The Unofficial, Semi-Comprehensive, Quintessential UC Davis Computer Science Major Handbook and FAQ

The Davis Computer Science Club

Version 0.1
March 18, 2017

Contents

1 Version History of this Handbook

Version 0.0

September 14, 2016. First version of the handbook written.

March 18, 2017. Style changes. Made language section less opinionated. Brought version history up to date.

2 Introduction and Disclaimer

I started writing this handbook with a few others because I saw a sore lack of resources for not just the UC Davis Computer Science community, but the tech community in general. More importantly, I knew that a lot of my peers, especially peers who weren't considered the stereotypical Computer Science student, had questions but were too scared to ask them due to fear of backlash and critique.

Like most CS students, I started CS without any Computer Science background and learned everything in this handbook the hard way. Hence, this handbook was written in mind with the audience being a complete newbie (aka a first year with no Computer Science knowledge at all) as a brief overview.

I don't know how helpful this handbook will be or if anyone's attempted anything like this before, but I thought I'd try my hand at sharing my knowledge with anyone who'd be interested. I don't claim to know everything and I'm sure those with more experience could add more nuance and details onto whatever modest amount I have here. If you're interested in adding more to this handbook, please submit a pull request to the GitHub repository where this is hosted. Please feel free to contact The Davis Computer Science Club with any questions!

2.1 How to Best Use This Handbook

There's a lot in this handbook! You don't have to read this handbook cover to cover, word by word (though I'd love it if you did!), but just read whatever sections you're interested in by going to the Contents page and clicking on any section you'd like to read. Any URL to another site is highlighted in cyan blue, and any links/hooks within the document are highlighted in plain blue.

2.2 Disclaimer

This handbook only offers advice and resources given by few members of the UC Davis Computer Science Club – it is by no means a strict guideline for one to follow or a guarantor of success and is not meant to serve as a replacement or substitute for advising by a counselor or adviser. If you have questions, this handbook is a place meant to start your research, not to complete it. While we do our best to make sure our information is accurate and updated, mistakes can be made either from human error or the handbook simply not being updated to match current changes– please make sure to confirm all information with an adviser or through other resources.

If you have questions or comments, or would like to provide suggestions, please feel free to email one of the current officers here: The Davis Computer Science Club. If you would like to add more to the handbook or fix a mistake, please feel free to submit a pull request to the repository where this handbook is hosted. Thank you!

Written with love in LaTeXby:
Stephanie Chang '17

With contributions by:
Alex Fu '17
David Lin '17
Christina Zhu '17
The Davis Computer Science Club

3 The Davis Computer Science Club

The Davis Computer Science club runs during the school-year and provides resources, workshops, and a community for the UC Davis Computer Science community. Here's a breakdown of the Davis Computer Science Club:

Besides the main board, we have three committees – Tutoring, Pragmatic Programming, and Professional Development. Each committee is headed by a chair and their possible, respective co-chair. Our website and various pages can be found below:

Link	Description
Website	Our main page – information about the Davis CS Club, its officers, committees, and various events are located here.
Facebook Group	Chat with other CS majors at UCD and find out about our events
Facebook Page	A little different from the Facebok group – you can find photos and event information here.
GitHub	Our study guides for upper division classes, class notes, meeting minutes, officer board, bylaws, and whatnot are all hosted here.

One of our old committees, the Game Development Committee, has branched off from the Davis Computer Science Club and has created their own club – if interested, please check here.

3.1 General Events

Here are the general events, workshops, and programs the Davis Computer Science Club holds either quarterly or yearly. Some events may or may not be included at the discretion of the current board whereas some events may be added. To keep updated, join our Facebook group or follow us on our Facebook page!

- Bit/Byte Mentorship Program
- Gender Diversity in Tech
- Courses Preview
- Socials
- DCSC End-Of-Year Banquet
- Elections
- T-shirt contest

3.2 Tutoring Committee

The Tutoring Committee serves the Davis Computer Science community by training and providing volunteer tutors for Computer Science (ECS) courses at UC Davis.

Tutors are typically located in Kemper Basement during their signed up hours, but are possibly available upon request. The Tutoring Committee also holds review sessions for lower division ECS courses and a select few upper division courses depending on the availability of the tutors who are able to lead the review sessions.

Tutors are able to receive undergraduate ECS credit with a certain amount of hours. To become a tutor, contact the current Tutoring Committee chair(s) or the Davis Computer Science club.

To find a tutor, please check athena for available tutors during the week, or Kemper 75 or Kemper 77 during CSIF hours on weekdays.

3.3 Pragmatic Programming Committee

Pragmatic Programming tends to lead workshops on various skills, tools, languages, frameworks, and so on forth in tech and Computer Science. They may be spread out in a series of weeks. Past workshops include, but are not limited to:

- iOS and Android Development I, II, III, IV
- Web Development
- Javascript
- Git
- Building Your Own Application
- Optimization

And so on forth. If interested in participating in events, please check out the Facebook group or page to see if any events are running! If you're interested in helping out, please get in contact with our current Pragmatic Programming chair(s).

3.4 Professional Development Committee

The Professional Development provides a series of workshops and events to help students with advancing their careers. Past events have included, but are not limited to:

- Mock Interviews
- Resume Workshops
- Negotiating Offers
- Alumni Talks
- Professor Chats with Professor Rogaway, Professor Gysel, Professor Filikov, and so on forth

And so on forth. If interested in participating in events, please check out the Facebook group or page to see if any events are running! If you're interested in helping out, please get in contact with our current Professional Development chair(s).

3.5 Web Development Committee

Web Development is in charge of our website, daviscsclub.org. They design, host, and code up the website and keep it updated. There are no particular events that the Web Development committee hosts, but people are welcome to attend and help out the committee and website and get acquainted with the code that is involved with our website! If you're intersted, please contact our Web Development chair(s).

3.6 How to Get Involved

Attend workshops, events, and officer meetings, join our Facebook group and follow our Facebook page! Become a mentor a mentee for the Bit/Byte mentorship program, and volunteer at our events! We need volunteers to help out with any large event we plan, such as Gender Diversity in Tech and our DCSC End-Of-Year Banquet. If interested in helping out or attending, message us on our Facebook page. :)

3.7 Officer Board 2016-2017

You can find a description of the officer board in our Bylaws here. Our weekly officer meetings are open to the public – if interested in attending, please message the Davis Computer Science Club on our Facebook page as they are subject to change weekly depending on the availability of the board.

