



HOW TO LAND A JOB IN TECH

**Ava Meredith, B.S.C.S,
MBA**

TABLE OF CONTENTS

	INTRODUCTION
1	THE INTERVIEW PROCESS
2	SKILLS ASSESSMENT
3	THE TECH JOBS LANDSCAPE
4	RESUME
5	COVER LETTER
6	ONLINE PRESENCE



7

TECH JOBS SEARCH

8

ACING THE CODING INTERVIEW

8.1

An Overview of the Coding Interview

8.2

Programming Languages and Technologies

8.3

Programming Fundamentals

8.4

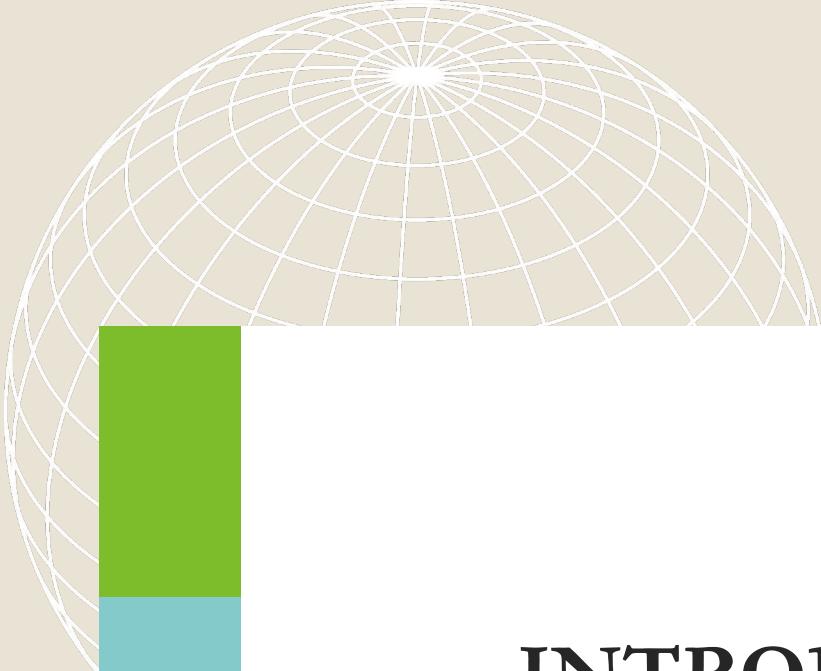
An approach to solving coding problems

8.5

Technical Topics to Study



8.6	A Word about Testing
8.7	Coding Problems and Solutions
9	SOFT SKILLS QUESTIONS
10	NEGOTIATING THE OFFER
11	HOW TO INCREASE DIVERSITY IN TECH



INTRODUCTION

This book is intended as a *clear step by step guide* to help you develop the skills and prepare the artifacts needed for landing a job in tech. You will be provided with a roadmap to follow, because there is a lot more to the process than showing up for an interview (such as getting called for one from a myriad of candidates). The book will outline step by step action items and provide you with the information needed to help realize your dream of working in tech.

We will start with an overview of the interview process, assess your current skills and strengths. You might not be sure if you should list that babysitting job (perhaps the only one you have on your current resume), or how you can gain the prerequisite experience if you haven't started working yet, how to go about building your online portfolio, or what steps you should take to make your resume stand out from other applicants. We will discuss the importance of your online presence and networking which can open doors for you during your job search.

Next, we will focus on your interview readiness. You will learn what topics to study, how to prepare and handle both behavioral and technical questions, as well as develop some techniques that will allow you to keep your cool (such as, did you remember to bring a protein bar or a snack to your interview).

We will go over common interview topics, some sample questions, and answers that are likely to be part of your interview. Of course, no one can guarantee what exactly you will be asked, but there are patterns that emerge, and we will discuss those. In addition, we will go over some general problem-solving techniques you can apply to virtually all problems, including if you are faced with a problem that you have never seen before. This said we will devise a practice plan for you to cover as much ground as possible before the interview.

Many people accept a job offer immediately, glad to have been extended one, but we will discuss the importance of negotiating, and how to masterfully do so once you have landed the coveted job offer.

Landing a job in tech can be challenging, but it is also very rewarding.

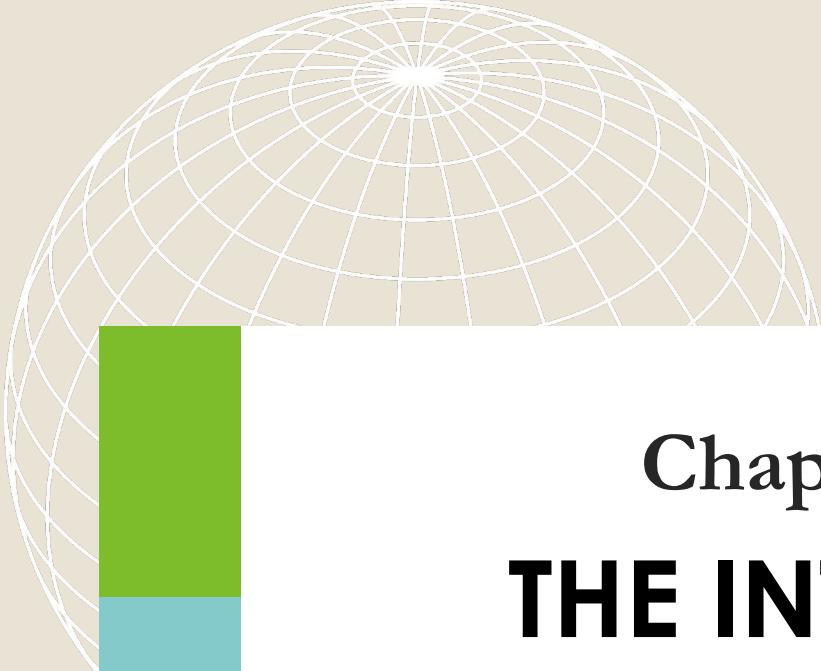
Let's get started!

Who this book is intended for?

This book is intended for first time tech job seekers or for those wishing to transition to a tech job from another industry. The book will take the guesswork out of making a plan to follow during the job hunting process and will cover the basics of the topics you need to know.

This book is not a computer science course, but you'll be given plenty of references to pursue if you are new to computer programming. The coding examples included assume a basic understanding of computer coding, but this might not be the case for some readers. Therefore, I have included a brief introduction to basic programming concepts, but you might find it helpful to complete a programming fundamentals course in Python or Java.





Chapter One

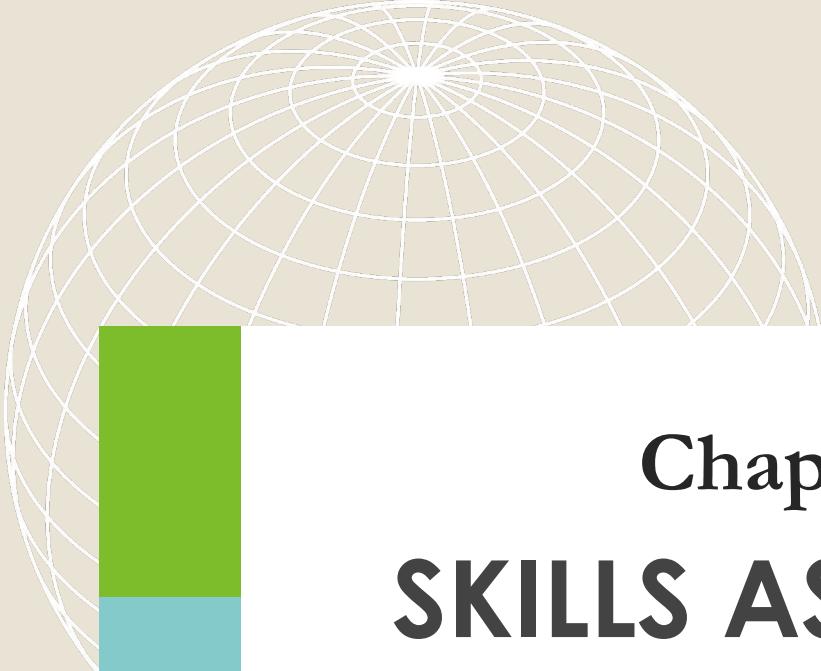
THE INTERVIEW PROCESS

You have decided that it's time to pursue your dream job in tech. Where to begin?

The process begins with assessing your current skills, identifying any gaps you might need to fill, both on the coding and behavioral sides and producing an outstanding resume and cover letter template. A template, because you will need to put in the time to customize your resume and cover letter for each job you apply for. It will take some work, but it will also increase your chances of getting a call for an interview. In addition, you will update your social media presence, especially on LinkedIn, and your coding portfolio on Github. Once your resume and cover letter templates are ready, you'll focus on your job search and submitting resumes. We will discuss how to optimize the job search process so that you get called in for an interview. Next, we will focus on preparing for the interview(s). We will cover topics, resources, tips, and tricks, and coding problems and solutions. We will discuss the day of the interview and how to prepare for it.

Many people don't realize that they can negotiate an offer and instead accept the first offer they are presented with. We will discuss some negotiation strategies to help you get the most optimal offer.





Chapter Two

SKILLS ASSESSMENT

Let's take an inventory of your existing skills and of the skills you still need to acquire.

We begin with a popular question: Do degrees matter?

This is a question that often comes up. Having a degree could improve your chances of getting called for an interview, but not having a degree is acceptable in most companies for candidates who have built up their resumes with relevant work experience, as well as project portfolio pieces. We will discuss in later chapters how to create your coding portfolio.

How about degrees in non-technical fields? Having a Bachelor's degree in a nontechnical field, paired with a programming certificate from a reputable institution could make you an attractive candidate for a job.

Note that there is a variance between companies in their hiring practices for non college graduates. Compared to other fields, tech sector is more open to hiring employees without degrees if they can demonstrate technical proficiencies.

What if you don't have any technical experience?

Should you list your pet sitting or retail job on the resume? Only if the skills demonstrated are relevant to your new career. What kind of skills could they be? Technical skills, such as software you learned on the job, soft skills, such as working on a team, meeting deadlines, or problem solving. Especially if you have a lot of jobs to list, there is no point to list those that are not relevant to your new career direction.

What if you have no work experience or any relevant background?

In this case, you need to get creative and perhaps volunteer for relevant positions, such as tutoring, building or maintaining websites for nonprofits, contributing to open source projects, or completing an internship. Of course, you will need to acquire the essential technical skills first. Workforce retraining programs, which often offer funding to cover tuition, books, and more, as well as some boot camps, generally a more expensive option with sometimes inflated promises of job placement, can bootstrap your programming knowledge in a relatively short time. They can also provide you with a degree or certificate to add to your resume. In addition to learning new skills, by enrolling in a class, you'll build connections and start growing your professional network.





Hands-on Activity: Skills Assessment

Use this Skills assessment template to evaluate your existing skills and identify any gaps. You likely can list past experience for the nontechnical skills, such as problem solving or teamwork.

