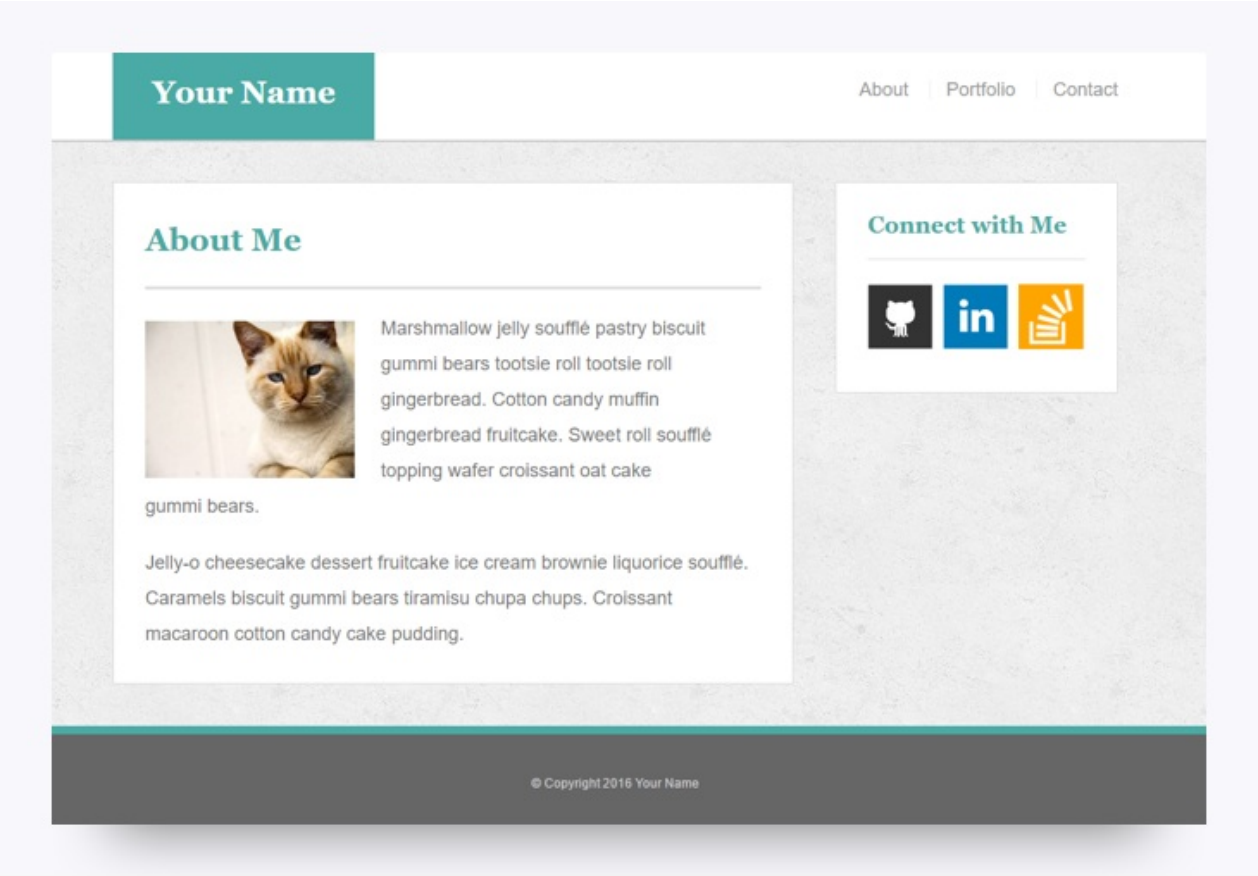
What's this Wizardry??

If you spend a few moments looking at the new code, you will notice that we used a consistent syntax like the below.

```
HTML-TAG {
        CSS-PROPERTY: VALUE
    }
```

In effect, we're referencing specific HTML elements and then applying changes to how they are formatted. The format options are each known as properties in CSS, with specific options for how they can be modified. Don't expect to memorize all of the formatting options available through CSS. For many of your early weeks and months as a developer, you will need to continually reference websites like W3 Schools, among others, to recollect the exact syntax.

Believe it or not, through just simple HTML and CSS alone, you will be able to build complex web layouts like the one below. (In fact, this is a screenshot of your first homework assignment in class.)



But What's with the Classes?

If you were paying close attention, you may have noticed that certain HTML elements in our last example also included a mysterious use of the word `class`. We used these throughout the list of favorite chocolates:

```
<ul>
    <li class="almond">Almond Joy</li>
    <li class="butterfinger">Butterfinger</li>
    <li class="ferrero">Ferrero Rocher</li>
    <li class="all">But I also love all kinds.</li>
</ul>
```

Classes (along with IDs) offer us a method to style specific or multiple HTML elements using the same CSS. We'll talk a lot about classes and IDs during the program, but it's helpful to have some exposure in advance.