3.7.1 Core Body

Title	Officer	Contact
President	Pooja Rajikumar	prajkumar@ucdavis.edu
Internal Vice President	Stephanie Chang	ischang@ucdavis.edu
External Vice President	Ariel Shin	arishin@ucdavis.edu
Treasurer	Sam Tsoi	sgtsoi@ucdavis.edu
Secretary	Trevor Glynn	tfglynn@ucdavis.edu
Events Coordinator	Sravya Divakarla	sdivakarla@ucdavis.edu
Marketing Director	Prajakta Surve	prsurve@ucdavis.edu

3.7.2 Committees

Committee	Chair	Contact
Tutoring	Alex Fu	aafu@ucdavis.edu
Pragmatic Programming	Pranav Gupta	phgupta@ucdavis.edu
Professional Development	Zain Budhwani	zbud@ucdavis.edu
Web Development	Teresa Liu	stiliu@ucdavis.edu

3.8 FAQ

Are there any requirements to becoming a member of the Davis Computer Science Club?
Nope! Regardless of major or age, anyone is free to participate in our events and workshops. We only ask that you be polite and respectful to everyone else.

Do you charge to become a member?
As of version 0.0 (check the date, please) of this handbook, we do not currently charge to become a member or have any dues. That may change in the future, however, depending on the discretion of the officer board.

Can I post a job posting/advertisement of my product on your Facebook group or page?
For the most part, no. Any job postings should be directed to our Computer Science adviser, Natasha Coulter, who sends out Listserv emails to all the Computer Science undergraduates. Her email is nbcoulter@ucdavis.edu.

If you would really like to post on our page still, message us with your content and we'll take a look! Without approval, any advertisements and job postings will be promptly removed and after multiple times, the user will be banned.

How do I become an officer of the Davis Computer Science Club?

There are yearly elections at the end of the year that decide next year's officer board. To be eligible, you need to attend a certain amount of meetings and/or a certain amount of events. The requirements change depending on the current officer board, so either message the current board on our Facebook page or keep a lookout for the event on our Facebook group or page.

However, students are welcome to shadow officers or attend our officer meetings, which are typically open to the public. Please contact an officer to get more information!

Do you sell t-shirts? Where can I get them?

We do sell t-shirts! Message the current officer board and see when/where they are sold.

Do you sell graduation stoles? Where can I get them?

Graduation stoles are dependent on interest – some years we have them, some years we don't. If enough people are interested, we may order them again. Please contact the current officer board to express interest.

Do you sell Computer Science major sweaters? Where can I get them?

We are not allowed to sell Computer Science major sweaters with the UC Davis logo. However, while we do not have one without a logo, if you are interested in a Computer Science sweater without the UC Davis logo, please contact our current officer board to express your interest. Otherwise, please contact the Computer Science department.

Who can I talk to if I feel uncomfortable at an event or workshop held by DCSC?

Please contact the Internal Vice President – they handle all the internal affairs of the club. DCSC has a zero tolerance policy against harassment and discrimination. Please let us know if you feel uncomfortable or harassed at an event and we will do our best to to stop or prevent the harassment.

Where can I find the meeting minutes?

Our meeting minutes are on our GitHub page, under Archives. They are typically updated by our Secretary on a semi-weekly basis.

How can I contact the Davis Computer Science club?

We prefer being contacted through Facebook via our Facebook page. However, you are welcome to contact the current board, especially our president, for any questions, comments, or concerns. The Davis Computer Science Club welcomes any sponsors and if interested, please email or message us!

4 Classes and Waitlisting

4.1 Helpful Links

4.1.1 Advising

Link	Description
CS/CSE Major Requirements	Various major requirements and catalogs live here.
Quarter Deadlines	Deadlines for the quarter and drop deadlines.
OASIS	Calculate your major GPA, GE requirements, change and add majors, schedule advising, or add a minor here.
College of Letters and Science Degree Check	Get a degree check if you're in the College of Letters and Science! Submit a form and you'll get a degree check by an adviser returned to you in a week or so.
College of Engineering, Degree Check	Get a degree check if you're in the College of Engineering! Submit a form and you'll get a degree check by an adviser returned to you in a week or so.

4.1.2 CSIF, UNIX, tutorials

Link	Description
Computer Science Instructional Facility, or CSIF	Information about our basement labs and our computers and hours are here.
UNIX and CSIF set up	Need help setting up or learning UNIX? Lecturer Sean Davis has an excellent tutorial here, along with a PDF of useful bash commands.

4.2 Class Dependency Tree

Figure 1 shows the prerequisites needed for CS classes at UC Davis.

