**Pre-Work Journal**

**Errin Mixon**

**1.2**

**Mattermost**

<http://chat.kickstartcoding.com>

errin

**LinkedIn**

[www.linkedin.com](http://www.linkedin.com)

[errinmixon@gmail.com](mailto:errinmixon@gmail.com)

**GitHub**

[www.github.com](http://www.github.com)

errinmarie

**Heroku**

[www.heroku.com](http://www.heroku.com)

[errinmixon@gmail.com](mailto:errinmixon@gmail.com)

**Stack Overflow**

[www.stackoverflow.com](http://www.stackoverflow.com)

Errin

**NPM**

[www.npmjs.org](http://www.npmjs.org)

errin

Right now I am happily employed, and not seeking a web developer position. I want my LinkedIn profile to indicate a strong work history, but satisfaction in my current position (because I will be visible to my colleagues). I also hope to indicate an interest in continuing education with mention of Kickstart Coding, and I’m hopeful that I’ll be able to flesh this section out once the course is completed. After building this page from scratch, I realize how important it is to keep my resume up to date, even during times of employment. I always want to be ready for collaboration and opportunity, if not a full career change.

My GitHub page says literally nothing about me, which is fine for now, because I have 0 experience. I will keep it up to date as I go along.

**2.1 - Jargon**

**Programming language**

A language comprised of a set of instructions used to produce various kinds of output. Programming languages are used to create programs that implement specific *algorithms* (specifications for how to solve a class of problems). Most consist of instructions for computers (although programming language pre-dates computers – consider the player piano!). Thousands of different programming languages have been created, mainly in the computer field, and more are being created every year. Many take an *imperative* form, meaning a sequence of operations to perform. But some take a *declarative* form, meaning the desired result is specified, but not how to achieve it.

**Native app**

An application developed for a particular platform or device (iPhone, Android, etc). It can work, in most cases, with no internet connection (depending on the nature of the app). Because native apps are written for a specific platform, they can interact with and utilize OS features and other software that is typically installed on that platform (for instance, GPS). *Native apps need installation* – Example: Facebook app

**Web app**

A client-server computer program, which the client runs in a web browser. A web app can be used on various devices, but requires internet access, and operation speeds are dependent on the quality of the cell signal, or the strength of the wifi. *No installation required* – just use the URL address. Example: [www.facebook.com](http://www.facebook.com)

**API**

*Application Programming Interface*

Think of an API like a menu in a restaurant. The menu provides a list of dishes you can order, along with a description of each dish. When you specify what menu items you want, the restaurant’s kitchen does the work and provides you with some finished dishes. You don’t know exactly how the restaurant prepares that food, and you don’t really need to.

Similarly, an API lists a bunch of operations that developers can use, along with a description of what they do. The developer doesn’t necessarily need to know, for example, how an operating system builds and presents a “Save As” dialog box. They just need to know that it’s available for use in their app.

APIs make life easier for developers.Let’s say you want to develop an app for an iPhone. Apple’s iOS operating system provides a large number of [APIs](https://developer.apple.com/api-changes/)—as every other operating system does—to make this easier on you.

If you want to capture photos or video from the iPhone’s camera, you don’t have to write your own camera interface. You use the camera API to embed the iPhone’s built-in camera in your app. If APIs didn’t exist to make this easy, app developers would have to create their own camera software and interpret the camera hardware’s inputs. But Apple’s operating system developers have done all this hard work so the developers can just use the camera API to embed a camera, and then get on with building their app. And, when Apple improves the camera API, all the apps that rely on it will take advantage of that improvement automatically. This helps reduce the amount of code developers need to create, and helps create more consistency across apps for the same platform.

**Version control**

The task of keeping a software system consisting of many versions and configurations well organized.

Version control systems (VCSs) are a category of software tools that help a software team manage changes to source code over time. Version control software keeps track of every modification to the code in a special kind of database. If a mistake is made, developers can turn back the clock and compare earlier versions of the code to help fix the mistake while minimizing disruption to all team members.