To get you started, this next assignment will give you some practice in styling HTML, using such CSS selectors like classes and like HTML tags.

Which is a great transition...

Your Turn!

It's time to get coding! This next assignment will have you styling very simple HTML, using CSS.

Assignment (Required):

- Style This Page!

Additional Reading:

- W3 Schools - Intro to CSS

Supplemental Resources (Recommended):

- Learn CSS - Crash Course #1 (Free)
- Code School - CSS Cross Country

Copyright

Assignment #9: Style This Page!

Overview:

In this assignment, you will be tasked with using CSS to style an HTML page.

Before You Begin:

As we discussed in the chapter, CSS allows us to stylize our HTML webpages with colors, with fonts, and with much more. It allows us to define a style once and to easily use it repeatedly. Exciting stuff right? So, let's dive right in!

Instructions:

1. Create two files on your computer using Visual Studio Code. One should be called `index.html` and the other should be called `style.css`. These should be in the same directory.
2. Then copy (and save) the code below into your HTML file. This is your starting HTML.

```
<!DOCTYPE html>
<html>
<head>
    <title>CSS Practice</title>
    <!-- Link to your external CSS -->
```

```
</head>
<body>

    <!-- Center -->
    <h1>The Wonders of CSS!</h1>

    <hr>

    <h2>Fun With Text!</h2>

    <!-- Change text color -->
    <p>This text is red.</p>
    <p>This text is blue.</p>
    <p>This text is green.</p>

    <!-- Change font-size -->
    <p>This text is BIG!</p>
    <p>This text is tiny...</p>

    <!-- Center -->
    <p>~~This text is centered~~</p>

    <!-- Bold -->
    <p>If we have something important to say, we can make it bold!</p>

    <!-- Change font -->
    <p>We can even change our font-family if we are feeling creative!</p>

    <!-- Multiple attributes -->
    <p>Try combining multiple attributes to make big, orange, bold text!</p>

    <hr>

    <!-- Classes -->
    <p class="styledFont">We can also apply the same style to multiple HTML elements.</
p>
    <p class="styledFont">Each one of these lines is within a different &ltp&gt tag, y
et they all have the same styling.</p>
    <p class="styledFont">That's because they are all a part of the same CSS class!</p>

    <p class="styledFont">CSS classes let us define a style once and let us then apply
 it to multiple elements!</p>
    <p class="styledFont">In your external CSS file, create a style for the class styl
edFont.</p>
    <p class="styledFont">Notice how every time you change the class styledFont, all o
f the text changes.</p>

    <hr>

    <p>Just as classes can be used to style multiple elements, id's are used to style
single, unique elements. Give the following image a border by adding style attributes
to your external CSS file. (Hint: Remember to include a border width, style, and color
```

```
.)</p>

    <!-- Border & id -->
    <img id="styledImage" alt="Picture of the CSS Icon" src="https://www.viewlike.us/wp-content/uploads/2013/04/css3-markup.jpg">

    <hr>

</body>
</html>
```

3. Now write CSS in your `style.css` file such that each of the paragraphs of text are formatted with the appropriate styles.
4. Your finished product should look something like the below.

5. After you finish, include both your HTML file and your CSS file in your Pre-Work Submission folder.

Hints:

1. You may need to modify the HTML for this to work. Think `classes`.
2. Look back at the working example from the chapter to find out how to `link` HTML and CSS files.
3. When in doubt W3-Schools is where you should be.

Bonus:

- Revisit your fan page from the last exercise and add some style to it with CSS.

Resources

- HTML Styles - CSS

Copyright

Module #11: Time to get Employable! (Required)

Career Services

Our Career Services team is built on the principle of empowerment. We are here to **empower you** to find a job using the skills you have learned in our program. We can't do the work for you, but we can motivate, guide, and equip you to reach beyond your comfort zone to find the right opportunity for you.

Don't forget that we can only help you based on the information you share with us; the more engaged you are, the more successful you will be!

Career Services Resource Library (Review required)

Career Services Resource Library

This library is an addition to your Career Curriculum and serves as a source of guides and templates to help you navigate the job search process. Please ensure that you visit this library often to help supplement your Career Curriculum and provide you the resources you might need to be successful.

What to expect on the job hunt:

When transitioning into a new career, a reasonable timeline from starting applications to accepting a company's offer is 30-90 days. You'll spend time preparing your professional narrative and network just as you spend time honing your coding skills.