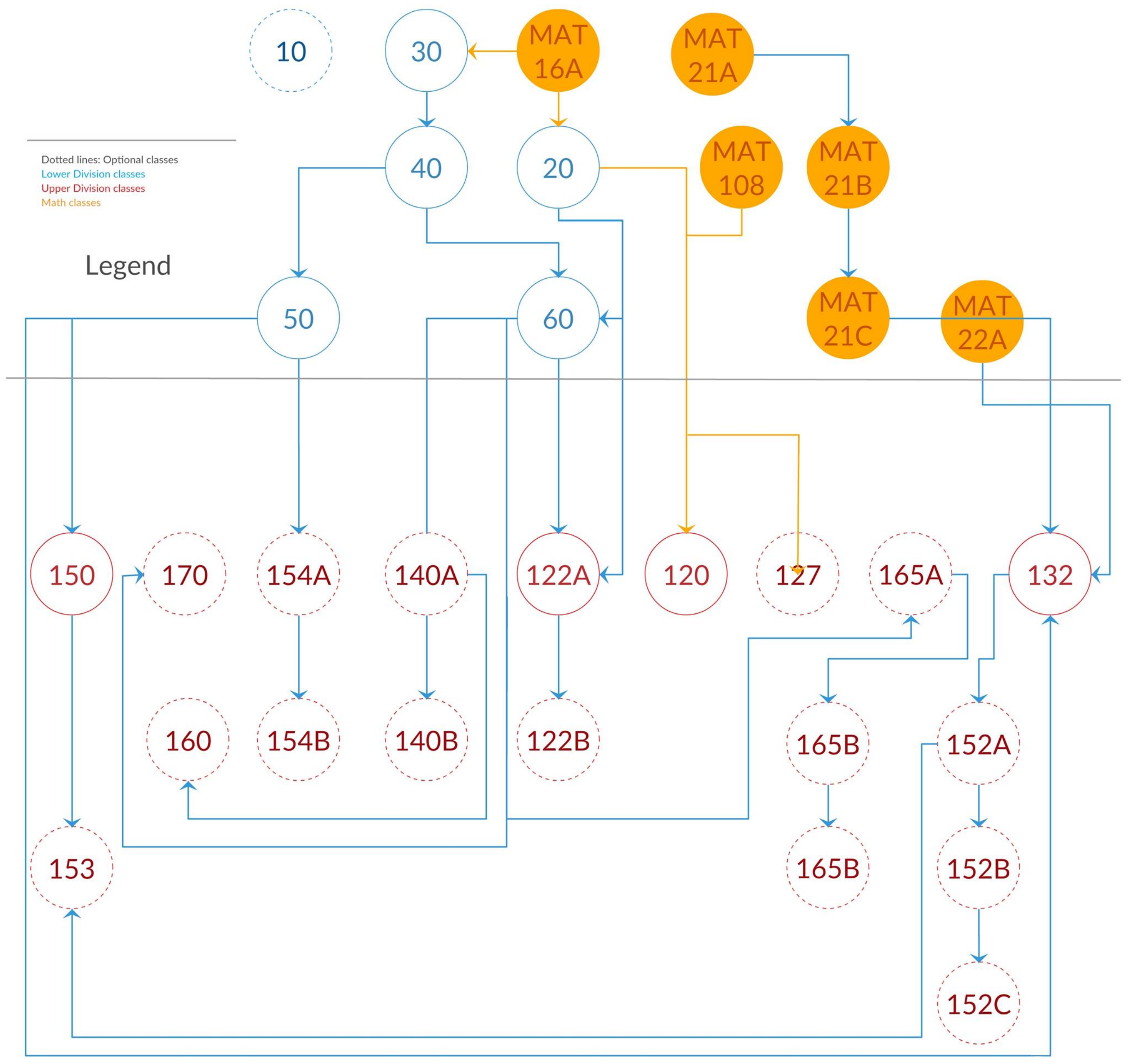

Figure 1: Class Dependency Tree made by David Lin, '17.

4.3 FAQ

What are my chances for getting into x class if I'm position y on the waitlist?

There's no definitive answer for this question. Since CS and CSE are now considered impacted majors, it's probably going to be difficult to get into classes, especially if you're not a declared CS/CSE major.

There are some known classes that are more difficult to get into even if you're high on the waitlist, such as core upper division classes and especially ECS 188. If you're a CS major and in need for upper division electives, math upper divs (besides 111) are now an option to replace an upper div elective since catalog 2014.

Most lower div classes, such as ECS 30, ECS40, and ECS 60, are easy to get into regardless of your position on the waitlist given its high drop percentage and turnover, but there's still no total guarantee. Make sure you have backup CRNs stored up and research other alternative before your pass time!

Can I follow a graduation requirement catalog that's not my own?

You can only graduate from catalogs that have been changed since you started UC Davis – for example, if Jill started UC Davis in 2013, Jill can follow any catalog changes to graduate after she came in, such as 2014, 2015, 2016, etc. However, Jill is not able to follow any catalog changes from before she came in – so 2012, 2011, etc are out.

Is my w x y z class load manageable?

That's hard to say. Common classes to take together have been ECS 40 and ECS 20, along with ECS 50 and ECS 60. Depending on professors, some class loads may be more manageable than others.

Where can I find tutors for ECS? classes

Please check out athena, our tutor office hours page.

How do I get credit for my ECS internship/tutoring/research?

If you're doing research or an internship with a professor or company, click here for more information to possibly get credit. If you'd like to tutor with the Davis Computer Science Club and get credits, please either contact the current Tutoring Club Chair at Officer Board 2016-2017 or click here.

How can I prepare for x class?

The best way to prepare for a class is look up the class and find the syllabus. If it's an intro class, try looking up the projects that were done and code up the solutions on your own. ECS 10 uses Python, ECS30 uses C, and C++ is used throughout most of the classes after, such as ECS 40 and ECS 60, and other upper division classes.

If you're just starting to take ECS classes, a good way to start is to familiarize yourself with the language and learn the basics: if/else statements, loops, arrays, pointers, and possibly structs or classes. KhanAcademy, Coursera, CodeAcademy, and StackOverflow are good resources for beginners and professionals alike.

The various data structures taught in ECS 60 are usually: linked lists (single and doubly linked), stacks, quadratic probing and linear probing hash tables, queues, arrays, and the like.

What are some easy GE's?

Popular GE classes mentioned have been NUT 10, ASA 1/2/3/4, ENL 5F/5P, CLA30, ECM1, COM1/3, ECN1, etc. Alternatively, localWiki has a list of classes here.

What are some easy or popular math/stat classes to take for that one upper division math elective?

MAT 145 (Combinatorics) and MAT 108 (Discrete Math) are popular options.

What's a good GPA to have for a CS/CSE major?

There's no good answer for this question – it all depends on what you plan on doing after school.

If you plan on going into industry, certain companies may care more about your GPA than others and you may be guaranteed interviews if your GPA is high enough and above a certain cutoff. However, most companies stress personal projects and experience, whether from hackathons, work, or in your own free time, and your GPA isn't much of a deciding factor, if at all. It doesn't hurt to have a high GPA, but in general, it matters more to have projects and experience. From the unconfirmed grapevine, it's been said that anything above a 3.0 is considered decent or competitive, and the few, rare companies that have a GPA cutoff require a 3.0 or above.

Graduate school has different requirements and standards and may stress GPA more heavily than industry, but that depends on who you are, which school, and the research you've done.

The average GPA for a CS major seems to be around a 2.9, so take with that what you will.

What's the best OS/computer/device/etc to own to code with?