**Caching**

A cache is a temporary storage area. For example, the files you automatically request by looking at a Web page are stored on your hard disk in a cache subdirectory. When you return to a page you've recently looked at, the browser can get those files from the cache rather than the original server, saving you time and saving the network the burden of additional traffic. Caching is the process of storing data in a cache.

**Database system**

A database (DB) is a collection of data that lives for a long time. This could be a paper-based file system, a notebook, or a computerized system. A computerized database is usually accessible by many concurrent users. A Database Management System (DBMS) is a system (software) that provides an interface for this information storage and retrieval. The common features of a DBMS include:

* capacity for large amount of data
* an easy to use interface language (SQL)
* efficient retrieval mechanisms
* multi-user support
* security management
* concurrency and transaction control
* persistent storage with backup and recovery for reliability

**Scrum team**

A *scrum* is a game plan for managing work (usually software development). A *scrum team* is a team of developers, usually 3-9 people, who break their work into actions that can be completed in a certain amount of time. These chunks of time, usually called *sprints*, are 30 days or less (commonly 2 weeks). Progress is tracked in *daily scrums*, 15-minute meetings dedicated to monitoring headway and adjusting course as needed.

**2.1 – Coding-Related Positions**

**Front-end engineer**

Even the most useful websites will only attract users if its pages are cutting-edge, interactive and constantly updated. Front-end engineers program the visible and interactive features of a website that users see (animations, text boxes, navigation buttons, games and forums). They typically work with back-end engineers who process data behind the scenes. *But the front-end engineer must think like the typical user and design a site that is engaging and easy to navigate.* Their tasks are:

1. analyze the needs of the website audience
2. work with managers, marketers & tech staff to develop specifications for design and development
3. make templates that web developers can use, and create pages and applications using such languages as JavaScript, HTML and CSS; 4)
4. test their development to ensure it meets all usability requirements.

**Back-end engineer**

The technology and programming that “power” a site—what your end user doesn’t see but what makes the site run—is called **the back end.** Consisting of the [server](https://www.upwork.com/hiring/for-clients/a-guide-to-server-technology/), the [database](https://www.upwork.com/hiring/data/a-guide-to-database-technology/), and the [server-side applications](https://www.upwork.com/hiring/development/server-side-scripting-back-end-web-development-technology/), it’s the behind-the-scenes functionality—the brain of a site.

* The back end is a combination of a [database](http://www.upwork.com/hiring/data/a-guide-to-database-technology/) and a software written in a [server-side language](https://www.upwork.com/hiring/development/server-side-scripting-back-end-web-development-technology/), which are run on web servers, cloud-based servers, or a hybrid combination of both.
* This server-side application directly interacts with the database via an application programming interface (API), which pulls, saves, or changes data.
* The data are returned and converted into front-end code a user interacts with: filling out a form, creating a profile, shopping online, etc.
* In general, anything you see on a site is made possible by back-end code, which exists on, and is powered by, a server.

**QA (Quality Assurance) Engineer**

Software QA engineers make sure that new products work before they are released to the public. They monitor every phase of the software development process to ensure quality and deliver on time and on budget. (Each factor bears weight in the product’s financial success.)

QA engineers are sometimes confused with software testers. Software testers test parts of the software at different stages of development, whereas a QA engineer oversees the entire development process, which includes software testing, from start to finish.

**Project Manager**

Software project managers provide the consistency needed to see software projects through from start to finish. With a mixture of technical knowledge and leadership, they oversee both the product being produced and the people performing the work. Software project managers generally work full time in an office environment, though performing extra hours is commonplace when problems need resolution or deadlines near. After years of experience, software project managers may move on to positions such as senior product manager or software engineering manager.

