Having completed Code In Place, there are many paths you might choose to take: this handout is intended to provide a possible roadmap for some of them. Like all guidelines, this is an approximate and non-exhaustive list on how to learn about a subject. There are a plethora different ways to achieve mastery that might not be represented here.

We have split up this resource into umbrella topics that students have been asking about. In reality, there are a lot more ways to apply Computer Science, such as music, healthcare, social good and education.

> Each area has a list of **core topics** you need to understand to engage with meaningfully with that discipline. After you develop a solid understanding of these core topics, you can pick and choose whatever subareas you find interesting and focus there.
>
> If there is any topic not covered here that you are interested in, ask about it in the comments! We will try to answer those as well.

How to use this resource:

Since we can't cover the vast range of possible material in the world, most of the resources linked below are websites of university courses that cover the relevant concept. ***This doesn't mean you have to take the academic course to learn that material***. Instead, the course syllabus can serve as a reference point for which topics are important to cover. You can then go about actually learning these topics however you want. Some potential ways to learn could be to search for the topic on google or youtube, read textbooks, or browse reddit.

Foundations of CS

These are major concepts that are important to solidify, irrespective of what area of CS you are interested in. It consists of all the foundational concepts: programs, variables, loops, conditions, data structures, and core algorithms.

CS106A:

You have done most of this! Congrats :)

CS106B:

Stanford's sequel class to CS 106A. In CS 106B, you further develop your toolkit by deepening your understanding of how a computer represents data and learning about additional problem-solving techniques and structures. The class is taught in C++, one of the most popular programming languages in history.

Course website: http://web.stanford.edu/class/cs106b/

Old recorded lectures: https://see.stanford.edu/Course

Area Specific Paths

Data Science, AI, and ML

Core:

- Probability and Statistics (CS109)
- Data science with numpy and matplotlib (CS102, Harvard Data Science)
- Linear algebra and Multivariable Calculus (Mathematics for Machine Learning, 3b1b Linear Algebra, 3b1b Calculus, MIT18.06, Coding the Matrix)

AI/ML:

- Machine learning (CS231N, CS229)
- Natural language processing (CS124, CS224N)
- Computer vision (CS231N)
- Reinforcement learning (CS234)

Systems

Core:

- Computer Architecture and Systems (CS107 website and videos)
- Principles of Systems (CS110)

Graphics/Game Design/VR

Core:

- Linear algebra and Multivariable Calculus (Mathematics for Machine Learning,3b1b Linear Algebra, 3b1b Calculus, MIT18.06, Coding the Matrix)
- Core physics (mechanics, differential equations)

Areas:

- 2D/3D Graphics, rendering, animation, geometry (CS248)
- Virtual Reality (EE267)
- Animation and Simulation (CS348C)
- Introduction to Game Design and Development (CS146)

Web dev (frontend/backend)

Web development is how we make websites and online applications to do useful things. It consists of two major areas.

Frontend: This deals with everything related to what a website user can see and interact with such as the design, style, menus, text, images, etc.

Backend: This deals with everything that has to do with all the logic and internal working of a website that is not typically visible to a user. This is stuff like storing data in databases, making a server, authentication, creating users, generating dynamic pages, etc.

These resources will generally cover both frontend (HTML, CSS, javascript) andbackend programming:

- Mozilla: Learn Web Development
- The Odin Project
- The Flask Mega Tutorial

Mobile dev

- iPhone: any Swift resources provided by Apple
- Android: any Kotlin resources provided by Google

General Resources

Programming Tools

- One of the most wonderful things about the field of Computer Science is how collaborative and open a field it is. This is enabled by websites like Github, GitLab and BitBucket, in which programmers can share and work together on their code. Underlying each of these websites is a system called Git, which allows you to manage the different versions of your program with minimal fuss. Learn the basics of Git using Github's tutorial, or BitBucket's help center. Some SLs wrote up really great intros to Github. See this ed post and this other video!
- In addition to PyCharm and Ed, there are countless other editors and IDEs, each with their own strengths and which can be customized for your purposes. Two of the most popular editors are Visual Studio Code and Sublime Text.
- As you work on projects of your own, you are bound to run into bugs. Stack Overflow is a question & answer forum that is probably the single best resource on the internet to get advice on resolving bugs.