Any operating system will do, and any computer will do. None are the best. The CSIF lab computers run Fedora (one of many Linux distributions), so you may want to try that.

Note that you can run whatever operating system you want inside a virtual machine using software including, but not limited to, QEMU and VirtualBox.

5 Hackathons

“Hackathon” is a conjunction of two words: hack and marathon. Participants can either go solo or form teams and create projects for a certain amount of time, anywhere from 12 to 72 hours. At the end of the hackathon, there’s usually a demo session where projects are judged and the best projects take home great prizes.

Hackthons are usually hosted by universities or organizations with various company sponsors. Most hackathons are free and provide wi-fi, sleeping areas, food, mentors if you get stuck, and plenty of swag* for everyone to take home. Sometimes hacakthons will provide a certain amount of travel reimbursement for the participant to attend the hackathon.

Hackathons are a good way to buff up your resume with projects and expose yourself to more technologies, APIS, and companies. It’s also a way to network with other professionals, students, and recruiters in the field. Some companies will recruit and even interview at a hackathon, so if you’re looking for job opportunities, keep your resume updated and bring a few hardcopies!

Contrary to popular belief, anyone can attend a hackathon, even if you’re not a computer science major or student. A successful team may need designers, product management leads, and business majors to keep the project going and even if you don’t plan to hack on/make anything at the hackathon, attending one and immersing yourself in tech culture is a great learning experience for anyone, so don’t count yourself out!

5.1 A Few Major hackathons

Location	Name	Theme (if any)
UC Davis	HackDavis	social good
San Francisco	Stupid Sh*t	useless hacks
	HackTheLeft	leftist hacks
	Slash Hack	
UC San Diego	SDHacks	
UC Berkeley	CalHacks	
UC Santa Barbara	SBHacks	
Stanford	TreeHacks	
Los Altos	LosAltoHacks	high school students
San Mateo	HackingEDU	education
NorCal	Spectra	women only
Palo Alto	GunnHacks	high school students
MIT	HackMIT	
Duke	HackDuke	social good
University of Waterloo	HackTheNorth	
LinkedIn	Hackday	bay area interns only

5.2 Other

Other places to discover hackathons in your area are: Major League Hacking EventBrite

*free goods (branded or otherwise) from companies or universities, like tote bags, blankets, water bottles, t-shirts, stickers, toothbrushes, etc.

5.3 Tips

Sleep A lot of hackathons stress not sleeping, but four cans of Redbull and working for 9 hours straight can really put a toll on your health and productivity. Find a place to sleep, set breakpoints throughout the hackathon, and just take a shut eye. You'll thank yourself later.

Hardcode No one comes out of a hackathon with a perfect project. Most of the prototyping comes from hardcoding a bunch of the logic – a successful hack doesn't come from a perfect implementation, but a good pitch, idea, and appearance. No one, not even the judges, is expecting a perfect product.

Winning isn't everything You don't have to win a prize in order to call a hackathon succesful for you. If you made something or even learned something, it was a successful hackathon.

Network Communicate with your team, and other teams, and help each other out! The best part of the hackathon is checking out the demos and seeing what other people have made and talking to other developers about their interests and hacks. It's the best way to learn and expose yourself to other areas in tech.

Attend workshops and make use of the mentors Most, if not all, hackathons will have a few workshops and mentors. Make use of them – they're here to help you get through your bugs and guide you through your learning process.

Hackathons Demystified Page 29 has a list of terms that you can look over to get familiar with jargon, and has a few other tips to get you prepped!

5.4 FAQ

I'm not 18+, can I still attend a hackathon?
Yes, you can! There are plenty of hackathons that allow attendees other 18. There are high school hackathons and hackathons that allow minors, such as /hack. Keep a lookout for them!

How much does it cost to attend a hackathon?
Hackathons are usually free, unless otherwise specified. Professional hackathons can cost money, so contact the hackthon's organizers for more details.

What should I bring to a hackathon?
Your laptop, charger, ID, a creative spirit, and anything else you need to hack with! If it's an overnight hack, blankets, sleeping bag, toothbrush/toothpaste and any hygienic products would also be recommended.

I've never coded before/I'm a beginner/I'm scared to drag my team down – can I still go to a hackathon?
Yes, yes, yes! Hackathons are for everyone regardless of experience. In fact, beginners are highly encouraged to attend. If youŕe worried about not knowing anything, find likeminded beginners and form a team with them. Hackathons are all about learning, so don't be afraid if you don't know anything!

6 Applying For Jobs and Internships

6.1 How and Where to Apply

There's a few way to apply and score Interviews.

1. Cold Applying
2. Referrals
3. Career Fairs

Cold applying is going to each company's website or portal and applying. The top ones to hit up every year are always the major companies: Google, Facebook, Amazon, Apple, and Microsoft. Always apply, even if you don't think you have a chance – they'll keep your resume and profile in their archive for a while and, in the future, they might look you up and reach out to you! If you think of a company you might want to work for, look them up and apply to them!

Tip: Keep a spreadsheet of the ones you've applied to, the positions you've applied to, the date you applied, and the status of your application. If it's been a few days since you've heard from your recruiter, make sure to follow up!

Before you apply, however, you're probably going to need at least a resume, and maybe a cover letter.

6.2 Resume

The sections of a tech resume can be broken down like so:

1. Name and Contact Information (email, phone, GitHub, LinkedIn, website)
2. Skills (languages, libraries, frameworks)
3. Projects (school, hackathons, personal)
4. Experience (mostly tech, if possible, and/or leadership/volunteering)
5. Education (major(s) and minor(s), graduation year, GPA optional)
6. Relevant Coursework (optional)

You should definitely include your Education and GPA, the latter being optional and only if your GPA is decent, but neither are particularly interesting unless either are truly exceptional, so put your skills, projects, and experience at the top as much as you can.

Relevant Coursework is also good section to also add, especially if you're looking for an internship or full-time position and don't have much experience under your belt. Any CS classes you've taken or are projected to take in the next quarter, especially Data Structures and Algorithms, are always good classes to add.

6.2.1 Building Your Resume

If you haven't taken many classes or have skills or relevant experience (which most people don't when they try and find internships or jobs, so don't worry), you're going to have to start by building projects.