What you DO need to succeed:

1. A professional narrative that showcases your strengths & passions.
2. Ability to showcase your previous skills as adding to your value as a technology worker.
3. A friendly demeanor - people hire people they like!
4. Ability to talk about technology - even technology you aren't familiar with - by relating it to concepts you do know. Yes, jargon matters. We need to call it an HTML element, and not a "thing".
5. Practice, practice, practice - the more networking events and interviews you complete, the more prepared you'll be for your next opportunity.

What you DON'T need to succeed:

1. **"Perfect" looking code.** No one's code is perfect! Does it work as expected? You're golden!
2. **Mastering all technologies taught in the course.** You'll learn so much in this bootcamp, it'll make your head spin. The skill you're really trying to pick up: learning how to teach yourself new technologies using online resources.
3. **Previous coding experience.** Many companies will let you learn on the job and will appreciate the unique skill set you bring to the table.
4. **Every skill listed on the job description.** Often, the job description isn't written by anyone familiar with technology, and it can be misleading. Apply anyway! Talk about your strengths, and the ability to learn quickly based on your bootcamp performance!
5. **Certificate of completion from the bootcamp.** Once we've approved your Bootcampspot career profile - you're ready to start applying! Don't be afraid to start early, practice makes perfect, and you'll only get better the more interviews you complete.

Create accounts (Required):

They're all free! Here are some resources that will get you started on the pathway to a new career:

1. LinkedIn

   Join LinkedIn to get the latest news, insights, and opportunities from over 3 million companies. It'll act as a professional profile, and it has an activity feed full of wisdom from companies & individuals you connect with and\/or follow.

2. Stack Overflow

   Stack Overflow is the Wikipedia of code - question & answer style. It's like yahoo answers, except it has *good* answers. You can create questions and answer them too!

3. Builtin

   Applicable in the following markets: Austin, Chicago, Colorado, LA, NYC, and Boston.

   Builtin is a recruitment site for tech companies and startups! Hover over "Our sites" to get started. They'll keep you in the loop by delivering news & notifying you of job openings.

4. Angel List

   Where the world meets startups. Angel List helps investors, startups, and job seekers connect!

5. Meetup

   Meetup brings people together in thousands of cities to do more of what they want to do in life. This site is chock full of opportunities to meet with others who are passionate about learning code - just head to the technology category!

6. Gmail

   Gmail is email that's intuitive, efficient, and useful. 15 GB of storage, less spam, and mobile access. It comes with free storage on Google Drive which you'll use to submit some of your homework.

7. Master the Google Document

   You will be using Google documents to submit the first several career homework assignments. We want to ensure you know how to make your document editable for your Career Services team to be able to provide timely and meaningful feedback. Please watch this short video to become familiar with Google documents. Don't forget to try to create a shared document right after watching the video!

8. Medium

   Medium is home to the world's curious minds. It's a blogging platform full of technology content - start following Free Code Camp, and browse other technology recommendations today!

Assignment (Recommended):

- Connect professional dots!

Other Resources:

- Stack Overflow Meta - Stack Overflow, for beginners! The moderators are kinder, and the community is more forgiving of newbies. Signing up for Stack Overflow does not sign you up for Stack Overflow Meta - they're separate!
- Eventbrite - Bringing the world together through live experiences. A great tool for finding recurring or single events.

Final Notes:

Job hunting is stressful, and trust us, *no one likes it*. Don't worry - we'll be here to support you through learning the hard skills and prepare you to succeed in the interview process. We need your participation and hard work - **but we'll get through this together.**

Good luck!

Copyright

Assignment #11: Connect professional dots

Overview:

This assignment is designed to dip your toe in the networking water and talk to others in the technology field!

Instructions:

1. Read through the "Time to get employable" chapter of Pre-Work.
2. Browse Eventbrite and/or Meetup for an event that occurs **before the bootcamp starts** that sparks your interest!
3. Attend the event, alone or with friends, and talk to at least two people **you did not know previous to the event**.

Note:

- No trick assignment here. We want you to meet people, have fun, and reflect.

Copyright

Module #10 - Jiggle into JavaScript (Required)

You've been through a lot already, but now it's time to bring out the big guns!

Using JavaScript, you'll be able to power your websites with effects, with interactivity, with data communication, and with much more. Throughout the course of the bootcamp, you'll be spending nearly 15 weeks on JavaScript alone. It's a critical tool in every web developer's arsenal so let's take an introductory look now.

The Joys of JavaScript

JavaScript (or JS for short) is a high-level dynamic programming language that underpins the web. JavaScript, along with HTML and CSS, is one of the core technologies that makes our web experience what it is.