**DevOps Engineer**

A [DevOps](https://searchitoperations.techtarget.com/definition/DevOps) engineer works with software developers, system operators ([SysOps](https://searchitoperations.techtarget.com/definition/system-operator-sysop)) and other production IT staff to oversee code releases. The role calls for someone who has the necessary [hard](https://searchcio.techtarget.com/definition/hard-skills) and [soft skills](https://searchcio.techtarget.com/definition/soft-skills) that are required to overcome the traditional barriers between software development, [testing](https://whatis.techtarget.com/definition/software-testing) and [operations](https://searchitoperations.techtarget.com/definition/IT-operations) teams.

Traditionally, the developers (people who create software) had incentives that were vastly different from operations (people who run software.) For example, as a developer, I want to create as **many**new features as fast as possible. After all, this is my job and that’s what customers demand! However, if I’m an ops person, then I want as **few**new features as possible because every new feature is a change and change is risky.

As a result of this misalignment of incentives, DevOps was born. DevOps attempts to fuse development and operations into one group. The idea is that one group will now share both the pain and the responsibility (and presumably, the rewards) of creating, deploying, and generating revenue from customer-facing software.

**Data Analyst**

[Data analysts](https://www.snagajob.com/job-search/q-data+analyst) translate numbers into plain English. Every business collects data, whether it's sales figures, market research, logistics, or transportation costs. A data analyst's job is to take that data and use it to help companies make better business decisions.

Though I’m not currently considering a career change, I think the Front-End Engineer position appeals to me most. It seems the most visually creative, and I’m accustomed to (and good at) designing & writing materials with ease of use in mind.

Since my personal goal is to develop my own apps, I think it will be crucial for me to understand both front end and back end engineering, and best practices for QA. I’m not currently interested in data analysis or DevOps, and I have a good handle on the concept of project management (albeit in other industries). But these seem more geared to a company-culture. I’m leaning more toward the entrepreneurial.

**2.2**

Hacker News July 2018

* Python - 61
* Node - 35
* Ruby - 30
* Django - 14
* React - 71
* Angular - 9
* jQuery - 1
* Postgres - 37
* Firebase - 0
* MySQL - 4

**Python**

Python is a high-level, general purpose programming language. (General purpose languages can be used for both front & back end.) It was designed for writing [software](https://en.wikipedia.org/wiki/Software) in the widest variety of [application domains](https://en.wikipedia.org/wiki/Application_domain). (A counter-example would be a specific application domain like a [page description language](https://en.wikipedia.org/wiki/Page_description_language), designed to write programs that control the layout of text and graphics on a page). Python’s design philosophy emphasizes [code readability](https://en.wikipedia.org/wiki/Code_readability), using [significant whitespace](https://en.wikipedia.org/wiki/Significant_whitespace). <https://wiki.python.org/moin/BeginnersGuide/Overview>

**Django**

A free and open source web application framework, written in Python. A *web framework* is a set of components that helps you to develop websites faster and easier.

When you're building a website, you always need a similar set of components: a way to handle user authentication (signing up, signing in, signing out), a management panel for your website, forms, a way to upload files, etc. Luckily, other people long ago noticed that web developers face similar problems when building a new site, so they teamed up and created frameworks (Django being one of them) that give you ready-made components to use. Frameworks exist to save you from having to reinvent the wheel and to help alleviate some of the overhead when you’re building a new site.

<https://tutorial.djangogirls.org/en/django/>

**PostgresSQL (or Postgres)**

A powerful, open source, object-relational database management system. Postgres uses and extends the SQL (Structured Query Language), with an emphasis on safely storing and scaling the most complicated data workloads. It can handle workloads ranging from small single-machine applications to large [Internet-facing applications](https://en.wikipedia.org/wiki/Web_service) with many [concurrent users](https://en.wikipedia.org/wiki/Concurrent_user). On [macOS Server](https://en.wikipedia.org/wiki/MacOS_Server), PostgreSQL is the default database. It is also available for [Microsoft Windows](https://en.wikipedia.org/wiki/Microsoft_Windows) and [Linux](https://en.wikipedia.org/wiki/Linux).