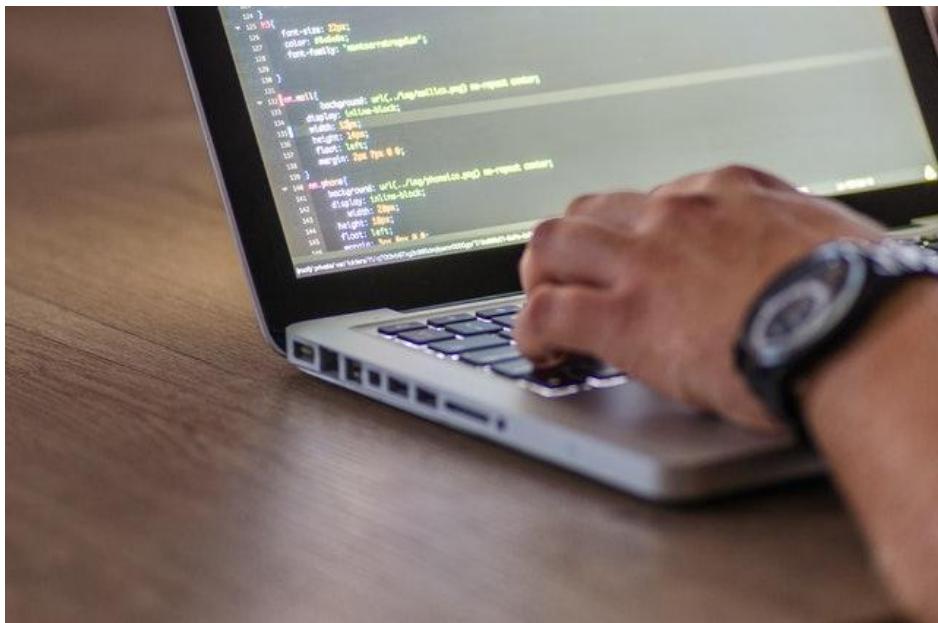
*Tip: Sign up for a free cloud account such as [Google drive](#) and save the answers to the Hands-on activities to it for future reference and possible updates as you grow your skills set.

Type of Skill	Yes or Needs work	Notes
Coding skills At least one, preferable two+ coding languages such as Java, Python, JavaScript or other		
Software development Version control such as github Agile development methodology		
Object oriented Programming Abstraction Encapsulation Inheritance Polymorphism		
Systems Design		
Software testing and debugging Basic testing terminology knowledge Unit Testing		
Problem solving and logical thinking		
Written and verbal communication		
Teamwork		

Technical training options

After completing the Skills Assessment, you might conclude that you need technical training before pursuing a career in tech. If this is the case, you have some options to consider based on several factors, such as your learning style, work, family and financial situation.

1. ***College Courses leading to a degree or certificate.*** Many colleges offer flexible and relevant technical programs. Depending on the college, costs vary. Community colleges in particular offer affordable options. While a college degree is not required for many tech jobs (provided you have the skills and experience), having one might help.
2. ***Bootcamps*** have become popular, in part due to their aggressive marketing and often extravagant promises. Some bootcamps do offer great programs, but the costs and time commitment required make them not an ideal option for some students.
3. ***Self study*** Many of us learn better in a structured environment. Research shows that the attrition rate for MOOC (Massive Open Online Courses) is very high. If you work or learn well independently, a MOOC might be a good, low cost option to consider.





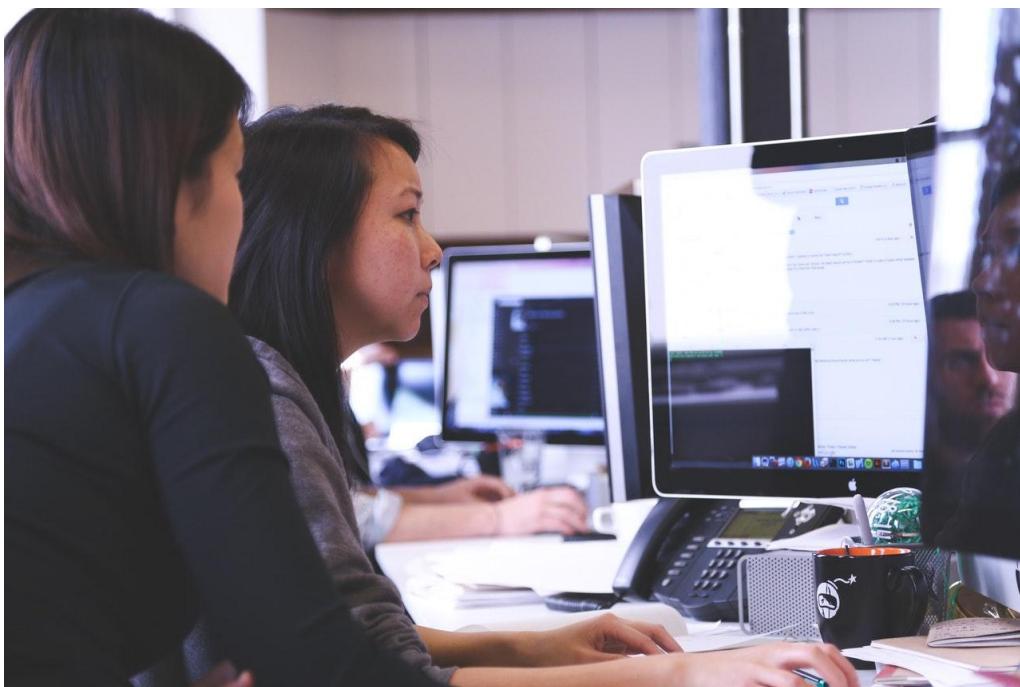
Chapter Three

THE TECH JOBS LANDSCAPE

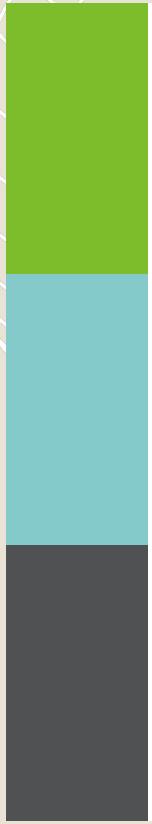
Types of Employment and Tech Job Roles

There are many roles in the tech industry that you can aim for. There are also different types of positions based on the hiring contact type, salary and benefits structure they offer. Let's take a look.

Type of Employment	Brief Description
<i>Direct Hire</i>	"Permanent" position, eligible for benefits such as health care, dental and 401K retirement
<i>Contract</i>	Hired for a specific job, time duration and pay rate. Not considered an employee of the company.
<i>Freelance</i>	Self employed or employed by a temp agency for specific work. Charges by the hour or per project.
<i>Entrepreneur</i>	Sets up and owns a business and could hire other people or use automation to earn profits.
<i>A combination of the above</i>	Some companies have policies that do not allow full time employees to moonlight.



Tech Role	Brief Description	Entry level average base pay *
Software engineer	Write, debug, maintain, and test software in one or more programming languages.	\$74,532
Web Developer/Front End Developer	Implement the visual components of web applications.	\$76,929
Mobile Developer	Design, debug, test, and maintain mobile applications.	\$96,016
Cloud Engineer	Design, plan, maintain and support cloud software.	\$97,906
Games Developer	Use computer languages and creative skills to create computer games.	\$50,923
Database Administrator	Plan, develop and troubleshoot databases with the goal of data consistency.	\$78,779
Data Scientist	Use analytical, statistical, and programming skills to collect, analyze, and interpret data.	\$122,948
Data Analyst	Interpret data for patterns and trends.	\$62,453
Test Engineer	Test software prior to launch to identify to ensure quality.	\$71,478
Technical Project Manager	Ensure that projects are completed to specification, and within the established budget and timeline.	\$88,397
Network Automation Engineer	Manage a network with software, reduce manual work, improve tools and processes.	\$116,017
Data Engineer	Data generation by transforming data into a format that can be easily analyzed.	\$102,864
*Sources: glassdoor and indeed		



Chapter Four

Resume

How long do you think recruiters spend reviewing a resume?

A few minutes at least? According to a study by The Ladders, recruiters spend six seconds per resume. It might not seem fair, but since there are generally more applicants than job openings, the recruiter is looking for reasons to narrow down the interview pool. A poorly written resume gives them a good excuse to disqualify you before you even had a chance to showcase your skills.

How to write a great resume for a tech job

Structure your resume with clearly defined sections, such as ***Header, Education, Work Experience, Skills, Projects.***

List ***relevant*** experience. If you are a recent graduate or switching from a non-tech career, be sure to also list any relevant school projects, internships, tutoring, part-time work or volunteering. On the other hand, if you have a long employment history, focus on the last 10 years, unless you have relevant work experience before that.

Your resume is a narrative, a story you tell about yourself that should present you in the best possible light. Each line of your resume should send a message that you are qualified for the job. Ask yourself “***What story does my resume tell about me?***” Do you like the story it tells?

One way to structure your story is by focusing on *accomplishments, challenges, and lessons learned during your career*. How do these experiences make you qualified for the job?

Mind the Gap (in Employment)

Recruiters are not fond of gaps in resumes. If you have such gaps after, for example, staying home to raise children, you should account for them on your resume. Think of any learning, training, side projects, or volunteering that you did during this time frame and list it on your resume.

Follow this UI/UX advice Use one of the resume friendly fonts (source: monster.com): *Calibri, Times New Roman, or Arial*. It is acceptable to use multiple colors, but you don't want your resume to be memorable for its colors alone. Add plenty of *white space* for readability and use no more than *three different font sizes*.

Save your resume as PDF, so that the formatting is preserved. Sometimes .docx documents distort the format. Of course, you should check what is the preferred submission format for the company.

Since you are looking for professional employment, you should **get a professional-looking email address** to list on the top of your resume and to exchange emails with the recruiter.

List your **Github and LinkedIn** accounts in the header of your resume. This is now standard practice for tech jobs.

Be sure to run a spelling and grammar check on your resume with tools such as the free [grammarly](#).

Non native speakers In addition to spelling and grammar checks, personal information such as age, nationality or country of origin is not listed on resumes in the US (as it is in some other countries).

A word about keywords. An entire section on keywords follows, but it deserves a mention here that you should focus on keywords *exactly* matching the job description. Tech skills keywords, such as “React JS” are likely weighed heavier than subjective keywords such as “problem solver.”

Use **measurable verbs** such as “designed”, or “implemented. Bloom’s taxonomy of measurable verbs can be found [here](#).

Quantify your work with metrics For example, instead of writing “Worked on an auto dealership website” you could write “ Increased auto dealership website daily views from 20K to 150K.”

If you don't have a lot of experience, list and discuss technical projects (can be under its own Projects section or as part of Experience). You could use school or independent projects. Briefly describe the scope of the project, its goal and the technology you used to build it. Note: Open source contribution projects are well regarded by many employers.

Resume length Unless you have an extensive and relevant work history, a resume length should be kept under 2 pages. This is not a hard and fast rule, but a guideline.

No need to list basic computer skills such as Microsoft Word or Excel unless they are mentioned in the job description.

How many programming languages should you list? This is a tricky question, because on one hand you want to show knowledge in multiple languages, but on the other hand, if you list a language, you might be asked to code in it or discuss its intricacies. One way to approach this dilemma is to add a *competency level* for each language. For example: Python (expert), Java (proficient), C# (familiar with). Be wary of using buzzwords such as “big data” or “artificial intelligence,” unless you have real proficiencies in those areas.

What about degrees? While having a Computer Science degree from Stanford will not hurt your employment prospects, degrees, and especially degrees from prestigious institutions do not carry as much weight in the tech world as they do, for example, in the world of business and MBAs. Big tech companies do recruit from top schools, but in general tech companies focus on the skills more so than the degrees.

What about hobbies? If your hobbies highlight a quality that you think is *relevant* for the job, you should list them. For example, if you build computers for fun or mine bitcoin in your spare time, this could be viewed as technical proficiency.