Like C++, Java, Ruby, and Python, JavaScript is written with all the fixings expected of a complete programming language. In it, you will find variables, conditionals, loops, functions, and so much more. We'll be using JavaScript extensively to create the logic that defines the behavior of our web applications.

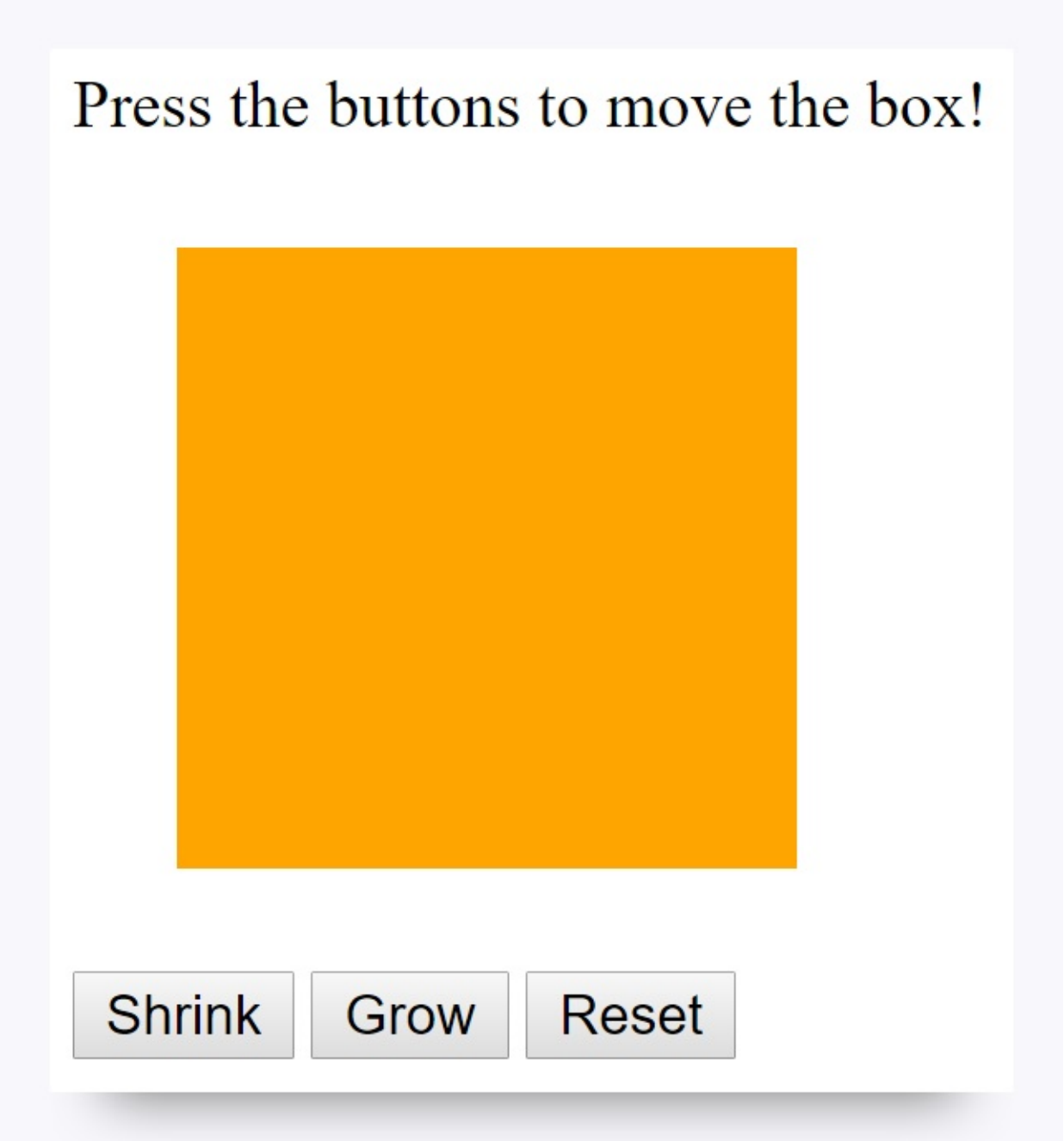
How to JavaScript

While the full power of JavaScript is beyond the scope of this pre-work, let's get a small taste of what's possible.

Go ahead and open Visual Studio Code and create a new file called `index.html`. Then copy the below code into this file and save it somewhere on your computer.

```
<!DOCTYPE html>
<html>
<head>
    <title>Jiggle Into JavaScript</title>
</head>
<body>

    <p>Press the buttons to move the box!</p>

    <div id="box" style="height:150px; width:150px; background-color:orange; margin:25
px"></div>

    <button id="shrinkBtn">Shrink</button>
    <button id="growBtn">Grow</button>
    <button id="resetBtn">Reset</button>

    <script type="text/javascript">

    </script>

</body>
</html>
```

If you view this page in the browser, you should see a page that looks like the below.