SQL is the most common standardized language used to access databases. A relational database stores data in separate tables rather than putting all the data in one big storeroom. The database structures are organized into physical files optimized for speed. You set up rules governing the relationships between different data fields, and the database enforces these rules. With a well-designed database, your application never sees inconsistent, duplicate, orphan, out-of-date, or missing data.

PostgreSQL is free and open source software. You are free to use, modify and distribute PostgreSQL in any form.

**React**

React is a JavaScript library for building user interfaces (UIs). At the heart of all React applications are *components*. A component is essentially a piece of the UI. So, when building applications with React, we build a bunch of independent, isolated and reusable components and then compose them to build complex UIs.

<https://programmingwithmosh.com/react/react-tutorial/>

**Hacker News articles**

<https://thisissecurity.stormshield.com/2018/08/28/acridrain-stealer/>

AcridRain is a new password stealer that showed up on forums in mid-July 2018. This is interesting because I didn’t realize that bad guys had forums. I didn’t understand most of the article, but I did notice that AcridRain seems to be a *programming language* – a set of instructions designed for a specific output. (Would this be a *declarative* language?) It follows a series of steps to set itself up, first retrieving info about the environment, running various tests, and targeting specific browsers, to access credentials, cookies and credit cards. These are ‘dumped’ (or copied), then output to a zipped file, giving the bad guys a handy parcel of all your private info.

I realized belatedly that there aren’t many comments on this article, but the gist seems to be that it’s really easy to steal your stuff. If you can execute code on the computer (as the user), you can decrypt the credentials, as long as the user is logged in.

Great.

<https://thenextweb.com/hardfork/2018/08/29/china-bitcoin-data-dark-web/>

I learned more from the comments than the article itself. ‘Hackers sell data of 130 million Chinese hotel guests on the dark web for 8 Bitcoin’ – the title sums up the article. But the comments informed me:

* The article had misinformation. It claimed the company’s developers accidentally uploaded their entire database to GitHub. In fact, they only uploaded a file that contained the password to the database. This tells me that news writers aren’t tech people – and perhaps they should be, when writing tech articles. Most people do not understand this stuff; it is a completely foreign language to much of the world. We do not understand how our data is being protected, breached, etc. The article spreads misinformation, but the tech community neatly sidesteps it, giving the impression that this happens all the time. This makes me wonder: should computer science be a mandatory course of study? Or is this the way the world has always worked: with only a small percentage of people understanding the course of progress?
* Apparently, it is really easy to accidentally upload stuff to GitHub, and some people think there should be protections in place to guard against this sort of thing.
* The article said that the dark web could be accessed by VPN, which made me take note, because we use a VPN at my job. The comments refute this.
* What makes this newsworthy (beyond the data breach) is that the price was set so low, and the sale was made public. Also, someone ‘reputable’ was monitoring the dark web and spotted the ad in the first place. (I don’t actually know what the dark web is, so I’m picturing undercover cops when it’s probably much more mainstream.)
* According to one commenter, you can buy all the info it takes to steal an identity (name, DOB, address, ss #, etc) for between $1-10.
* According to another commenter, all that info can be generated – faked – by remixing data from former hacks.

**2.3 Linux & free software research**

**Proprietary (closed-source) software**

*Only the original authors of proprietary software can legally copy, inspect, and alter the software.* If we make a copy and give it to a friend, if we try to figure out how the program works, if we put a copy on more than one of our own computers in our own home, we could be caught and fined or put in jail. That’s what’s in the fine print of the license agreement you accept when using proprietary software. (Microsoft Office and Adobe Photoshop are examples of this.)

**Open source software**

*Source code* is the part of software that most computer users don't ever see; it's the code computer programmers can manipulate to change how a piece of software works. Programmers who have access to source code can improve that program by adding features to it or fixing parts that don't always work correctly. *Open source software has source code that anyone can inspect, modify, and enhance.*

**Free software**

Free software gives the user the freedom to run, copy, distribute, study, change and improve it. *We call this free software because the user is free.* (To understand the concept, you should think of “free” as in “free speech,” not as in “free beer”.)