Other Programming Languages

Python is a wonderful language, but there are many other wonderful languages you might be interested in learning:

- HTML, CSS and Javascript are the best tools for developing internet-based application. Mozilla's resources are a fantastic introduction.
- C and C++ are two of the most commonly used programming languages, and are great for programmers who want more direct control over what their computers are doing. You can learn C++ from a combination of CS 106B and CS 106L, and you could learn C here.
- Rust is a more recent language that also affords programmers very low-level control of their computers.
- Java and Go are great choices to build systems that must handle large amounts of data.
- Haskell, Scala, and OCaml are programming languages that promote a style of programming known as functional programming, which often is enormously helpful in processing data and is a fascinating intellectual endeavour.

Tech Interviews

Interviews for tech internships and jobs are kind of their own skill that really gets better with practice. These are some good resources to practice for tech job interviews. Remember, you don't want to be memorising these answers. Instead you want to develop your computational thinking so that you can figure out these answers on the spot!

- Cracking the Coding Interview
- Hackerrank

Collection of Resources by SLs

These are a collection of various resources contributed by the section leaders of Code in Place. They aren't structured in any particular way but they might serve as a useful reference for you!

Take a MOOC, such as CS106B and CS107, or the Coursera version of CS 229 or Coursera's Deep Learning
More MOOCS: list from Class Central
Try codewars.com to learn more Python and other languages
Find a learning partner (in your section, or someone else in the class) so that you can help each other continue learning to code -- having peer/community support can help a lot! (It could help if a "partner-seeking" or "team-seeking" thread were posted on Ed for this purpose.)
Work through interview prep problems
    Hackerrank in general is great - it gently introduces Python concepts through simple problems, and also has non-Python Algorithm tests that are pretty difficult and interesting
Build something cool!
    Glue existing libraries into useful software
Completing Python bootcamp course from Udemy
Subscribe to PyCoder's weekly for a weekly dose of Python news and others surrounding Python
Learning the tools in computer science through MIT's The Missing Semester of Your CS Education
Check out some conference tutorials or talks.
    This year's pycon

Pycon youtube search
Raymond Hettinger's talks - IMO one of the best speakers on python - here are some of my favorites that are great for beginners, so you can watch all of these immediately.

- The Mental Game of Python
- Transforming Code into Beautiful, Idiomatic Python - great to learn language specific features of python if you already know some of c, java, etc this will show you how to write better python, and if you don't then you still learn how to write great python code.
- Being a Core Developer in Python

Awesome Roadmaps - A curated list of awesome software development roadmaps

- Coding Interview University Roadmap - This is a list of free resources to learn the content that a CS undergrad would learn in college
- Web developer roadmap 2019 - good starting places shows you lots of useful technologies
- Go developer roadmap 2019
- UI/UX designer roadmap 2017
- Game developer roadmap 2018
- Deep Learning Papers Reading Roadmap
- iOS developer in 2018
- Mobile developer in 2017
- React developer in 2019
- Vue developer in 2019
- Vue.js developer in 2018
- Node.js Developer Roadmap
- .net back-end Roadmap
- ASP.NET Core Developer roadmap in 2019
- Data science roadmap
- Hacker roadmap
- Software architect roadmap
- Angular Developer Roadmap
- Front-end HTML5/CSS3/Javascript related technologies to learn in 2017
- Software Quality Assurance Roadmap
- Android Developer Roadmap in 2019

Software-adjacent areas

- Design/UX (CS147)
- Product Manager
- Project Manager
- Sales for software

Help a GitHub project

Learn about how a computer executes their code

- NAND2Tetris
- Build an 8 bit computer from scratch

Learn how to build intelligence into real things outside computers

- Arduino tutorials