References Section This section is optional. If you add it at the end of your resume, you could write “References available upon request.”

Whether or not you have a references section in your resume, start gathering references, because you will be asked to provide them when you are offered the position. College professors, employers or colleagues are some references to reach out to. Be sure to let your reference know to expect a call or email from the company.

In most cases adding your **home address** to your resume is not necessary. Keep in mind that this is sensitive information, so only share it if you have a good reason to do so, for example if the geographical location has significance for the job position.

Resume or CV? Resumes are used when applying for most business and industry positions. A resume is designed to showcase your relevant education, work experience and skills for a job opportunity. A curriculum vitae (CV) is used for job opportunities in academia and research and often includes publications or research experience.

Below is an example resume for a Junior Software Enginner.

Andrea Jones

Software Engineer

[Linkedin profile](#)

[Github profile](#)

(555) 555-5555

andrea.jones@gmail.com

SKILLS

- BAS in Application Development graduate with strong software engineering and programming skills in multiple languages and across multiple platforms.
- Experienced in developing, testing and debugging code, object-oriented programming, web and mobile development.
- Quick learner who delivers results both as a team member and individually.

EDUCATION

School Name, Location - Degree

MONTH 20XX - MONTH 20XX

Academic Honors: Dean's List (2 quarters)

Capstone Project:

- Project Scope: Design and develop a food delivery mobile app for restaurants who want to serve their customers without an intermediary. The app was designed with Material Design. The app was developed as part of a four member team working with the SCRUM methodology. Analyzed functional requirements, produced code and conducted unit and user testing.
- Tools: Android Development Studio, Kotlin programming language.

Courses Completed:

- Version Control
- Python Programming and Object Oriented Programming
- Java Programming, Data Structures and Algorithms
- Data Analysis with Python
- Database Development with SQL
- Web Development with HTML, CSS, JavaScript and ReactJS
- Android Mobile Development
- Cloud Computing
- Systems Analysis and Software Development Methodologies

EXPERIENCE

School Name, Location - *Computer Lab Support Specialist*

MONTH 20XX - PRESENT

- Provide desktop and application support, account maintenance and printing assistance to students and faculty.
- Troubleshoot technology problems and provide timely solutions.

Company Name, Location - *Web Development Volunteer*

MONTH 20XX - MONTH 20XX

- Volunteered to design and develop a mobile friendly website for non-profit Company Name.
- The website resulted in 25% increased web traffic.

Company Name, Location - *Barista*

MONTH 20XX - MONTH 20XX

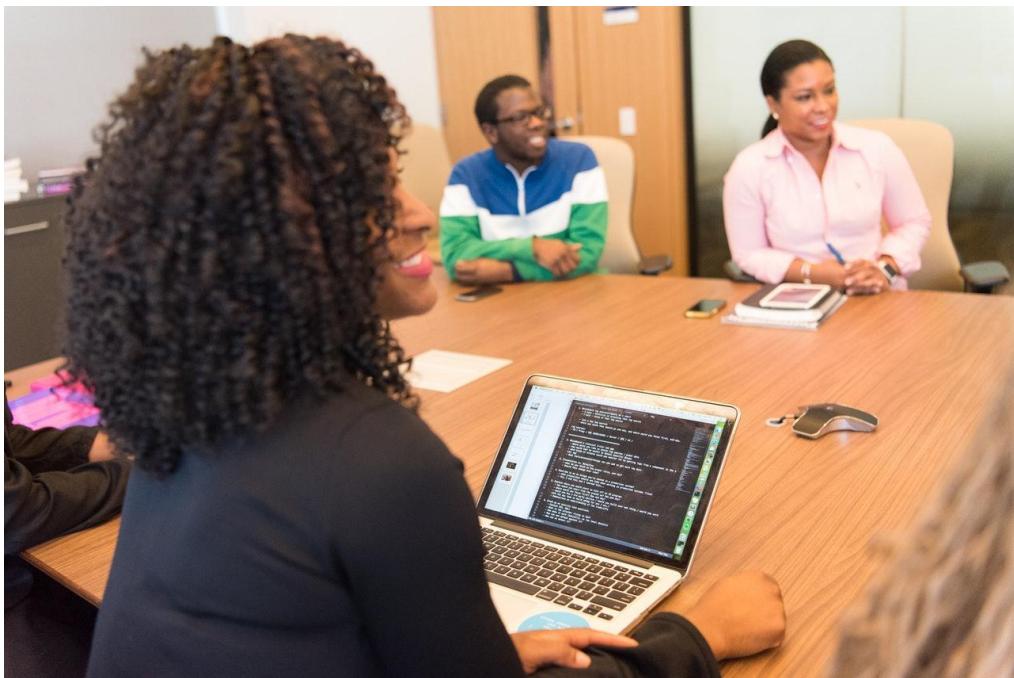
- Delivered excellent customer service resulting in repeat customers.
- Developed great time management skills while balancing work and college studies.

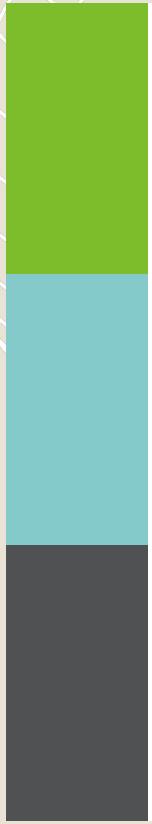
TECHNOLOGY SUMMARY

Python, Java, Kotlin, XML, Android Studio, HTML, CSS, JavaScript, React, Node.js, PHP, MySQL, Wordpress, Linux

▶ ▶ ▶ Hands-on Activity

1. Search online for a junior job posting you are targeting, or use the resume example from this chapter. Create a resume following the tips and techniques discussed in this chapter.
2. Once you create a resume, save it as a template to your cloud drive to customize for specific positions you will be applying for in the future.





Chapter Five

Cover Letter

How to write a great cover letter

A **Cover Letter** supplements and elaborates, in a narrative form, the points you have listed on your resume. When applying for a job you should always include a cover letter along with your resume.

Describe your experience and how it makes you a great fit for the role within the company. Express your excitement about the role and the company. Refer to the company by name rather than the generic “your company.”

Without copying and pasting directly from your resume, describe your relevant experience and skills in a narrative form. Give specific examples and use measurable verbs where possible.

Close your cover letter by reiterating your interest in the role and thanking the employer.

How to address a cover letter The cover letter is usually read by the hiring manager. Chances are that you do not know who the hiring manager is. You have a couple options here. One is to ask the recruiter who the hiring manager is. Alternatively, you could conduct a search for the name of the hiring manager on the company website or on LinkedIn. Another option is to use one of the more generic addresses listed below.

Addressing the hiring manager by name shows initiative, but if you didn’t get the correct name, it would not make the desired impression. There are known cases where the hiring manager changed between a candidate’s phone and in person interview. This might not be a typical scenario, but it does happen. It is up to you which option you chose. If you decide to use a generic address, here are some suggestions:

Dear Hiring Manager

Dear <Insert Company and Department Name> Team

How to close a cover letter The following are suggestions for closing cover letters or other professional correspondence. They can also be used when writing a *Thank you note* after an interview (you always should do so).

Sincerely

Sincerely yours

Regards

Best regards

Respectfully

Thank you

Thank you for your consideration

In addition to the closing address, add your contact information, so that it is easily accessible by the recruiter and hiring manager. Alternatively, contact information can also be added on top of the cover letter.

Sincerely,

Name

Email Address

Phone Number

LinkedIn Profile URL

Below is an example of a Web Developer cover letter.

Evan Harris

[Linkedin profile](#)

[Github profile](#)

(555) 555-5555

evan.harris@gmail.com

August 4th 20XX

Dear Hiring Manager,

I am pleased to apply for the Web Developer position at Company XYZ. As a recipient of the Web Development Certificate from ABC College, I have experience in designing, developing and testing websites using JavaScript, React and Node.js. In addition, my background includes working in fast paced development environments while meeting schedules and producing deliverables.

I am a quick learner who takes the initiative and delivers results. In my previous experience as a Web Development Intern at ABC College, I worked on several time sensitive website updates. I was involved in the redesign, development and deployment process and also worked on an enhancement request after the website went live.

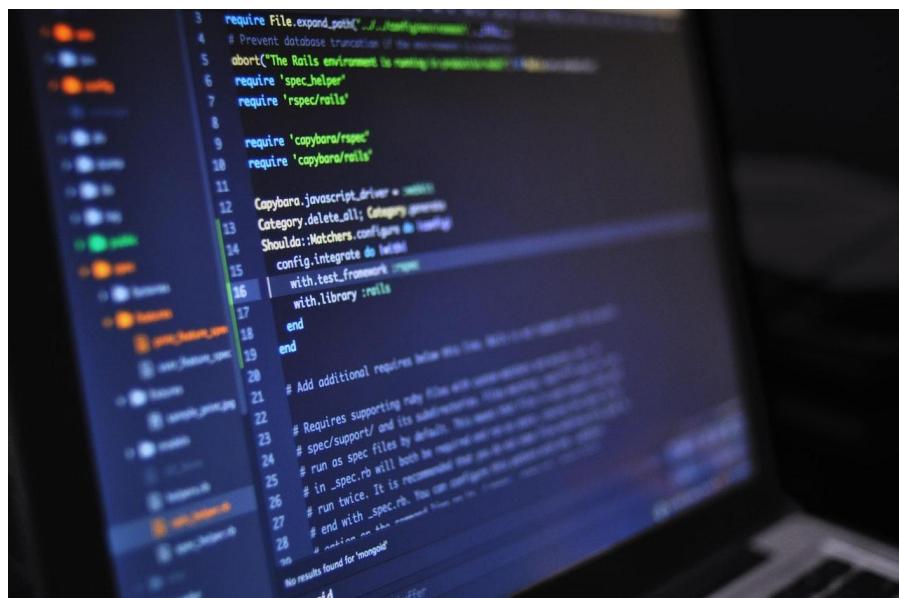
Thank you for taking the time to review my credentials and accomplishments. I am very passionate about delivering technology solutions that improve people's lives. Therefore, I believe that I am a great fit for this role. I hope to have the opportunity to discuss my portfolio examples with you. I look forward to answering your questions and learning more about this position and your team.

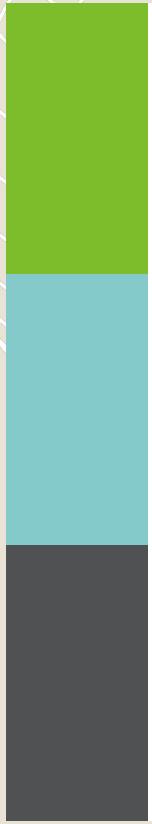
Sincerely,

Evan Harris

▶️ Hands-on Activity

1. Search online for a junior job posting you are targeting, or use the cover letter example from this chapter.
2. Create a cover letter following the tips and techniques discussed in this chapter.
3. Once you create a cover letter, save it as a template to your cloud drive to customize for specific positions you will be applying for in the future.





Chapter Six

Online Presence

LinkedIn

LinkedIn is a social network for professional networking and career development, owned by Microsoft.

How to create a great LinkedIn profile:

If you don't already have a LinkedIn profile, create one at [linkedin.com](https://www.linkedin.com)

Complete as many sections of your profile as possible. It is a good practice to use the [LinkedIn Profile Strength Meter](#) tool. The tool gauges the strength of your profile and provides suggestions on what to add in order to increase the discoverability of your profile (source: LinkedIn).