Pretty fancy!

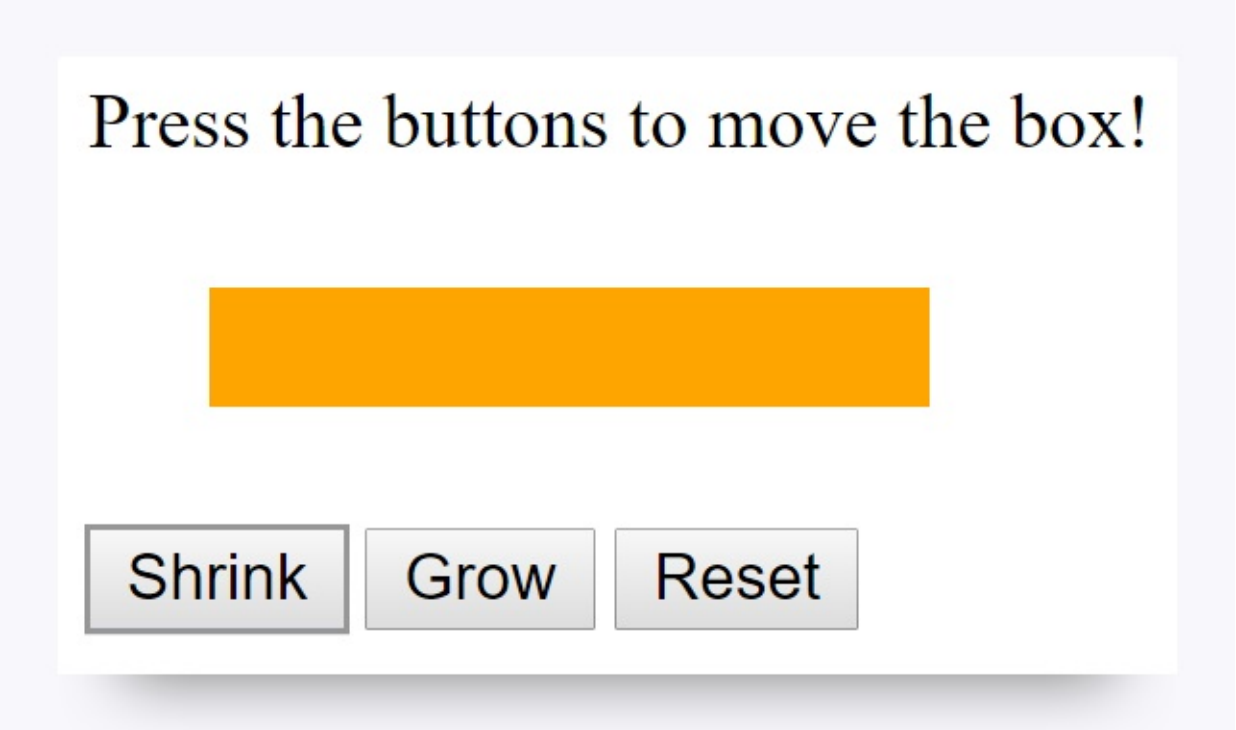
The thing is... if you try clicking the buttons, you will quickly realize that they have no impact on the position of the box. No fun whatsoever. This is where JavaScript comes in.

Copy the below code over your previous HTML file and re-save.

```
<!DOCTYPE html>
<html>
<head>
    <title>Jiggle Into JavaScript</title>
</head>
<body>

    <p>Press the buttons to move the box!</p>

    <div id="box" style="height:150px; width:150px; background-color:orange; margin:25
px"></div>

    <button id="shrinkBtn">Shrink</button>
    <button id="growBtn">Grow</button>
    <button id="resetBtn">Reset</button>

    <script type="text/javascript">

        document.getElementById("shrinkBtn").addEventListener("click", function(){

            document.getElementById("box").style.height = "25px";

        });

        document.getElementById("growBtn").addEventListener("click", function(){

            document.getElementById("box").style.height = "250px";

        });

        document.getElementById("resetBtn").addEventListener("click", function(){

            document.getElementById("box").style.height = "150px";

        });

    </script>

</body>
</html>
```

Now open the file once again in the browser.

This time, clicking the buttons will lead to changes in box height.



The Magic Behind the Magic

So how did that work?