You can build projects by going to Hackathons, making your own website or app, contributing to open source projects, etc. It doesn't have to be big. You can make a Unitrans app, a Tic Tac Toe app, a Google Chrome extension, your own website, or anything simple (or complicated) to learn and show initiative and incentive. See below at Project Ideas and APIs for more ideas.

You can also build your resume by taking more classes. Your UC Davis classes aren't the only ones you can add to your resume – in your free time, taking an online Coursera class or two and finishing the course is also something you can add to your resume. Keep in mind that some Coursera classes are free, and some are not.

Please also check Online Presence to build your resume and online presence.

Even if you don't have many projects, classes, experience, or skills, it doesn't mean you shouldn't still try your luck at applying! It's always good to gain experience in applying for jobs and networking.

6.2.2 Resume Tips

- If you have a lot of experience, choose a few and tailor your resume for the job you're applying for.
- The general rule of thumb is one page for every ten years of relevant experience and if you're applying for a new grad or internship position, keep it to one page. If a recruiter can't skim your resume and read everything quickly, then it's most likely going to be tossed or archived.
- Black type on white paper. Size 10 font, minimum. Half inch margins, minimum. Equal borders and spacing for all items. Your wording should be concise, legible, but have brevity. Your resume can be pretty, but it doesn't have to be a work of art. Keep it straightforward and neat – all your information should be easily found.
- Stretch your content across the page – white space should only be used to separate your content for legibility – you don't need splotches of white space on your page for no reason. It makes it look like you don't have enough.

  You can always fill up your resume with something, whether skills or projects or coursework or experience. If you don't have any of those, learn them, build them, take them, get them.
- Other items you can include in your resume: awards, honors, scholarships, fellowships, publications, research, interests, etc.
- You don't need to put communication, leadership skills, or anything similar as your skills – that's either evident through any large scale team projects you've listed or it'll be evident enough enough during a phone and onsite interviews.

  There's also no need to put Microsoft Word, Powerpoint or anything similar as your skills unless the position specifically asks: most people know Microsoft Word and if not, they can quickly learn.

  There's also no need to put Objective as a section either – it's outdated. References is also not a needed section if you're going to write "Available upon request" – recruiters will ask for a reference or a background check if they need one from you.
- Microsoft Word with tables is the most commonly used software to make resumes, and some people also choose to use LaTeX
- Formats can change with every file extension and version, so try to keep an edited version (in Word or LaTeX) and send recruiters the PDF version of your resume.
- Don't put your address. No one contacts anyone for job opportunities via address anymore and if recruiters need your address, they'll ask. Leaking your address is a dangerous potential for identity theft – your phone number, email, and any social networking sites (LinkedIn, GitHub, your website, etc) should suffice as contact information.
- Keep descriptions of each position you've held short – two to three bullet points at most, with a line each. A good description should be like what Churchill said about speeches and skirts: "long enough to cover everything, but short enough to be interesting".

- If possible and space permits, it's also nice to include one position you've held that is not tech related – it shows that you have interests outside of tech. For example, if you volunteer at a children's center for reading, or held a leadership position in a running club.
- Check for typos! Don't lie! These are the big ones that can be easily fixed!
- Keep your name large, nice, and bold – it makes you memorable.

6.3 Cover Letter

A general template for a cover letter looks something like this:

1. Dear XYZ,
2. Your name, introduction, the position you're applying for, how you heard about the company (especially if it's through referral), and anything you know about the company.
3. A short discussion about your relevant skills, projects, and skills and what you can bring to the company technical skills wise. What challenges you think the company faces and what interests you about them, and how passionate you are about their product. Do some research here.
4. An optional paragraph on your leadership skills, collaborative skills, various interests, and what you can bring as a leader or team member to the company.
5. Ending note, conclusion, and state that you believe your attached resume (or website, or whatever you choose) will highlight any of your other qualities. Thank them for their time.
6. Sincerely, Your Name

You shouldn't have to write a brand new cover letter for every company you apply to. Your cover letter should be a malleable template and only changed a little for every company – keep the main details the same, such as your skills, but do some research and add or change details about the company you're applying to as you see fit.

Your cover letter should also be one page, and should contain the same header as your Resume (Name and Contact Information).

6.3.1 Cover Letter Tips

- If they ask for an optional cover letter, include your cover letter. You're going the extra mile to show them you want the job – do it.
- Check for typos! Don't lie! Be consistent with your resume!
- Don't rote list your resume out – your cover letter is supposed to add to your resume, not be a summary. Add details that can't be seen on your resume, such as what you enjoyed most about a project or class or what kind of technology stack you've worked with and are learning.

6.4 Referrals

This is the best way to get an interview. Your application is sent to the top of the stack and recruiters will review referred applicants first opposed to those that cold apply. It's sad, but true. Network, network, network – make friends in your classes, make friends at hackathons, make friends at work. These people will get jobs and these are the people who can push your career forward someday.

It's also, well, just nice to have friends in tech. :) And don't forget to return the favor and refer them when you get a job as well!

6.5 Career Fairs

UC Davis has several career fairs in a year, and one specialized just for engineering, with a list of them here: Career Fairs.

If you're actively looking for a job or internship, go to career fairs! Dress business formal/casual, print 15-20 resumes, research the companies you want to talk to, get there early if you can, and start hitting up each recruiter and talking to them. Even if you're not looking for a job at the moment, you can just go and pick up all the swag the companies are handing out.

One thing to do before a career fair is to practice your elevator pitch. It's a 1–2 minute spiel about yourself that goes something like this:

> Hi, my name is Jill, I'm a 2nd year and I study Computer Science and Engineering at UC Davis. I'm really interested in your Test Automation Internship position. I've known about X company for a long time and I'm very passionate about your product. Can you tell me more about yourself and the various positions and X company?

6.6 Online Presence

The first thing a recruiter does is look at your resume. Then they either see if you have any more information (website, LinkedIn, GitHub, Devpost, etc) on your resume that they can check online for and if not, they Google you. You're going to seem less appealing than a candidate who has a fleshed out online presence that the recruiter can get to know, so building your presence as soon as you start applying for jobs is key.

You should make a:

1. LinkedIn
2. GitHub

and optionally, a website or portfolio.