Add a Headline to your profile The Headline is your “sales pitch” condensed to 120 characters or less. Keywords in the headline, matching jobs you are interested in, will increase your chances of appearing in LinkedIn searches.

Add a ***professional looking profile picture*** and a ***custom background image***. Before selecting your images, ask yourself what message do you want to convey about yourself to potential employers? Outgoing? Competent? Friendly?

Once your profile is (mostly) complete, start making connections on LinkedIn. Invite other students or co-workers to connect with you. You could conduct a search on your educational and/or professional institutions and initiate contact with people in these networks. Ideally you should have 100 or more connections.

Request LinkedIn recommendations or endorsements. A recommendation is a brief letter from your college professor, manager, or colleague. Your LinkedIn connections can endorse the skills you have listed on your profile. Be sure to keep your skills and endorsements current as you continue to grow your skills.

One strategy for gathering endorsements is to endorse the skills of others in your network. Chances are that they will endorse you back. If your profile does not have an endorsement section, you can add it by clicking on: *Me icon (top of your homepage) -> Select View profile -> Add Profile section -> Skills*

Optional Customize your LinkedIn URL. You can change the URL to include your name. Go to *Me icon (top of your LinkedIn homepage) -> View profile -> Edit public profile & URL -> Public profile settings -> Edit your custom URL -> Edit. After entering your custom URL click Save.*

Once your profile is (mostly) complete, you can start joining LinkedIn Groups (also discussed in the networking section). LinkedIn Groups present a great way for networking. Search for groups in your areas of interest, e.g. ReactJS Web Developers, Python Data Science community or employment related groups in your area. It is best if you can be an active participant, engaging in discussions or sharing information and resources. Use the LinkedIn messaging feature to email recruiters or other professional contacts. Note that LinkedIn messaging outside of your immediate plus one degree of separation network is a premium service.

Searching for jobs on LinkedIn In addition to hosting your LinkedIn profile, LinkedIn has a job searching site you can visit [here](#). Note that in some cases applying for jobs is a premium service. You can use the “*Easy apply*” option for free, but it only provides a snapshot of your profile to the recruiter.

GitHub

GitHub is a code hosting and collaboration platform owned by Microsoft.

GitHub can be used by the individual programmer or in a team setting. The platform provides tools for uploading and managing code, as well as for collaborating with others. Some of the supported features include copying code from another account (forking), notifications to the original owner about changes made to their code base (pull requests) and integrating code originated by other developers (merging). GitHub can be used from the command line or with a visual interface.

The content of your GitHub account is visible to recruiters, hiring managers, and other programmers. Similarly to your resume, cover letter and LinkedIn profile, you want to make the best impression with your GitHub profile. Having a GitHub profile is a requirement for virtually all tech jobs today. What type of projects should you store on GitHub? A variety of well organized and documented projects that showcase your coding skills, as well as your ability to organize and document your work.

How to create a great GitHub portfolio

If you don't have a GitHub account sign up for one [here](#). Add your first and last name, a professional email address and a profile picture (of you) to your profile.

Initialize every project with a README file. The README should contain a brief project description, project features, and if applicable, installation instructions, a link to a live website, and screenshots of the working project.

Note: If you are not familiar with GitHub, find an online tutorial and start by learning the basics. There are plenty of free GitHub tutorials out there. The official documentation is also a good starting point.

Ideally, you are contributing consistently to your repositories at least a few times a week. The timeline of your submissions on GitHub is displayed on the front page. Github's calendar displays submissions in green. You want your calendar to be colored in green.

Use a directory structure to organize your files, add program headers and comments to your files. You want to give the impression of *readability and organization*. While it is possible to store binary files on GitHub, they often add unnecessary volume to your repository. You should include them only if you have a good reason to do so.

You can pin your favorite projects on GitHub, so they are the first thing a visitor will see. Here is a [link](#) describing how to do it.

What types of projects should you include

Classic programming exercises, in your language of choice, such as FizzBuzz , Eight Queens Puzzle, Pig Latin, Sieve of Eratosthenes, Conway's Game of Life.

Develop a website in your language or framework of choice and add it to your repository.

Develop a game such as Tic Tac Toe, Pong or Sudoku and many others.

Develop a mobile app This can be a game, such as Dice Roller, a Task List Manager, Reminder app, Location based apps, and many others.

Develop a script, utility or plugin. For example, you could write your own jQuery plugin for a sliding image gallery or a test automation script.

If you have a specific company in mind, create a project that calls the company's **API** (Application Programming Interface). Many companies these days provide API. For example, if you are interested in working for Twitter, you could build a Twitter sentiment analysis program, or if you are interested in displaying images from Flickr in real time, you could use their Flickr API.

Include a **Data Science or Data Analysis project**, such as data visualizations or recommender systems.

Contributing to **Open Source Projects** is a great way to get involved in the developer community and to boost your GitHub portfolio.

Take a look at [this](#) repository, featuring beginner friendly open source projects in a variety of languages.

Additional ways to interact with the development community Submit a pull request, add a new feature to a project, fork a project, submit bug reports or add to the documentation of existing projects.

For more ideas take a look at [this](#) list of coding projects. For inspiration, take a look at [these](#) most active github contributors and their profiles.



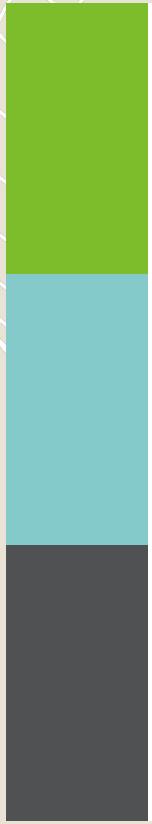
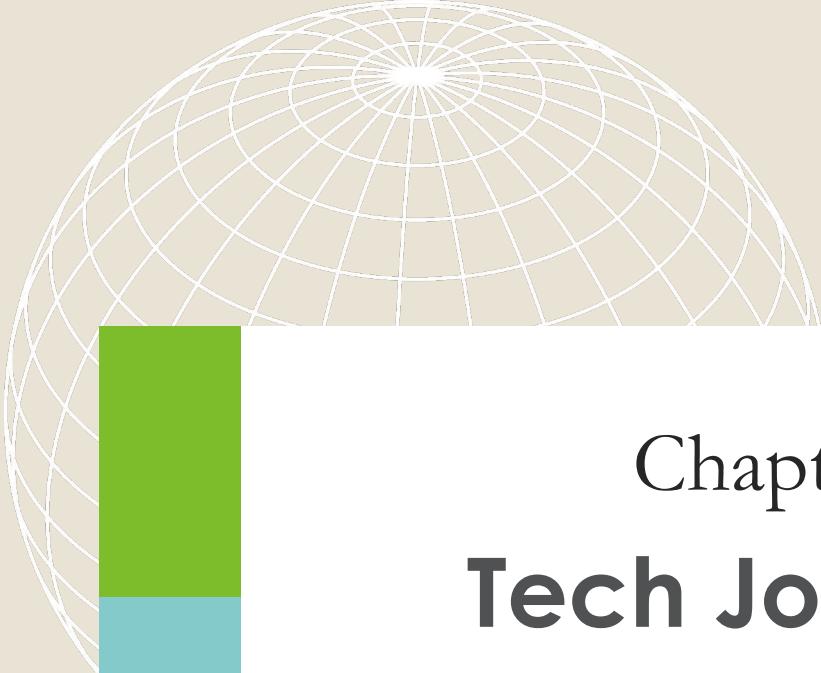
Hands-on Activity

1. Sign up for LinkedIn and create a public profile, following the LinkedIn wizard and adding as many steps as possible to a 100% complete profile.
 2. Search for classmates or colleagues and request to be linked to them.
 3. Message some of your connections and ask them to endorse the skills you added to your LinkedIn profile (e.g. Agile development, Java, Python, etc)
 4. Search and sign up for groups relevant to tech industry job seekers.



Hands-on Activity

1. Create your GitHub Portfolio planner using the following template and selecting 4-7 projects from the ideas discussed in this chapter.



Chapter Seven

Tech Jobs Search

Resume scanning tools

An ***Applicant Tracking System (ATS)*** is an automated system that scans a database of resumes for matching keywords. The ATS does not know who is actually a qualified candidate, nor does it have the nuanced thinking of a human. Instead, it determines a good match based on *keywords*.

Most of the resumes never make it in the hands of HR or a hiring manager, because companies employ ATSS to filter out the resumes that do not match the job description close enough. *This is why it is so important that you tailor your resume for each position you are applying for.*

How to win against the AST

While there are different applicant tracking systems, they generally operate on the same principle: scanning your resume and matching keywords against those imputed by the recruiter. When a recruiter searches for a specific job posting, the resumes with the most accurate keyword matches will be pulled up. Note that tech skill keywords, such as listing a technology are usually weighted heavier than soft skill keywords, such as teamwork or problem solver.

Your resume should match the exact wording of the job description (e.g. Wordpress is not a match for CMS (Content Management System), CS not a match for Computer Science. This also includes spelling, plural vs. singular words, numbers (five years vs. 5 years), etc. Be sure to always write out years without abbreviations, e.g. 2020 rather than 20.

At this stage, listing the exact technical skills matching the job description should be your focus. Of course, you should list soft skills as well. Your writing skills will be assessed by your resume and cover letter. For some positions you might be asked for a writing sample.

With the use of a ***Resume Keyword Tool*** you can compare your resume to the job description and find out how closely you are matching the description. Resume Keyword Tools can also suggest keywords to add. A couple tools to consider are [JobScan](#) (up to 5 free job scans per month at the time of this writing), or a word cloud tool such as <https://www.wordclouds.com/> (copy and paste the job description for a visualization of the most frequently used words).

Where to look for and apply for jobs online

- ***Job search engines*** such as [indeed.com](https://www.indeed.com) aggregate job postings from companies or job boards.
- ***Job Boards*** such as [careerbuilder.com](https://www.careerbuilder.com) or [monster.com](https://www.monster.com)
- ***Target your job search*** by making a list of companies you are interested in. Then you can tailor your skills and your resume accordingly. Also, look on company websites for job postings.
- ***Alumni or professional association websites*** are another source for job postings.

Below is a list of some of the top job search engines for 2020 (source: Career Sidekick).

Tip: create an alert on one or more of these websites to receive email with new job postings daily.

[indeed.com](https://www.indeed.com)

[careerbuilder.com](https://www.careerbuilder.com)

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

[glassdoor.com](https://www.glassdoor.com)

[simplyhired.com](https://www.simplyhired.com)

linkup.com

ziprecruiter.com

roberthalf.com

Networking

Networking is one of the most important activities you can do while looking for a job. Consider attending job fairs and other networking events, conferences (there are free ones you can attend) and presentations. Meetup.com is a great networking resource. Join the platform for free, specify your areas of interest and start joining groups. You will receive email alerts for upcoming events in the area. Many of these events are free. At the time of this writing, virtually all events are held virtually due to the Covid pandemic.

For example, the Android Developers meetup takes place once a month and is hosted by tech companies in the area. It is not uncommon for free food to be served during the in person meetings.

Another virtual place to network is LinkedIn Groups. Join groups in your areas of interest and when possible, participate in the discussions and make yourself visible and known to the other group members.