## The difference between open source and free software

## The *free software* movement began in 1983, with the goal of protecting the users’ freedom. But in 1998, part of the community splintered off and began campaigning in the name of *open source*.

*Open source* soon became associated with ideas and arguments based only on practical values, leaving the ethical considerations aside. Open source proponents believe that when programmers can read, redistribute, and modify the source code, software evolves. People improve it, adapt it, and fix bugs at an astonishing speed – much faster than the slow pace of conventional software development. This makes open source software more business friendly. Proprietary software is simply an inferior solution to a practical problem.

For the *free software* movement, free software is an ethical imperative, essential respect for the users' freedom. It says that non-free software is a social problem, and the solution is to stop using it.

While the philosophies of the two movements are different, both cooperate in their efforts toward software development, and against proprietary software, software patents, and the like.

**Free software license**

Grants the recipient extensive rights to modify and [redistribute](https://en.wikipedia.org/wiki/Software_distribution) that software. These actions are usually prohibited by [copyright](https://en.wikipedia.org/wiki/Copyright) law, but the rights-holder (usually the author) of a piece of software can remove these restrictions by accompanying the software with a [license](https://en.wikipedia.org/wiki/Software_license) which grants the recipient these rights.

**Open-source license**

Allows the [source code](https://en.wikipedia.org/wiki/Source_code), blueprint or design to be used, modified and/or shared under defined terms and conditions. Open-source licensed software is mostly available [free](https://en.wikipedia.org/wiki/Freeware) of charge, though this does not necessarily have to be the case. These licenses may have some restrictions, such as a requirement to preserve the name of the authors and a copyright statement within the code, or to redistribute the licensed software only under the same license.

**FOSS development (Free and Open Source Software)**

*Traditional software* *development* is likened to the way cathedrals were built in ancient times. Small groups of skilled artisans carefully planned out the design in isolation and everything was built in a single effort. Once built, the cathedrals were complete and little further modification was made.

Software was traditionally built in a similar fashion. Groups of programmers worked in isolation, with careful planning and management, until their work was completed and the program released to the world. Once released, the program was considered finished and limited work was subsequently done on it.

In contrast, *FOSS* *development* is more akin to a bazaar, which grows organically. Initial traders come, establish their structures, and begin business. Later traders come and establish their own structures, and the bazaar grows in what appears to be a very chaotic fashion. Traders are concerned primarily with building a minimally functional structure so that they can begin trading. Later additions are added as circumstances dictate.

Likewise, FOSS development starts off highly unstructured. Developers release early minimally functional code to the public and then modify their programs based on feedback. Other developers may come along and modify or build upon the existing code. Over time, an entire operating system and suite of applications develops and evolves continuously.

**GPL (General Public License)**

A widely used [free software license](https://en.wikipedia.org/wiki/Free_software_license), which guarantees [users](https://en.wikipedia.org/wiki/End_user) the freedom to run, study, share and modify the software. The license was originally written by [Richard Stallman](https://en.wikipedia.org/wiki/Richard_Stallman) of the [Free Software Foundation](https://en.wikipedia.org/wiki/Free_Software_Foundation) (FSF) for the [GNU Project](https://en.wikipedia.org/wiki/GNU_Project), and grants the recipients of a [computer program](https://en.wikipedia.org/wiki/Computer_program) the rights of [the Free Software Definition](https://en.wikipedia.org/wiki/The_Free_Software_Definition). The GPL is a [*copyleft*](https://en.wikipedia.org/wiki/Copyleft)*license*, which means that [derivative work](https://en.wikipedia.org/wiki/Derivative_work) can only be distributed under the same license terms. GPL was the first copyleft license for general use.