Your LinkedIn should not only reflect your resume, but anything else you didn't have a chance to add on your resume, such as classes, more details on any projects you've done, other work or volunteering experience, etc. It should be detailed and thorough, with any images or links to projects you have, along with any coursework you have. Recruiters will often look at your LinkedIn and may contact you from there.

Your GitHub should have all your personal projects and code samples. If you don't know what to add, try solving some interview problems on sites such as HackerRank or Project Euler and putting it in a repository.

Your website, if you decide to make one, doesn't have to be fancy – it can be merely a static page with your name, your email, your LinkedIn and GitHub and any relevant websites. You can talk about your interests, volunteering, work history, hackathons you've attended, projects you've made, etc. It's your website, but also your portfolio and a chance to show off your technical skills. See more at Website/Personal Portfolio.

Other ways you can build your online presence is being active in various other communities, such as StackOverflow or StackExchange, or contributing to various open source communities.

Note: The Davis Computer Science Club does not advise putting projects done from classes on GitHub – it is unfortunately common for students to cheat and plagiarise from code past students have made available online.

6.7 Website/Personal Portfolio

If you've never built a website before, here's the place for you! The easiest place to host your website is at: GitHub Pages and they have a convenient and simple tutorial.

A few suggestions to drum up your website are to use Twitter Bootstrap for the styling. Bootstrap is merely a set of CSS stylesheets you can use from, and you can download it from the link given or use any of the CDNs provided (more on the link).

6.8 Negotiating Offers

You should always try and negotiate your base pay, stock options, or benefits. In fact, most companies expect you to.

If you have another offer that's paying you more, you should definitely ask if the offer you're looking at if they can possibly match or exceed the offer of the other company.

You'll get more leverage if you have another offer, but by itself, you can leverage your skills and ask for more. Just simply ask your recruiter (nicely!) if they can possibly raise your base salary/benefits/etc since it seems a bit low. Most people seem afraid they're going to seem like an ungrateful money-grubber – you're not. It doesn't matter if you like the company or the product, there's nothing wrong with having more money and there's no harm in asking: you already got the offer!

If there's a certain salary/benefit/stocks combination you should definitely not go below, a number that you should determine by yourself based on research, try and keep to it. If you know the recruiter is offering you too low of a sum and they refuse to raise your offer, decline and interview elsewhere. Don't shortchange your worth just for the experience. You'll get other offers.

6.9 Rejection

It happens to everyone. They say that the first rejection is the hardest, but I find that all the subsequent rejections hurt as much, if not more. But it literally happens to everyone – the best engineers get rejected, and often. No one gets to where they are without it.

Some companies won't send a rejection letter. Some will. Both of them suck. Sometimes you won't even know why they don't want to hire you, and they could be for completely arbitrary, stupid reasons, like the recruiter just had a bad day or they misplaced your paperwork. It could be that another candidate was just too good – maybe they were a student from MIT with 22 papers under their belt and they've been coding since they could walk how could you possibly compete with that? And yes, they can be for technical reasons: your solution wasn't optimized enough, you couldn't find the edge cases, your skills aren't developed enough. Most companies won't tell you.

Take it as a learning experience to grow more. It's okay to cry. It's okay to feel like crap for a few days. But in the end, it's just another experience. It doesn't mean you're not going to get a job or that you can't apply again. It's not personal. You're smart, you're funny, you're likeable, and you're going to make it in the end.

6.10 Resources

List of Easy Applications

6.11 Tips

- Recruiters are people, too. Be nice to them. Thank them, and often. Thank your interviewers, too. Err on the side of being too polite, always.
- Follow up with your recruiter – if they haven't responded in a few days to a week about your interview or the next steps, email them again. And again. Call them, email other people from the company, and leave messages if you have to.
- A few good questions to ask your interviewer (pick and choose), technical or otherwise, at the end of the interview are:
  - What do you like about X company?
  - How is X company different from other companies?
  - What team do you work on at X company? What do you do?
  - What's the culture like at X company?
  - What's the technology stack at X company/your team like?
  - How flexible is X company with changing teams or learning new things?
  - What are some skills or values every intern/employee should have at X company?
  - What are the next steps of the process?
- Attend conferences or hackathons not just for the learning experience, but to network!
- UC Davis College of Engineering offers a few industry tours a year (such as Square, Google, Apple, etc), so if you're interested in taking a look inside at some pretty great companies, follow their Listserv or contact COE! This is open to UC Davis students based on a first come first serve basis.

6.12 FAQ

Should I put code I've written for class projects on my GitHub?

That's up to you. DCSC does not advise or endorse this, however – while it's a quick way to show your code snippets and many students do so, but at full disclosure, other students may look up your code and copy, so you might be at risk with the SJA. Posting your professor's code that you built upon and calling it your own may be considered plagiarism, so if you are going to do so, you might want to put a disclaimer.

When should I start applying for jobs and internships?

There's no 'should' or supposed timeline – just whenever you're comfortable! However, you can apply for internships starting your freshman year if you'd like. It's recommended to only start applying after you've taken ECS 60 Data Structures or already have a foundation in Data Structures, as most of your interview questions will be based on that.

7 Interviews

Most interviews for software engineering or related positions are split between two kinds of questions: behavioral and technical.

7.1 Interview Formats

Coding Challenge You'll be given the challenge in the form of a link, such as HackerRank, where you have to complete a certain number of problems in a given amount of time (30 minutes to a few hours, usually) by a certain deadline.

Make sure your laptop or computer is charged and you find an ideal and quiet place with internet to work and turn off any distractions for the amount of time you're interviewing. Once you start, you can't pause, so make sure you're ready.

Remote/Phone Interview This can either be a behavioral or technical interview and it's usually done via phone or video chat. There might be multiple rounds, sometimes back to back or spread out through several weeks or months. Always be prepared for both types of questions as you never know what they're going to throw at you.

If it's a technical phone interview, interviewers may ask you questions straight up, or ask you to code up a more complicated problem on a shared document such as GoogleDocs or CodePad. You may have to run the code after you're done, so code carefully if you don't want to spend most of your interview debugging.

Onsite If you were invited to an onsite, you're probably close to the end! Be prepared to spend a few hours/the whole day at the onsite location, and get there early if you can. An onsite usually has whiteboard coding and debugging the code with the interviewer in the room.