When in person interaction is possible, engage in conversations and do your best to exchange contact details. ***Should you have a business card?*** Having a business card can be helpful, but it is not essential. Many people these days connect directly on their phones via professional email, LinkedIn or other social networks. It is important to follow up with your new contacts, especially if you are a job seeker. Many people these days are busy and might simply forget to get back to you without a reminder.

Referrals

Research shows that many people land a job through networking with their contacts and social network. A referral from a current employee will bring your resume to the top of the queue. Employees often get bonuses for referrals leading to a hire, so they have an incentive to refer you. Don't be too shy to ask.

Elevator pitch

Imagine attending an event where a stranger walks up to you and starts a conversation. The stranger introduces themselves as a recruiter at a high tech company. Now it's your turn. ***What will you say?*** Ideally, you will have your elevator pitch ready, so that you can effortlessly and concisely introduce yourself in the best possible light. For most of us, it is helpful to have our elevator pitch ready ahead of time and to have practiced it.

How to create a great elevator pitch:

1. Your elevator pitch should include:
 - a. Introduction of yourself
 - b. Summary of what you do
 - c. Career goals
 - d. Call to action (give a card, invite a LinkedIn connect)
2. Write your elevator pitch on paper and memorize it.
3. Practice saying your elevator pitch in front of a mirror, by recording yourself, or rehearse it in front of friends or classmates.
4. Time yourself saying your elevator pitch. It should take about 60 seconds to say, or less.

Example elevator pitch

"Hi, my name is Jake. I recently graduated from Seattle Central College with a programming certificate. My area of focus is Android Mobile development. My most recent project was a fitness studio locator app. I am currently looking for Junior Software Engineer opportunities. Do you mind if we exchange cards or connect on LinkedIn?"





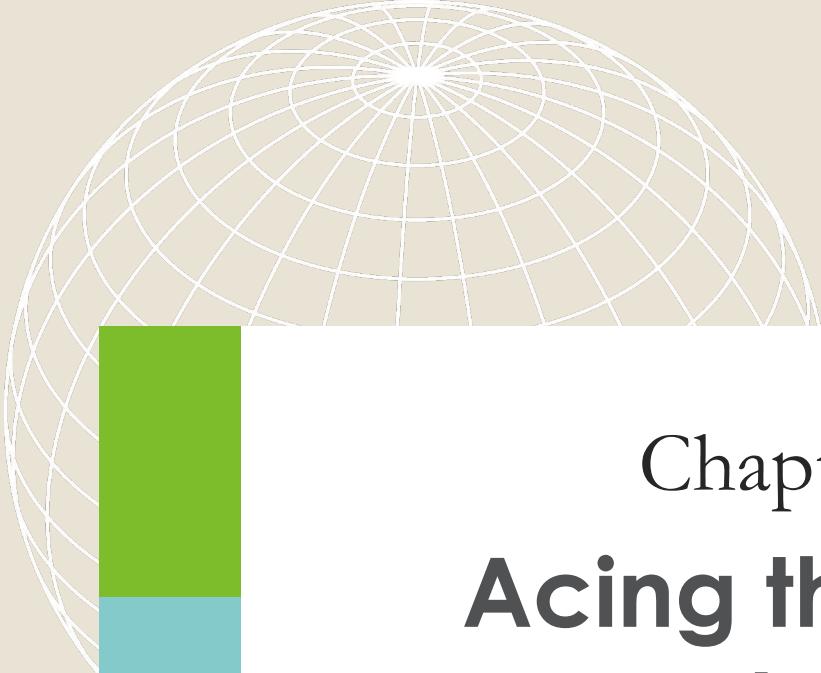
Hands-on Activity

1. Write your elevator pitch and practice saying it to friends or in front of a mirror.



Hands-on Activity

1. Run your resume vs. a target job posting on the jobscan tool. How did you do? Are there changes you need to make to your resume for a closer match?



Chapter Eight

Acing the Coding Interview

8.1 An Overview of the Coding Interview

Traditionally the coding interview has taken place in a company building on a white board. While there is some legitimate criticism if a white boarding interview truly reflects a candidate's coding ability, this has been the status quo way of conducting interviewing. Therefore, candidates have prepared for this style of interview where they need to solve coding problems on white boards without the usual developer aides such as documentation or Stack Overflow answers.

As a result of the Covid pandemic, most coding interviews currently take place virtually. While it is not known how things will change in the future, it is likely that virtual interviews are here to stay for many companies.

The type of virtual interview software varies by company. You might be asked to use an online coding editor and produce running code. Examples include [repl.it](#) or Hackerrank's [CodePair](#). Companies such as Zoom offer virtual white boards as well. Check with your recruiter which interview platform will be used, so you can become familiar with it ahead of time.

Tech Screens

Some companies also conduct remote technical screen interviews with a recruiter or an engineer prior to the official interview. While the process varies between companies, you will likely be asked to code in an online editor and produce working code.

White boarding and virtual interview preparation

To prepare for an in person interview, ideally you will practice white boarding mock interviews with a study group, so that you are not completely new to the process when interviewing at a company. You can purchase a mini erasable white board or try working on a blank sheet of paper. If you are in school, try joining (or starting) a study group to practice solving coding problems and doing mock interviews. Given that interviews these days are virtual, you can practice solving problems in an online coding editor such as [repl.it/](#) Repl has support for multiple coding languages.

These days there are plenty of excellent online resources that provide online practice environments. All you need is to find the time to practice.

Some popular online coding interview resources to get you started

[programminginterview.com](#) Coding questions and general interview advice

[hackerrank.com](#) Coding problems and interview preparation

[leetcode.com](#) Coding problems and interview preparation

[geeksforgeeks.org](#) Coding questions and interview related blog posts

[techdevguide.withgoogle.com](#) Includes past interview questions at Google

Day before the interview

Go over your resume and prepare your talking points about your projects. We will discuss behavioral questions in another section.

Have transportation and parking plans ready.

Try to go to bed early and get plenty of rest.

Day of interview

Dress comfortably, yet presentably. Business casual is usually a safe bet for a tech interview. We will discuss dress code in more detail in another section.

To a quote a software engineer friend “After your interview you don’t want to be remembered for what you were wearing.”

Water and coffee are usually provided, but you might bring your own caffeine source. Some interviews include lunch. This is something else to check on before the day of the interview. Many of the bigger companies include lunches.

Bring a protein bar or two. The interview might run for several hours, and you don’t want to run out of steam.

Remember to turn off your phone ringer.

Send a *Thank you note* to the recruiter and/or hiring manager after the interview.

8.2 Programming Languages and Technologies

Chances are that you were contacted for an interview, because you listed a programming language or technology on your resume that is in use at the company. You can plan on solving the coding problem in this language. On the other hand, you might be given a choice which language to use. Let's take a look at some popular languages, technologies, and their usage in tech.

Interested to learn what the most popular languages of the moment are? The Tiobe index provides a ranking of the most popular languages and updates it on regular basis. You can find the Tiobe index [here](#).

Android is a popular, open source operating system developed by Google and used for the development of apps for mobile devices. To develop apps for Android, developers need to know Java or Kotlin programming languages.

C is a popular and flexible general-purpose programming language used for the development of operating systems, enterprise software, games, and more.

C# is a general-purpose programming language developed by Microsoft. C# is used for software development with the Windows .NET framework.

Database Technologies:

Apache Cassandra is a scalable, high-performance, distributed, NoSQL database designed to handle large data sets.

MongoDB is a database that handles the storage of large datasets. MongoDB is a NoSQL database and uses key-value pair documents as its basic storage units.

MySQL is an open source Relational Database Management System (RDBMS). It uses Structured Query Language (**SQL**) to create, read, update, delete and manage databases.

PostgreSQL is an open source object-relational database system that uses and extends SQL.

SQLite is an embedded database system for local or client storage used by Android OS as well as some browsers.

Go is an open source programming language developed by Google. Go is flexible, easy to use and applicable to machine learning, as well as system and networking programming.

Java is very popular general-purpose programming language developed by Sun Microsystems and acquired by Oracle. Java is used in the development of Android apps, desktop applications, embedded systems and more.

JavaScript, “the language of the web,” is a scripting language used for the writing of client and server side web applications, mobile app and more.

NodeJS is a server-side platform used for developing server side applications, REST APIs, data streaming apps and more.

PHP, a recursive acronym for PHP: Hypertext Preprocessor, is an open source general-purpose scripting language used for server side web development.

Python is a popular, general-purpose coding language used in data science and analysis, machine learning, system scripting, software development and more.

React.js was developed by Facebook and is an open-source JavaScript library used for building user interfaces and handling the view layer of the MVC pattern for web and mobile applications.

Wordpress is a Content Management System (CMS) used for building websites. About 30% of websites are built with Wordpress.



8.3 Programming Fundamentals

Before working on interview readiness coding problems you should be familiar with basic programming concepts and syntax.

The coding snippets included below are in Python 3.X, a fantastic first language for new codes.

Variables

A variable is a container for storing data. More precisely, a variable is a location in memory where data is stored. The value of a variable can change.

```
>>> #Declare variables
>>> num1 = 5
>>> num2 = 7
>>> name = 'Sofia'
>>> num1
5
>>> #Variables can be reassigned
>>> num1 = 11
>>> num1
11
>>> print(num1, num2, name)
11 7 Sofia
>>> |
```

Expressions

Code that computes new data values by combining values and operators is called an expression. An expression evaluates to a single value. Simple identifiers can also be expressions.

```
>>> num1 = 5
>>> num2 = 7
>>> result = num1 + num2
>>> result
12
>>> |
```

Data Types

Variables can store data of different types. Different operations can be performed with different data types. Data types are built into a language and include strings, numbers (integers or floats), and booleans.

```
>>> type('Hello')
<class 'str'>
>>> type(55)
<class 'int'>
>>> type(3.14159265359)
<class 'float'>
>>> type(True)
<class 'bool'>
>>> type(['a', 'b', '12'])
<class 'list'>
>>> |
```

Conditional Statements

Conditional execution takes place when different branches of code are executed depending if a condition has been met. The condition checked is (or evaluates to) a boolean (True or False). Example of a conditional statement is Python *if...elif...else*

```
num1 = 5
num2 = 7
if num1 > num2:
    print(num1, 'greater than', num2)
elif num1 == num2:
    print(num1, 'equal to', num2)
else:
    print(num2, 'greater than', num1)
```

```
7 greater than 5
>>> |
```

Loops

In a loop a set of statements is executed as long as a condition is true. There are definite loops, where we know ahead of time how many times the loop will run. An example is the for loop. There are also indefinite loops, where we don't necessarily know ahead of time how many times the loop will run. An example is the while loop.

```
>>> for item in shopping_list:  
    print(item)  
  
cat food  
bread  
apples  
>>> |
```

Functions

A function groups together statements of code with the goal of accomplishing a task. Functions can take arguments as a mechanism for passing information to the function. Functions can return values as a way of returning information from the function to the calling program.

```
>>> # Define the function  
>>> def greeting():  
    print("Hello, friend")  
  
>>> # Call the function  
>>> greeting()  
Hello, friend  
>>> |
```

Sequences: Strings, Lists and Files

A string is a sequence of characters that can be accessed by their index. Lists are sequences of arbitrary values. In Python they can be a mix of different data types such as strings, numbers or even other lists.

```
>>> first_name = 'Sofia'  
>>> last_name = 'James'  
>>> full_name = first_name + ' ' + last_name  
>>> full_name  
'Sofia James'  
>>> |
```