*My understanding: You may only use this software if all your derivative works guarantee the same freedoms to users. Maybe not so great for businesses or proprietary software companies.*

**MIT license**

A [permissive free software license](https://en.wikipedia.org/wiki/Permissive_free_software_license) originating at the [Massachusetts Institute of Technology](https://en.wikipedia.org/wiki/Massachusetts_Institute_of_Technology) (MIT). As a permissive license, it puts only very limited restriction on reuse and has, therefore, an excellent [license compatibility](https://en.wikipedia.org/wiki/License_compatibility). The MIT license permits reuse within [proprietary software](https://en.wikipedia.org/wiki/Proprietary_software) provided that all copies of the licensed software include a copy of the MIT License terms and the copyright notice. The MIT license is also [compatible](https://en.wikipedia.org/wiki/License_compatibility) with many [copyleft](https://en.wikipedia.org/wiki/Copyleft) licenses, such as GPL. (MIT licensed software can be integrated into [GPL](https://en.wikipedia.org/wiki/GPL) software, but not the other way around.)

*My understanding: Businesses and/or proprietary software companies may use this software for their own purposes, even if they don’t subscribe to the FSF philosophy. But they cannot incorporate this software into their proprietary derivative works. Users must be made aware that this piece of the tech is open-source. Am I understanding this correctly?*

**Types of Linux FOSS**

**Linux**

Linux is the best-known and most-used [open source](https://opensource.com/resources/what-open-source) operating system. As an operating system, Linux is software that sits underneath all the other software on a computer, receiving requests from those programs and relaying these requests to the computer’s hardware.

In many ways, Linux is like other operating systems you may have used before, such as Windows, OS X, or iOS. In fact, many types of software from other OSs have Linux equivalents (such as word processing software). In many cases, the software’s creator may have made a Linux version of the same program you use on other systems.

But Linux also is different from other OSs. First, Linux is open source software. Its code is free and available to the public to view, edit, and contribute to.

Second, there are many distributions of Linux, which include different software options. This means that Linux is incredibly customizable, because applications (such as word processors and web browsers) can be swapped out – but so can core components (such as which system displays graphics, and other UI choices).

**Debian**

An *operating system (OS)* is the set of basic programs and utilities that make your computer run. At the core of an operating system is the kernel. The *kernel* is the most fundamental program on the computer and does all the basic housekeeping and lets you start other programs.

On top of the kernal are all the basic tools. Next is all the software that you run on the computer. At the top of the tower is Debian — carefully organizing and fitting everything so it all works together.

Debian systems currently use the [Linux](https://www.kernel.org/) kernel or the [FreeBSD](https://www.freebsd.org/) kernel. But Debian provides more than a pure OS: it comes with over 51000 free [packages](https://www.debian.org/distrib/packages): precompiled software bundled up in a nice format for easy installation on your machine.

**Ubuntu**

<https://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html>

The Ubuntu community is built on the ideas that software should be available free of charge, that software tools should be usable by people in their local language and despite any disabilities, and that people should have the freedom to customize and alter their software in whatever way they see fit. Ubuntu is the world’s most popular Linux-based desktop OS, boasting an estimated 20 million users worldwide. (Linux has a 2% share of desktop computer use.)

**Red Hat Enterprise Linux**

Designed for businesses and targeted toward the [commercial](https://en.wikipedia.org/wiki/Business) market, Red Hat uses trict [trademark](https://en.wikipedia.org/wiki/Trademark) rules to restrict free re-distribution of their officially supported versions of Red Hat Enterprise Linux, but still freely provides its source code. [Third-party derivatives](https://en.wikipedia.org/wiki/Red_Hat_Enterprise_Linux_derivatives) can be built and redistributed by stripping away non-free components like Red Hat's trademarks. RHEL can work on desktops, on servers, in hypervisors or in the cloud. RHEL and its community-supported counterpart, Fedora, are among the most widely used Linux distributions in the world.