The essence of this interactivity sits between our two `script` tags on lines 16 and 36. There you'll see a series of code blocks responsible for the changes being made. Let's take a moment to dissect the components:

1. The lines each begin with `document.getElementById`. In a way, this is simply a reference to say, "If you ever click the button with an id that matches then do *something*". These id's (e.g. `shrinkBtn`, `growBtn`, and `resetBtn`) correlate with the id's of the buttons in the HTML.
2. The lines then continue, to say that we'll `addEventListener("click"...)`. This code effectively means that we'll be *watching* for any clicks on our targeted buttons.
3. We then open up a `function` with some code inside. This code targets the `box` id and re-styles the height to an arbitrary pixel size. In a way, we're using JavaScript to dynamically change the CSS of our box in response to button clicks.

While the syntax may seem scary, the concepts are simple. Take a few moments to experiment some more. Here you can find an entire table of changes you can make using the `document.style` syntax we're using above.

Is That All?

Unfortunately, this simple example doesn't even scratch the surface. Through JavaScript (and its cousin jQuery), you can build truly powerful web applications with complex user interfaces, with dynamic live-reloading data visualizations, with geolocation tools, and with so much more. There's so much that JavaScript can do so come prepared to have your mind blown repeatedly over the course of the coming six months.

Your Turn!

Now it's time to step up to the JS bat. For the final assignment of this pre-work, you'll be taking your elementary knowledge of JavaScript to create a similar box-modifying application. The syntax might be tricky, but through a little persistence, you'll be box-changing in no time.

Assignment (Required):

- Watch That Box

Additional Reading:

- W3 Schools - Intro to JS

Supplemental Resources (Recommended):

- Learn JavaScript - CodeCademy
- JavaScript Road Trip - Code School

Copyright

Assignment #10: Jiggle into JavaScript

Overview

In this assignment, you will be using JavaScript to change the CSS properties of a box upon button clicks.

Before You Begin

You may want to spend a little time reading through each of the below references. Just skim enough to get the high-points.

- onClick Events
- JavaScript Click Examples
- JavaScript HTML DOM Events

Instructions

1. Create two files on your computer using Visual Studio Code. One should be called `index.html` and the other should be called `javascript.js`. These should be in the same directory.
2. Then copy (and save) the code below into your HTML file. This is your starting HTML.

```
<!DOCTYPE html>
<html>
<head>
    <title>Jiggle Into JavaScript</title>
    <!-- <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.1.0/jquery.min.js"></script> -->
</head>
<body>

    <p>Press the buttons to change the box!</p>

    <div id="box" style="height:150px; width:150px; background-color:orange; margin:25px"></div>

    <button id="button1">Grow</button>
    <button id="button2">Blue</button>
    <button id="button3">Fade</button>
    <button id="button4">Reset</button>

    <script type="text/javascript" src="javascript.js"></script>

</body>
</html>
```

3. Now write your JavaScript in the `Javascript.js` file such that clicking the buttons re-styles the box appropriately (i.e. When a user hits "Grow" the box should increase in size, when a user hits "Fade" the box should change opacity, etc.).

4. After you finish, include both your HTML file and JavaScript file in your Pre-Work Submission
folder.

5. Upload your Google Drive folder.

Bonus

Try adding a few buttons of your own, with different CSS effects. There's a LOT more than box-changing that you can do with JavaScript, but it's a fun start!

Hints

- Push yourself here. Be resourceful and use Google searches if you get stuck. There's plenty of material to learn from.

- If you don't quite get it to work, submit what you have! No shame in not knowing just yet. You've got six months to figure this stuff out.

Copyright

Module #12 - What Next? (Recommended)

Just the beginning!

So now that you've handled HTML, slain CSS, and jiggled into JavaScript, what's next? Well, first of all, congratulate yourself! If you've made it this far, then you've already learned some critical web development fundamentals, and you've set yourself up for success.

However, don't get too settled. Your journey is only just beginning. As you will soon discover, part of what makes being a web developer so exciting and so rewarding is the endless opportunity to learn.

So, where do I go from here?

The most important thing for you to do now is to simply continue coding. You have come a long way by completing this pre-work, but the longer you go without coding, the harder it will be to get back into the habit. Don't let your knowledge wilt on the vine.