Dictionaries

Dictionaries contain key-value pairs where each key is associated with a value. Keys must be unique, but values need not be. Dictionaries are not sequences and do not have indexing order. In Python the **dict** type implements a dictionary.

```
>>> test_scores = {  
    'Joe' : [100, 95, 75],  
    'Sofia' : [90, 90, 'absent'],  
    'Violet' : [100, 100, 95] }  
>>> print ('Joe\'s test scores are', test_scores['Joe'])  
Joe's test scores are [100, 95, 75]  
>>> |
```

8.4 An approach for solving coding problems

1. At the start, take a few deep breaths to relax your nerves. Practice slow deep breathing during the interview if you start feeling nervous or overwhelmed.
2. Carefully listen to the question and ask the interviewer any clarifying questions you have. Sometimes the interviewer might intentionally give you an incomplete problem statement, so asking clarifying questions is a good idea.
3. State back the coding problem to the interviewer, confirming that you understood it correctly.
4. Try solving the simplest case you can think of. At this point you are probably using pseudo code.
5. Review your solution and check to make sure it solves for the general case and not just a special case.
6. Check the Input, Processing and Output of your program.
7. Check for edge cases and boundary conditions.
8. After solving the problem “brute force,” examine its complexity class.
9. Think of ways to make your solution more efficient.
10. While working on the problem, maintain a dialog with the interviewer. This will keep them engaged. Also, their responses will guide you if you are going in the right direction with your solution.
11. Now it’s time to implement your solution in a programming language. In most cases the interviewer does not care which language you chose to code in. They will let you know if they want you to use a specific language.
12. Test your code. You are not done until you run a simple test through your code and are satisfied that it produces the expected result. Sometimes the interviewer might add additional test cases to check.
13. Ask the interviewer if there is anything else they would like you to consider or address.
14. If you are targeting a specific company, research what technologies are used there and focus on those areas. Interview questions and answers for many of the bigger tech companies are available online.

8.5 Technical Topics to Study

Assuming that you are familiar with programming fundamentals, the following topics are recommended for interview readiness. Note that there is always a chance you might be asked a question you haven't heard before. That is OK. Approach the question following the process outlined on the previous page. Your approach to solving the problem is equally important to many interviewers than the solution itself. On the job will be faced with new, unfamiliar problems all the time. Therefore, your ability to problem solve is key.

Be consistent with your study habits and practice, and you will see results.

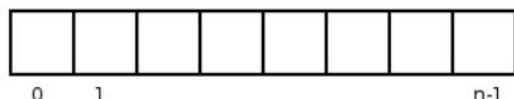
Data Structures and Algorithms

Strings

A string is a sequences of characters. The string data type is used in programs to represent text. The position of a character in a string sequence starts with 0, increases left to right. Individual characters are accessible with indexing. In languages such as Python or Java strings are immutable, meaning they cannot be changed after they were created.

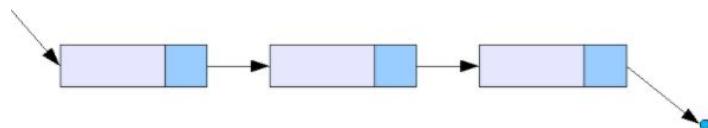
Arrays

An array is a data structure containing indexed elements. Each element can be accessed by its index. In most languages indices are 0 based.



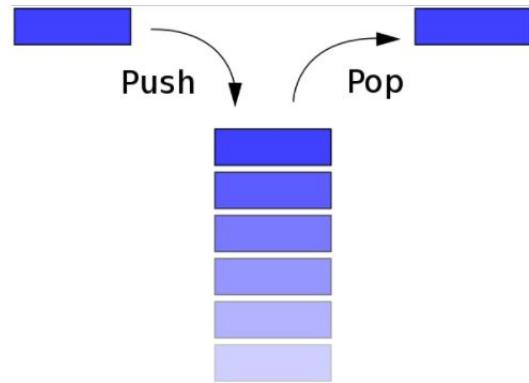
Linked Lists

A linear collection of elements (nodes) in which each element points to the next. Linked Lists can be singly or doubly linked. A analogy for a Linked List is a conga dance where each person holds on to the person in front of them. The last node points to null.



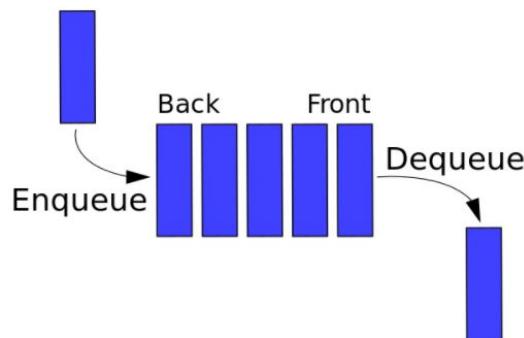
Stacks

A stack is a linear data structure in which the order of operations executed is Last In First Out (LIFO). A Post-it notepad is an example of a LIFO order.



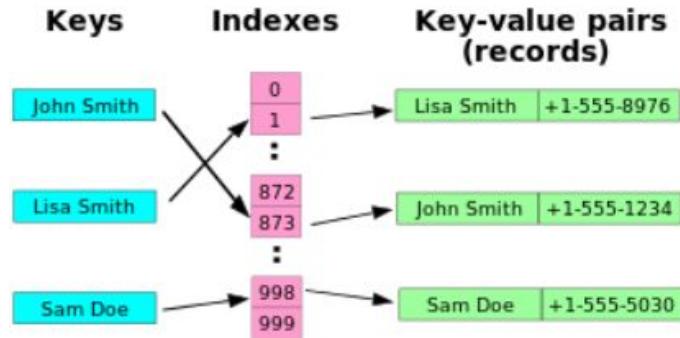
Queues

A queue is a linear data structure in which the order of operations executed is First In First Out (FIFO). A line of people waiting in the cafeteria to be served food is an example of a FIFO order.



Hash Maps

A HashMap stores data in (Key, Value) pairs where the key must be unique, but the values need not be unique.

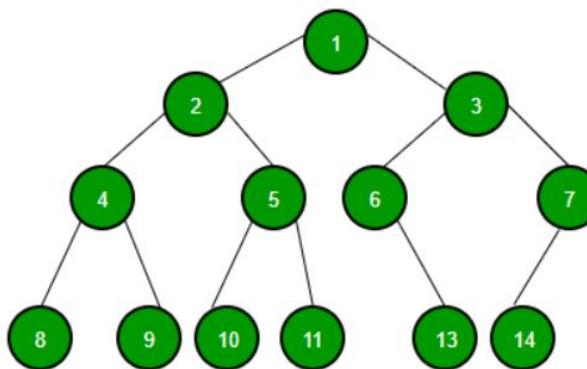


Sets

A set is an unordered collection of unique elements. The order of the elements in a set is not guaranteed. Set elements are mutable.

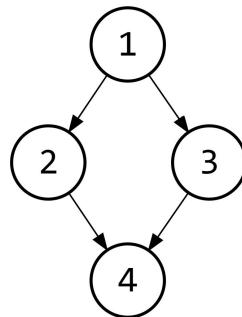
Binary Trees

A binary tree is a tree whose elements (nodes) can have at most 2 children: a right and left child. A binary tree nodes contain data as well as pointers to the left and right child nodes.



Graphs

A Graph consists of a finite set of nodes and edges, which connect pairs of nodes.

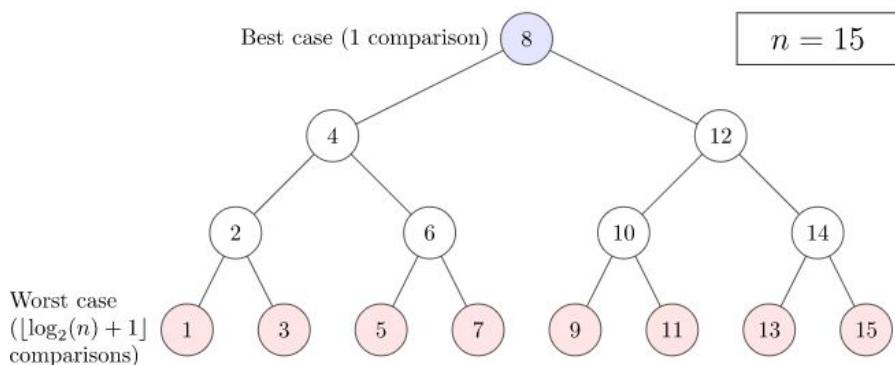


Search Algorithms

Binary Search

Binary Search uses a divide and conquer approach to search for a given value. Note that the array searched must already be sorted. If the search value is less than the middle, search only the first half of the array, otherwise search the second half.

Repeat until the value is found or the entire array has been searched. Binary Search is an efficient search algorithm with a complexity class $O(\log n)$



Tree Traversal Algorithms

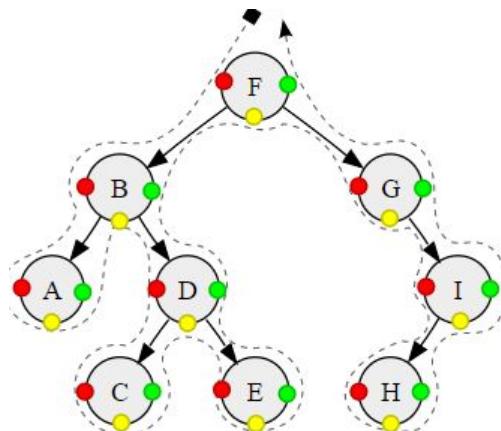
Trees can be traversed in the following different orders, where R is the root of the tree, and left and right are the left and right subtrees:

Inorder (Left, Root, Right)

Preorder (Root, Left, Right)

Postorder (Left, Right, Root)