[www.tourubuntu.com](http://www.tourubuntu.com)

The Ubuntu interface looks like a Mac. Downloading applications looks similar to using the app store on my iPhone – it seems very user-friendly compared to Windows, which makes me wary and unwilling to download apps without IT assistance. I wonder if their spreadsheet and text editor programs are compatible with Microsoft Office Suite.

[www.omgubuntu.co.uk](http://www.omgubuntu.co.uk)

**Flatpak** allows you to run multiple versions of the same app side-by-side, without issue.

**3.1 Skill Immersion**

[www.keyhero.com/free-typing-test](http://www.keyhero.com/free-typing-test)

Speed 73.71 WPM; Accuracy 91.19%

Speed 93.66 WPM; Accuracy 96.11%

Speed 94.43 WPM; Accuracy 95.60%

<https://www.keyhero.com/custom-typing-test/html-typing-test-2/>

Speed 19.00 WPM; Accuracy 100%

Speed 15.39 WPM; Accuracy 88.24 %

Speed 15.96 WPM; Accuracy 86.11%

Speed 28.48 WPM; Accuracy 98.48 %

Speed 37.07 WPM; Accuracy 98.61 %

Speed 23.02 WPM; Accuracy 93.75 %

Speed 12.75 WPM; Accuracy 79.49 %

Speed 25.20 WPM; Accuracy 92.86 %

Speed 22.28 WPM; Accuracy 88.24 %

Speed 29.31 WPM; Accuracy 100 %

<https://www.keyhero.com/custom-typing-test/bash-git/>

Speed 32.15 WPM; Accuracy 93.75%

Speed 71.58 WPM; Accuracy 100%

Speed 111.21 WPM; Accuracy 100 %

Speed 62.35 WPM; Accuracy 100 %

Speed 90.02 WPM; Accuracy 100 %

Speed 81.73 WPM; Accuracy 100 %

Speed 34.81 WPM; Accuracy 90.48 %

**3.2 Git practice**

try.github.io/levels/1/challenges/1

Version control systems (VCSs) are defined above. *Distributed version control systems* (DVCSs) allow full access to every file, branch, and iteration of a project. Every user can access a full and self-contained history of all changes.

*Git* is a FOSS DVCS. Every time you save the state of your project, Git basically takes a picture of what all your files look like at that moment and stores a reference to that snapshot. To be efficient, if files have not changed, Git doesn’t store the file again, just a link to the previous identical file it has already stored. In this way, it can handle projects from small to very large, with a tiny footprint.

*Write in your prework journal observations you make about controlling the files with git commands.* I had difficulty locating the tutorial – the link above directs me instead to a main page.

**3.3 HTML**

<https://developer.mozilla.org/en-US/docs/Learn/HTML/Introduction_to_HTML/Getting_started>

<p>They</p>

<p>are</p>

<p>taking</p>

<p>the</p>

<p>Hobbits</p>

<p>to</p>

<p>Isengard.</p>

<p>

<ul>

<li>

<a href=”https://www.facebook.com/”>Facebook</a>

</li>

<li>

<a href=”<https://www.amazon.com>/”>Amazon</a>

</li>

<li>

<a href=”<https://www.apple.com>/”>Apple</a>

</li>

<li>

<a href=”https://www.google.com/”>Google</a>

</li>

<li>

<a href=”<https://www.microsoft.com>/”>Microsoft</a>

</li>

</ul>

</p>

*I plugged this into the ‘editable code’ section of the HTML tutorial page and it looked right – but my links didn’t go anywhere, but I’m not sure why.*

**3.4 Python**

This is like Logo! I remember this from elementary school!

import turtle

wn = turtle.Screen()

alex = turtle.Turtle()

alex.forward(150)

alex.left(90)

alex.forward(150)

alex.left(90)

alex.forward(150)

alex.left(90)

alex.forward(150)

*Why did they stop teaching us this stuff after 6th grade??*

*I could be a billionaire by now.*