Depending on the company, you might interview with other candidates, have back to back rounds, have multiple onsites, or just have one interview. Either way, be prepared for the worst and practice, practice, practice.

7.2 Common Interview Questions

- Behavioral
- Technical

7.3 Resources and Practice

Title	Description
HackerRank	various interview questions from trees to data structures to database questions in any language you choose
GeeksForGeeks	lots of interview questions and their solutions in different languages
ProjectEuler	programming and math challenge questions. Google's advised website to check out for programming questions.
LeetCode	various problems and their solutions (with some digging)
CareerCup	programming problems and solutions in a Stack Overflow-like format
Reddit Programming Questions	problems on Reddit and people answering solutions, ranked easy–hard
Glassdoor	look up the company you're interviewing for and various questions they may have
Cracking the Coding Interview	a book with programming questions, hints, and solutions

- Big-Oh
- Interview Cheat Sheets

7.4 Tips

Think aloud If you're given a problem, think aloud and explain your thought process before starting. Don't be afraid to ask your interviewer questions if you're stuck or don't understand the problem – they don't want to see just your speed/efficiency in coding, but whether you're able to collaborate with other people to figure out the problem.

Optimize You're probably going to be asked what are the edge cases and Big-Oh of your solution and whether you're able to optimize your code further. When you're practicing for interviews, make sure you ask yourself these questions and try and optimize your code as much as you can. When in doubt, see if you can use a hash map or hash table.

Bruteforce, then optimize If you're truly stuck on a problem and how to get it to a nice optimization in regards of speed and memory, think of the worst solution you can then start from there and ask yourself: what's the bottleneck (slowest part/congestion) of this solution? How can I optimize this further?

Sleep This goes without saying, but get plenty of sleep before your interview. It's hard to code when you're nervous already; there's no need to add sleep deprivation to the mix.

Practice with friends Or go to Mock Interview sessions! Simulate an interview environment as much as you can and take turns asking each other questions, both behavioral and technical. It'll help you get into the rhythm and the types of questions asked.

Choose a language you're comfortable with If you're not sure which to choose, the most common language I've seen people interview with have been Python and Javascript. They're both clean scripting languages with simple and easy-to-remember syntax and implementing data structures is pretty cake with both. However, you may not always get a choice in what language you can interview in and some interview questions may be better written in other languages. Play to your strengths.

Don't neglect the behavioral A lot of companies stress culture fit and people want to hire who they want to work with. Learn how to pitch yourself and practice not just your marketing your skills and projects, but who you are as a person. Don't be afraid to talk someone's ear off about your hobbies and anecdotes – chat with your interviewers. They're people too.

Don't stop practicing If you plan on entering the tech industry in the future and continuing with it, you're going to be interviewing quite often. Always refresh yourself on your data structures and problem solving skills, and keep with it.

7.5 FAQ

There's so much material to cover, where should I even begin?

Start with reviewing any material you have from ECS 60 (Data Structures). Most websites or books about interviewing begin with easier questions so progress from there – when you've gotten the hang of solving simple questions, try out a few moderate-hard questions, think of your own solution and optimization, test your solution, then look at other people's answers.

Some times the best way to learn is to look at some of the ways other people use to solve their problems. After looking at their solution, code it up in a different language and test it. Then another. Keep going. Rinse and repeat.

What should I know for an interview?

General knowledge everyone should know: data structures – arrays, hashes, heaps, lists, queues, stacks, trees; strings; graphs; bit manipulation, Huffman encoding, and the like. It also depends on what position you're interviewing for.

For example, an interview for a front-end position may ask you primarily HTML/CSS questions. In fact, any skill you put on your resume as something you know or proficient in is fair game.

How should I dress/look for an interview?

Business formal at best, business casual at worst, unless otherwise specified. If it's a video interview, dress up as well, at least from the waist up.

8 Technical Skills

Some technical skills, languages, and frameworks that are out there and trending right now! These are only a subset of technologies that exist, so if you're interested, feel free to do some research on your own. :)

8.1 Languages

Depending on what part of software you'd like to get into, some may be more relevant or useful than others. You do not need to know ALL of these languages to be considered competitive! These are also just a small list of popular languages used today. Please Google them for more information.

Language	**Description**
C	Low-level language. You'll usually touch C if you're writing firmware or directly interacting with hardware.
C++	Originally based on C and Simula. A very featureful language used for a wide variety of applications, especially performance-critical ones.
Java	Popular cross-platform programming language. Code once, run using a virtual machine in all major platforms. Has a large overhead due to this fact.
JavaScript	The scripting language in your browser. Can be emulated on your desktop. Worth learing if you want to do web development.
HTML and CSS	The staples of web development. HTML is for creating structures, CSS is for applying styles to those structures.
Python	Quick and easy programming language. Easy to prototype functions, perfect for job interview questions. Become a Python 3 master and you won't regret it!
SQL	Database manipulation language. Used to insert, update, change, and retrieve data inside a database. A staple of backend developers.
Ruby	Similar in purpose to Python. Draws inspiration from Smalltalk and Perl. Became popular through the Ruby on Rails web framework.
PHP	Designed for web applications. Excessive use of the dollar sign for variable names will make you feel filthy rich.
Perl	Command line scripting language. Can be used in conjunction with command line tools to manipulate data or files.
R	If you're into math, stats, or physics, (or your name is Matloff) R is your best friend! It's the perfect language for your graphing or calculation needs.
Swift	A safer, more modern alternative to Objective-C. Used to create apps that work on Apple products.
Objective-C	Based on C and Smalltalk. The main language used for programming on Apple operating systems until the invention of Swift.

If you'd like to know more, other people have written better lists and descriptions here.

8.1.1 Caveat

Languages are not merely tools. Many languages are worth learning not because they are popular or useful, but because they will teach you new ways of thinking. Curious programmers might like Forth, Prolog, Erlang, Curry, Lisp, Mercury, Smalltalk, Haskell, Coq, and others.

8.2 Text Editors

There's a lot of argument on what's the best text editor to use, but here are some popular ones. Optionally, you can choose to use an IDE. Here are a few!