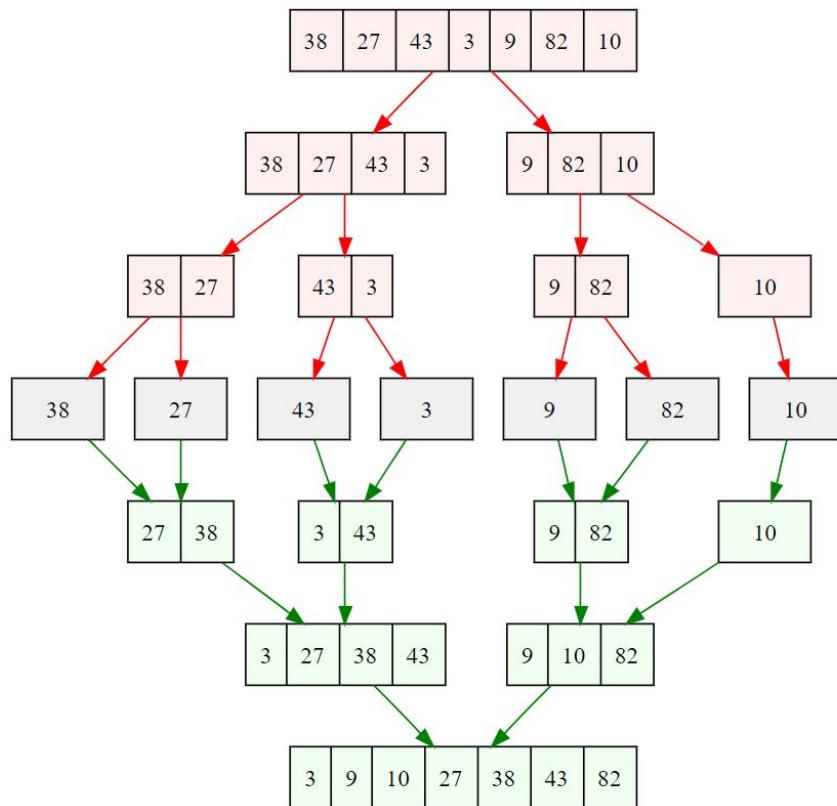
Complexity class: $O(N)$



Sorting Algorithms

Merge Sort

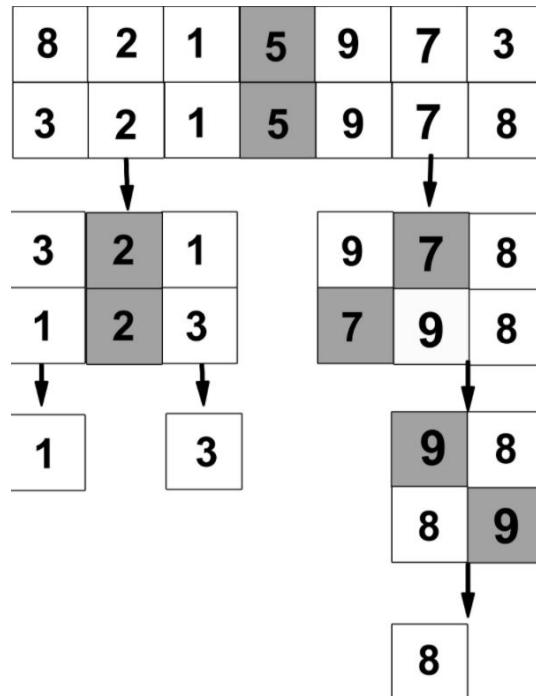
Merge Sort divides the array to be sorted in two halves, sorts each half, then merges the two sorted halved arrays. Merge Sort is an example of a Divide and Conquer algorithm with complexity class of $O(N \log N)$.



Quick Sort

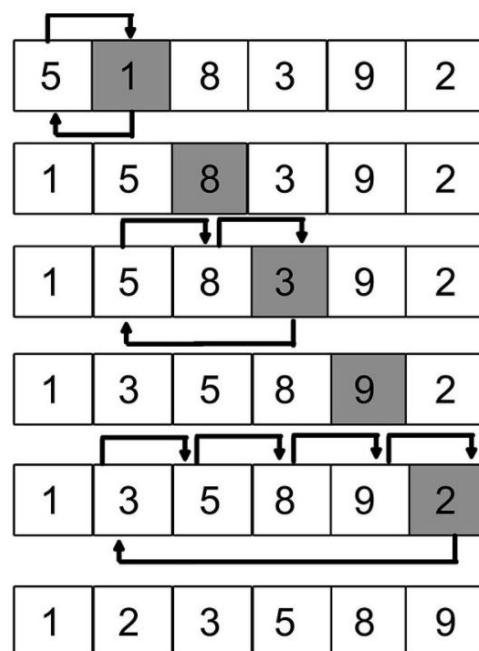
Quick Sort uses partitioning to sort. It uses a pivot value to start the process, then puts all smaller values before pivot value and all larger values after it.

Complexity Class $O(N^2)$



Insertion Sort

Insertion sort uses an algorithm where an array is divided into sorted and unsorted parts. Values are moved to be placed in the correct order so that the array becomes sorted. Complexity class $O(N^2)$



Selection Sort

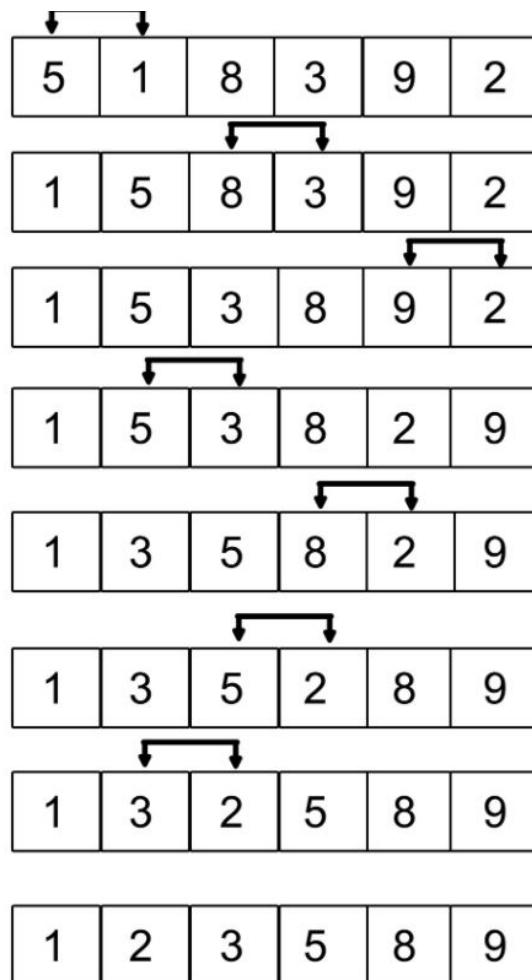
Selection sort moves the element with a minimum value to the front of the array, thus starting to form a sorted subarray. This process is repeated in multiple iterations over the unsorted subarray and the minimum element is moved to the sorted subarray.

Complexity class: $O(N^2)$

Bubble Sort

Bubble Sort repeatedly swaps adjacent elements in an array so that they are sorted.

Complexity class: $O(N^2)$



Classes and Other Object Oriented Programming Topics

Object oriented programming uses classes and objects to model real world systems and their interactions.

Classes

A class is like a blueprint from which objects are instantiated. Classes contain methods (behaviors) and attributes (characteristics). Special methods called constructors are used for initializing new objects. For example, if Employee is a class, employee objects such as “David Jones” can be instantiated from it. Multiple objects (employees) can be instantiated from the same class.

Objects

An object is instantiated from a class, its blueprint and has a state (the attributes or properties of an object), behaviors (methods) and identity: a unique name to identify an object and allow it to interact within a system of objects.

Inheritance

Inheritance is a feature of OOP where a class, the subclass, can inherit, reuse and if needed, extend, the methods and properties of another class referred to as the superclass. Some languages such as Java support single inheritance, whereas others, such as Python support multiple inheritance. Java provides additional features, called interfaces, that work similarly to multiple inheritance.

Polymorphism

Polymorphism is a feature of OOP which allows for different types of objects to be treated through a single interface, depending on the context.

Encapsulation

Encapsulation, or information hiding, refers to the data of a class being hidden from the “outside world” by declaring this data as private rather than public.

Recursion

Recursion takes place when a function calls itself repeatedly until a base case is met. It is possible to have more than one base case. Some problems are best solved recursively. Recursive solutions are considered elegant, but they are not always more efficient.

System Design

System Design is about creating complex systems while applying sound design principles such as separation of concern, abstraction, modularity, progressive enhancement.

Scalability problems

Scalability refers to a system's ability to scale up with an increasing load, such as when a website's traffic increases during a promoted shopping event. You should be able to compute the load experienced by the system while considering factors such as number of requests made, number of database calls, active users and amount of cache requests.

Regular Expressions

A regular expression is a sequence of characters that define a matching pattern for a search. For example, a regular expression could be used to validate if a credit card number entered by a user on a payment form is in the expected format.

Database topics

A database is a collection of structured data that can be accessed or stored on a computer system. A database can be managed with the use of a Database management system (DBMS). The components of a database include tables, which consist of rows and columns of data.

Structured Query Language (SQL) is used to create, read, update and delete data from relational databases. There are multiple flavors of SQL such as MySQL or PostgreSQL. There are also non relational databases that do not use SQL, such as MongoDB or Cassandra.

Internet and Web Technology

The Internet is a global network consisting of smaller networks that use several standard communication protocol layers including: an application layer which uses a HyperText Transfer Protocol(HTTP) and HTTPS protocol, a transport layer responsible for end-to-end communication over a network and a network layer which provides data routing.

The World Wide Web provides access to information on the internet via a markup language called HyperText Markup Language (HTML). HTML pages are hyperlinked to one another and if hosted on a web server accessible from a browser client via HTTP protocol and a Uniform Resource Locator (URL). Cascadian Style Sheets (CSS) are used to style web pages. JavaScript is used to provides interactivity.

Cloud topics

Cloud computing refers to the storing and accessing of data or programs via remote servers. Cloud computing architecture components include front end, back end platform, cloud delivery and a network. A couple popular cloud based services are AWS and Microsoft Azure. Many companies are transitioning from in-house hosting to cloud based hosting.

Big O Notation

Big O Notation, or efficiency, is used to classify algorithms based on how resource (space and time) requirements grow depending on the size of the input.

To determine an algorithm Big O notation, we assume the following:

1. Any single statement takes the same amount of time to run.
2. A method call's runtime is measured by the total of the statements inside the method's body.
3. A loop's runtime, if the loop repeats N times, is N times the runtime of the statements in its body.
4. The Big O complexity of a nested loop is $O(N^2)$
5. The highest order term determines Big O. For example, if we have $N^2 + 5N + 35$, the Big O complexity would be N^2 , because we are only concerned with the highest order term. The other two terms would be dropped.
6. Divide and conquer approach algorithms such as binary search have a complexity of $O(\log_2 N)$

How Big O Complexity is affected by doubling the value of N:

1. $O(1)$ constant. Time not dependent on N
2. $O(\log_2 N)$ logarithmic Time increase in a logarithmic fashion (not much)
3. $O(N)$ linear Time increases linearly as N increases
4. $O(N^2)$ quadratic Time quadruples as N doubles as N doubles
5. $O(N^3)$ cubic Time multiplied by 8
6. $O(2^N)$ exponential time increases greatly

8.6 A Word about Testing

There are several different kinds of tests used for testing code and software.

1. ***Edge test cases*** tests boundary conditions such as min, max values, or empty values such as empty strings.
2. ***Out of Bounds test cases*** address how does the code handle unexpected values, such as incorrect user input?
3. ***Functional test cases*** test the logic of your program. Does the method return the expected value, based on input parameters?
4. ***Randomized test cases*** take possible human bias out of the equation by running automated, randomized tests to check if the code works.
5. ***Load balancing testing*** is concerned with how the system performs under constraints such as high memory usage.

As a developer, you will be writing and running unit tests with tools such as jUnit.

These tests can be run manually, but in general they run automatically and produce a pass/fail result for each test case. It is critical to write meaningful tests and to run them before merging code with other developers' code.

During the interview focus on *functional, edge and out of bounds test cases*.

8.7 Coding Problems and Solutions

What follows are coding problems to work on to prepare for the interview. Some of the solutions are in Java, while others are in Python, so that you have the opportunity to practice with both.

In addition, there are many online platforms created specifically for coding interview practice. Some of these platforms have been listed under the *White boarding and virtual interview preparation* section of this book.



Coding Problem: FizzBuzz

Write a program that prints each number from 1 to 100 on a new line. For each multiple of 3, print "Fizz" instead of the number. For each multiple of 5, print "Buzz" instead of the number. For numbers that are multiples of both 3 and 5, print "FizzBuzz" instead of the number.

Hint: Use the % modulo operator, which returns the remainder of integer division.