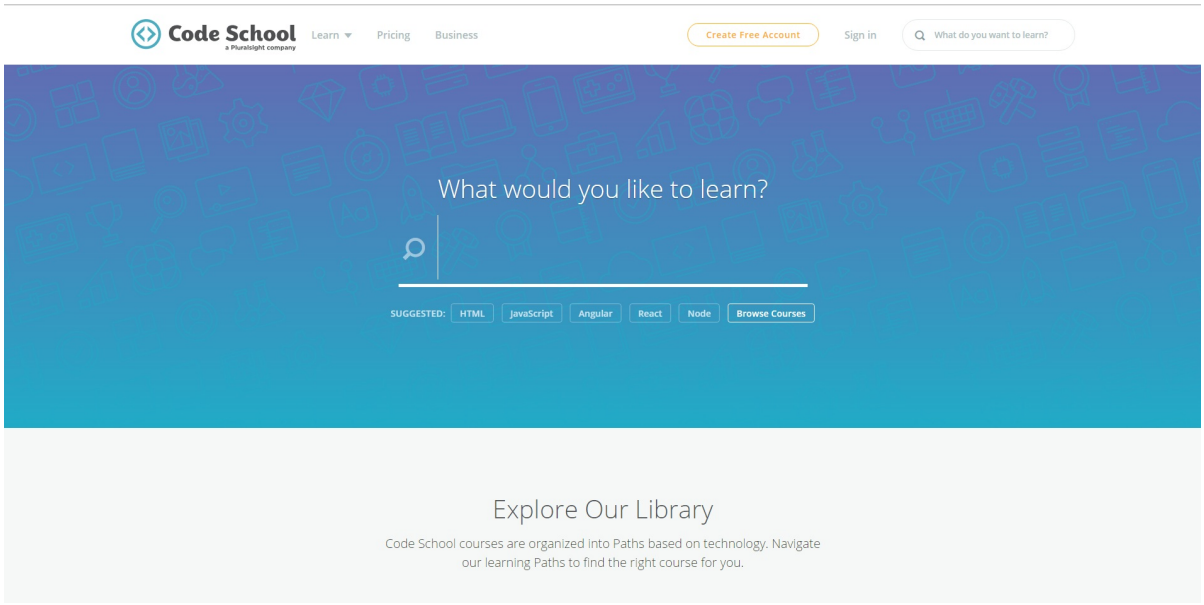
Becoming a successful developer is going to require a good deal of discipline and independent motivation. Begin looking for opportunities to expand your horizons. See a cool website? Inspect the source code! Have a problem in your everyday life? Try to figure out a way to solve it using logic and code. The most successful developers never cease learning or finding ways to improve.

Yes. But I need links...

That's the spirit! Here are some resources that you can use to reinforce and to expand upon your pre-work knowledge:

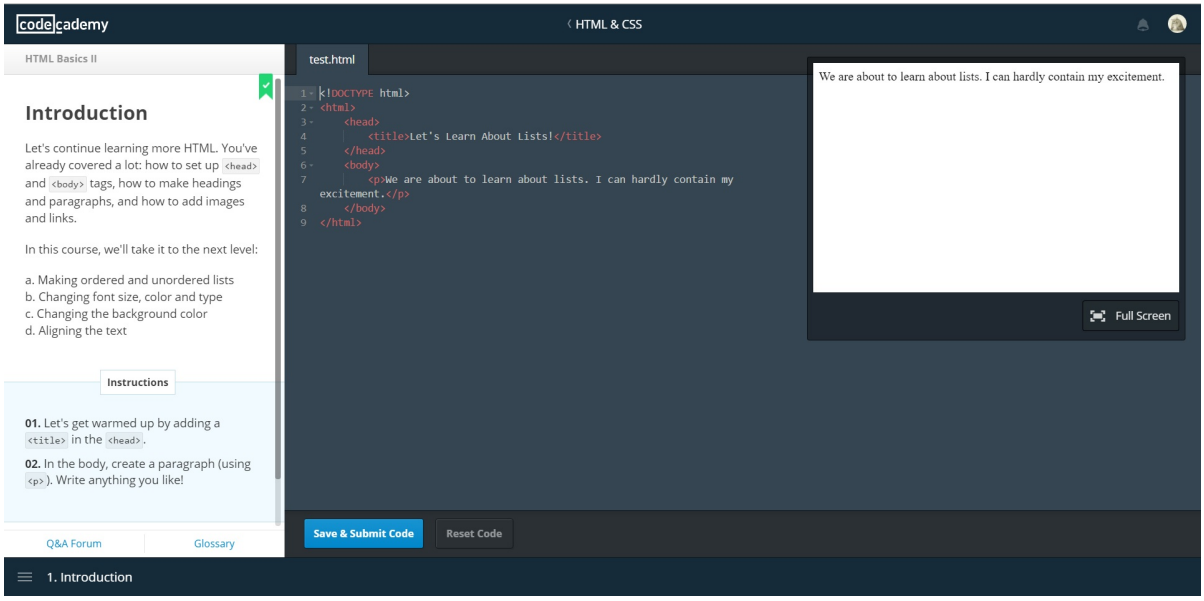
1. Code School

   Code School is one of the single best learning websites for beginners. If you've completed all of the pre-work modules, we strongly encourage you to complete as many of the HTML, CSS, JavaScript and jQuery modules as you can. It will be an enormous help for the first few weeks of class.