- Sublime
- Emacs
- Vim, vi
- Gedit
- Atom

8.3 Frameworks

Language	**Frameworks**
Java	Javaspring Spark
Ruby	Ruby on Rails Sinatra
Python	Django Flask
Javascript	Node.js React Angular.js
HTML/CSS	Bootstrap Semantic UI

8.4 FAQ

What language should I learn first?

It honestly depends on what you'd like to get into! Most people start off either learning C or Python, as they're good beginning languages to learn from, but honestly any language is a good start.

9 Building Projects

9.1 Project Ideas and APIs

Keep in mind these are meant to be simple ideas to get you started with creating a project on your own. Most, if not all of these, can be either developed into a phone application or a web application, or whatever you prefer. Get creative! They can be quirky, fun, or interesting. :)

Any ideas from other people are meant for inspiration only – do not take their code and claim it as your own. That's lying and famously called plagiarism, people.

9.1.1 Project ideas or inspiration

ideas

- a tic tac toe application
- a unitrains/bus application
- a spellcheck application
- a Pokemon weakness/strengths application
- a basic social networking site
- bill splitting for roommates
- a calculator
- an exam grade calculator
- an app that tells you if you should wear shorts or pants in the morning
- a weather app based on location
- project ideas, with examples

inspiration:

- 180 websites in 180 days
- checking weather with Drake
- website that tracks the time each gender speaks
- chrome extention that changes all instances of cloud on the page to butt

9.1.2 Useful APIs

Public and Free APIs

10 For Marginalized Folks in Computer Science

A few resources and clubs exist for those not commonly represented in the tech industry.

10.1 A Few Clubs On Campus at UC Davis

Name	Identities	URL
Society of Women Engineers (SWE)	women, engineering, CS included	swe.engineering.ucdavis.edu/
WiSTEM	women, stem	facebook.com/wistemucdavis/
Pilipinx Advancing Science Engineering (PASE)	pilipinx, STEM	joinpase.weebly.com/
Black Engineers Association	black, engineering	http://beaucdnsbe.weebly.com/
The American Indian Science and Engineering Society (AISES)	native american, STEM	http://aises.engineering.ucdavis.edu/
Chicano and Latino Engineers and Scientists Society (CALESS)	chicanx/latinx, STEM	caless.engineering.ucdavis.edu/
Queers in Science Club	queers in STEM	N/A, check with LGBTQIARC
BlackOut	black, queer	N/A, check with LGBTQIARC
La Familia de UC Davis	latinx/chicanx, queer	facebook.com/groups/2234329547/
Asian Pacific Islander Queers	AAPI, queer	facebook.com/groups/APIQatUCD/
Queer Student Union (QSU)	queer	N/A, check with LGBTQIARC

10.2 Centers and Counseling

Alternatively, the centers on campus are also helpful for general needs. There are CAN advisers at every center that can aid y'all with counseling and advising. The CCC, LGBTQIARC¡ and SRRC are located in the Student Community Center at UC Davis. The WRRC is located in North Hall.

Name	URL
Cross Cultural Center (CCC)	ccc.ucdavis.edu
Women's Resources and Research Center (WRRC)	wrrc.ucdavis.edu
Lesbian, Gay Bisexual, Transgender, Questioning, Intersex, and Asexual Resources Center (LGBTQIARC)	lgbtqia.ucdavis.edu
Student Resources and Retention Center	srrc.ucdavis.edu
Student Counseling	shcs.ucdavis.edu/counseling-services

10.3 Other Resources

A few other resources and organizations on campus can be found at the following links:

- UC Davis African American and African Student Resources

- UC Davis Chicanx/Latinx Student Resources
- UC Davis Multicultural Student Resources
- UC Davis Queer Student Resources
- Resources for Women in Tech
- People of Color in Tech
- Grace Hopper: Celebration of Women in Computing Conference
- Grace Hopper Conference Scholarships List

Hackathons Cheatsheet

Terminology: *Demystified*

Term	**Definition**
API	*Application Programming Interface* — a set of functions that allow a programmer to access a service. For example, Venmo provides an **API** that lets programmers write apps that accept payments via Venmo. If your car had an API, it would let you "honk", "turn on the windshield wipers", "return the number of gallons of gas in the car", etc.
Backend	The part of the application that the user doesn't directly see or interact with. This is usually made up of a server and/or database.
CSS	*Cascading Style Sheets* — a simple programming language used to style websites. Lets you change text size, color, format images and layout, etc.
Database	A database is any software that lets you save and retrieve data. Parse is an example of a free web-based database that provides an easy-to-use API.
DevPost	A website that lets hackathon participants show off their completed hacks. Also sometimes used for submissions during the event.
Frontend	The part of the application that user interacts with. For example, the frontend of a website is made up of the HTML, CSS, and JavaScript.
Git	Git is a tool that lets you maintain a history of the code in your hack, and synchronize your codebase across your teammates' computers.
GitHub	If you use Git, you can use GitHub to host your code online for free and easily share it with the other members of your team, and (if you want) the whole world!
Hardware Hack	A hack that involves the use of hardware other than a laptop. Generally these involve knowledge of electronics, such as motors, sensors, lights, and wires.
HTML	*Hyper Text Markup Language* — used to build the layout of websites
JavaScript (JS)	Programming language used to make websites interactive (frontend)
Terminal	A terminal is an application already installed on most computers that lets the user execute commands and run programs, such as Git

more on the back!

Tips & Tricks

- Within the first couple minutes, focus on finding a good table with outlets for your team… the tables go fast! Drop your stuff off in a safe place and go get swag from sponsors.
- Stay away from oily foods (get the salad instead of the pizza!), and don't overdose on caffeine. Many hackathons actually have healthy, good-quality food these days!
- Take "micronaps" to sustain productivity. Take turns sleeping! Nothing is better than waking up and seeing your teammates accomplish a major task or fixing a huge bug.
- Creation an actionable list of tasks and a reasonable time estimate, re-evaluate often!

Resources

Finding Hackathons

mlh.io/events
hackalist.org

Stay in the Loop

mlh.io/events

facebook.com/TheMellonHeads

mellonheads.org/list

mellonheads.org

Everything Else

post in the MellonHeads Facebook group!

GOOGLE IS YOUR FRIEND