2. Codecademy

Codecademy is another excellent site that offers free courses on languages such as HTML, CSS, and JavaScript. Each course is broken up into a number of step-by-step lessons that slowly guide you through learning the fundamentals and through completing simple projects.

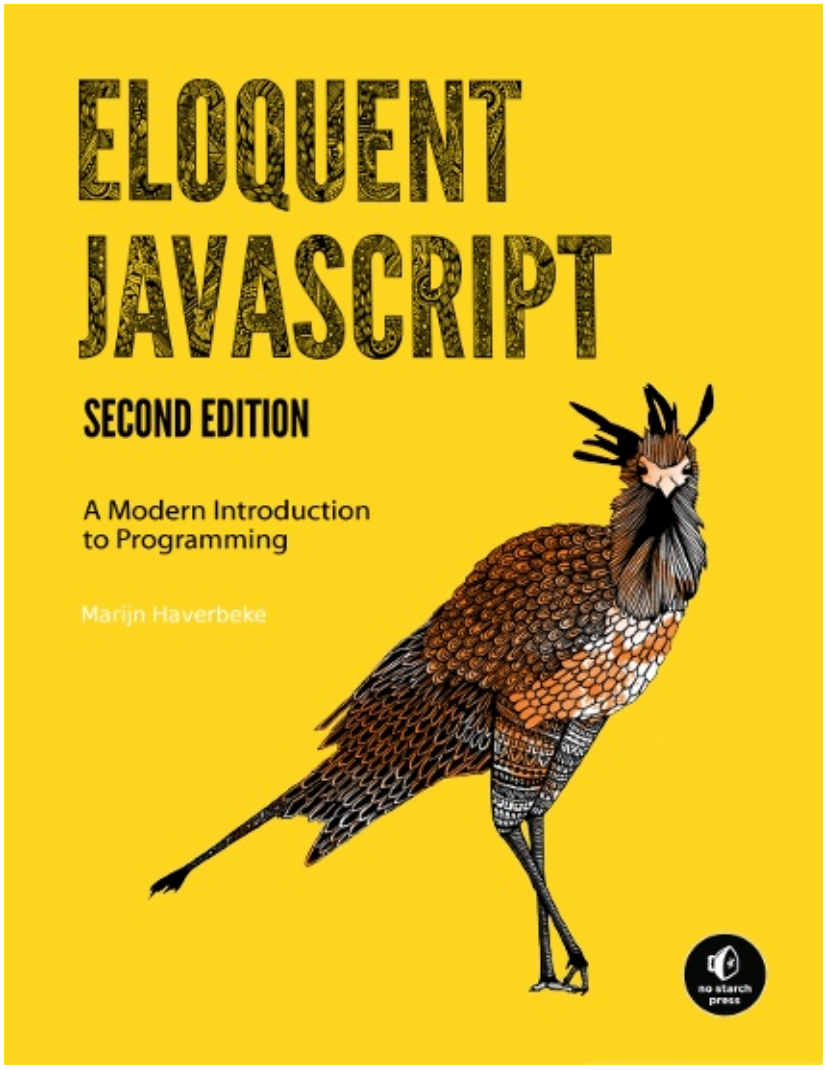


3. Free Code Camp

Not ready for summer to be over yet? Go back to camp! Complete lessons to gain certifications and to slowly build a portfolio. Finish all of your certifications, and you can get real world experience building projects for non-profits.

4. Eloquent JavaScript

   Great (free) textbook that really grounds you in JavaScript fundamentals and vocabulary.



Other Resources:

Reading:

- Learning How to Learn - One of the most important skills you will learn in this program is how to independently learn. This course grounds you in a methodology.

Learn from others:

- Livecoding.tv - Watch (and chat with) developers live as they code projects. Great for seeing the thought process of other developers. It's also fun to watch them struggle with bugs.
- TwitchTV - Programming - Similar concept as Livecoding.tv.

YouTube:

- LevelUpTuts - Great (free) resources for taking your web development skills to the next level. Both novices and advanced developers will find plenty to learn from here.
- LearnCode.academy - Another great resource, particularly for advanced topics on JavaScript, jQuery and Angular.js.

Final Notes:

Once again, congratulations on your enrollment into the Coding Bootcamp *and* on completing the pre-work. You've got a long road ahead, but we'll be cheering you on, every step of the way.

Good luck!

Copyright