Example

Input

```
int number = 5
```

Output

FizzBuzz

1

2

Fizz

4

Solution

```
public class FizzBuzz {  
  
    public static void main(String[] args) {  
        fizzBuzz(100);  
    }  
  
    public static void fizzBuzz(int number) {  
  
        for (int i = 0; i < number; i++) {  
  
            if (i % 3 == 0 && i % 5 == 0) {  
                System.out.println("FizzBuzz");  
            } else if (i % 3 == 0) {  
                System.out.println("Fizz");  
            } else if (i % 5 == 0) {  
                System.out.println("Buzz");  
            } else {  
                System.out.println(i);  
            }  
        }  
    }  
}
```

Coding problem: Reverse a String

Write a program that reverses a string. Assume that the input string is an array of characters. You may not use temporary storage, rather modify the input array in-place.

Example

Input

```
char[] input = { 'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd' };
```

Output

```
dlrow ,olleH
```

Solution

```
public class ReverseString {
    public static void main(String[] args) {
        char[] string = { 'H', 'e', 'l', 'l', 'o', ',', ' ', 'W', 'o', 'r', 'l', 'd' };
        char[] result = reverse(string);

        for (char ch : result)
            System.out.print(ch);
    }

    public static char[] reverse(char[] string) {
        for (int i = 0; i < string.length / 2; i++) {
            string[i] = (char) (string[i] + string[string.length - i - 1]);
            string[string.length - i - 1] = (char) (string[i] - string[string.length - i - 1]);
            string[i] = (char) (string[i] - string[string.length - i - 1]);
        }

        return string;
    }
}
```

Coding Problem with Arrays

Write a method returning true if an array's values are unique and false otherwise.

You can assume an array of integers.

Example

Input

```
int[] a1 = { 1, 2, 3, 4, 5 };
```

Output

```
true
```

Solution

```
public class ArrayUniqueValues {

    public static void main(String args[]) {

        int[] a1 = { 1, 2, 3, 4, 5 };
        int[] a2 = { 1, 2, 2, 4, 5 };

        System.out.println(unique(a1));
        System.out.println(unique(a2));
    }

    public static boolean unique(int[] a) {
        |
        for (int i = 0; i < a.length; i++) {
            for (int j = i + 1; j < a.length; j++) {
                if (a[i] == a[j]) {
                    return false;
                }
            }
        }
        return true;
    }
}
```

Coding Problem with Linked Lists

Write a method returning a list of all prime numbers up to a given number.

Implement using the [Sieve of Eratosthenes algorithm](#).

Example

Test

```
primesSieve(10)
```

Output

```
[2, 3, 5, 7]
```

Solution

```
import java.util.Iterator;
import java.util.LinkedList;

public class SievePrimes {

    public static void main(String[] args) {
        System.out.println(primesSieve(10));
    }

    public static LinkedList<Integer> primesSieve(int maxNum) {
        LinkedList<Integer> primeNums = new LinkedList<Integer>();
        // create a list to hold numbers from 2 to maxVal
        LinkedList<Integer> nums = new LinkedList<Integer>();
        nums.add(2);
        for (int i = 3; i <= maxNum; i = i + 2) {
            nums.add(i);
        }

        double sqrt = Math.sqrt(maxNum);
        while (!nums.isEmpty()) {
            // remove a prime number from the front
            int front = nums.remove(0);
            //add to the front of the primes list
            primeNums.add(front);
        }
    }
}
```

```
        if (front >= sqrt) {
            while (!nums.isEmpty()) {
                primeNums.add(nums.remove(0));
            }
        }
        // if a num is multiple of the prime number, remove it
        Iterator<Integer> itr = nums.iterator();
        while (itr.hasNext()) {
            int current = itr.next();
            if (current % front == 0) { //multiple of prime
                itr.remove();
            }
        }
    }
    return primeNums;
}

}
```

Coding Problem with Stacks

Write a method accepting two stacks of integers as parameters and returning whether the two stacks contain the same integers in the same order.

Example

Test

See code for declaration of s1 and s2 in the code below

```
same(s1, s2));
```

Output

```
false
```

Solution

```
import java.util.Stack;

public class SameStacks {

    public static void main(String[] args) {

        Stack<Integer> s1 = new Stack<Integer>();
        Stack<Integer> s2 = new Stack<Integer>();

        s1.add(1);
        s1.add(2);
        s1.add(3);

        s2.add(1);
        s2.add(1);
        s2.add(3);

        System.out.println(same(s1, s2));
    }

    public static boolean same(Stack<Integer> s1, Stack<Integer> s2) {
        // use a helper stack
        Stack<Integer> s3 = new Stack<Integer>();
        boolean same = true;
```

```
while (same && !s1.isEmpty() && !s2.isEmpty()) {
    int n1 = s1.pop();
    int n2 = s2.pop();
    if (n1 != n2) {
        // stacks do not contain the same values
        same = false;
    }
    s3.push(n1);
    s3.push(n2);
}
same = same && s1.isEmpty() && s2.isEmpty();
while (!s3.isEmpty()) {
    s2.push(s3.pop());
    s2.push(s3.pop());
}
return same;
}

}
```

Coding Problem with Hash Maps

Write a method that reverses a string map's keys to become its values and its values to become its keys.

Hint: Use the Java collections Map and HashMap.

Example

Input

```
HashMap<Integer, String> map = new HashMap<Integer, String>();  
    map.put(1, "Joe");  
    map.put(2, "Sofia");  
    map.put(3, "Violet");  
    System.out.println(reverse(map));
```

Output

```
{Joe=1, Violet=3, Sofia=2}
```

Solution

```
import java.util.HashMap;  
import java.util.Map;  
  
public class ReverseMapKV {  
  
    public static void main(String[] args) {  
  
        HashMap<Integer, String> map = new HashMap<Integer, String>();  
        map.put(1, "Joe");  
        map.put(2, "Sofia");  
        map.put(3, "Violet");  
  
        System.out.println(reverse(map));  
    }  
  
    public static Map<String, Integer> reverse(Map<Integer, String> map) {  
        Map<String, Integer> reversedMap = new HashMap<String, Integer>();  
        for (Integer key : map.keySet()) {  
            String value = map.get(key);  
            reversedMap.put(value, key);  
        }  
        return reversedMap;  
    }  
}
```

Coding Problem with Recursion

Find the nth Fibonacci number in the Fibonacci sequence. Solve the problem recursively.

Input

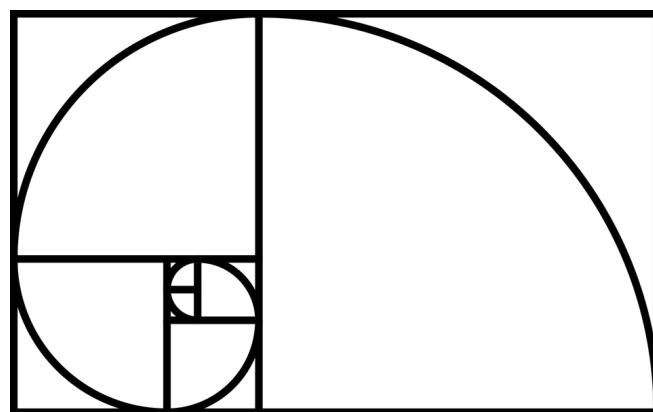
fib(8)

Output

21

Solution

```
public static int fib(int n){  
    if(n == 0){  
        return 0;  
    }  
    if(n == 1 || n == 2){  
        return 1;  
    }  
    return fib(n-2) + fib(n-1);  
}
```



Coding Problem with Strings and Recursion

Write a recursive program that checks if a string is a palindrome (a word or sentence that reads the same forwards and backwards)

Example

Input

```
s1 = 'cat'  
s2 = 'radar'  
palindrome(s1)  
palindrome(s2))
```

Output

False

True

Solution

```
def palindrome(s):  
    if len(s) < 2:  
        return True  
  
    elif s[0].lower() != s[-1].lower():  
        return False  
  
    else:  
        return palindrome(s[1:-1])  
  
def main():  
    s1 = 'cat'  
    s2 = 'radar'  
    print(palindrome(s1))  
    print(palindrome(s2))  
  
main()
```

Coding Problem with Sorting

Implement a Merge Sort Algorithm

Solution

```
import java.util.*;  
  
public class MergeSort {  
  
    public static void main(String[] args) {  
  
        int[] list = {14, 32, 67, 76, 23, 41, 58, 85};  
        System.out.println("before: " + Arrays.toString(list));  
        mergeSort(list);  
        System.out.println("after: " + Arrays.toString(list));  
    }  
  
    // Sort the elements of an array using the merge sort  
  
    public static void mergeSort(int[] array) {  
        if (array.length > 1) {  
            // split array into two halves  
            int[] left = leftHalf(array);  
            int[] right = rightHalf(array);  
  
            // recursively sort the two halves  
            mergeSort(left);  
            mergeSort(right);  
  
            // merge the sorted halves into a sorted whole  
            merge(array, left, right);  
        }  
    }  
}
```

```
// Process the first half of the input array
public static int[] leftHalf(int[] array) {
    int size1 = array.length / 2;
    int[] left = new int[size1];
    for (int i = 0; i < size1; i++) {
        left[i] = array[i];
    }
    return left;
}

//Process the second half of the given array
public static int[] rightHalf(int[] array) {
    int size1 = array.length / 2;
    int size2 = array.length - size1;
    int[] right = new int[size2];
    for (int i = 0; i < size2; i++) {
        right[i] = array[i + size1];
    }
    return right;
}
```

```

// Merge the left and right Arrays
//resulting in a sorted array

public static void merge(int[] result, int[] left, int[] right) {
    int i1 = 0;    // index into left array
    int i2 = 0;    // index into right array

    for (int i = 0; i < result.length; i++) {
        if (i2 >= right.length || (i1 < left.length &&
            left[i1] <= right[i2])) {
            result[i] = left[i1];    // take from left
            i1++;
        } else {
            result[i] = right[i2];   // take from right
            i2++;
        }
    }
}

//end Merge sort implementation

```

Coding Problem with Object Oriented Design

Create a deck of cards

Hint: To accomplish this task, create a class to model a Card and a class to model a Deck.

Example

Test

```
deck = Deck()
deck.shuffle()
#deck.show()
print('Top card', deck.deal())
deck.shuffle()
print('Top card after shuffle', deck.deal())
```

Output (Exact card values will vary due to the use of random.shuffle())

```
Top card 3 of Diamonds
Top card after shuffle 10 of Clubs
>>> |
```

Solution

```
import random

class Card():
    def __init__(self, suit, value):
        self.suit = suit
        self.value = value

    # Override built in methods
    def __unicode__(self):
        return self.show()
    def __str__(self):
        return self.show()
    def __repr__(self):
        return self.show()
```

```

def show(self):
    if self.value == 1:
        value = "Ace"
    elif self.value == 11:
        value = "Jack"
    elif self.value == 12:
        value = "Queen"
    elif self.value == 13:
        value = "King"
    else:
        value = self.value

    return "{} of {}".format(value, self.suit)

from Card import *

class Deck():
    def __init__(self):
        self.cards = []
        self.build()

    # Generate all cards
    def build(self):
        self.cards = []
        for suit in ['Hearts', 'Clubs', 'Diamonds', 'Spades']:
            for val in range(1,14):
                self.cards.append(Card(suit, val))

    # Show all cards
    def show(self):
        for card in self.cards:
            print(card)

```

```
# Shuffle the deck
def shuffle(self):
    random.shuffle(self.cards)

# Return the top card
def deal(self):
    return self.cards.pop()

def main():

    deck = Deck()
    deck.shuffle()
    #deck.show()
    print('Top card', deck.deal())
    deck.shuffle()
    print('Top card after shuffle', deck.deal())

main()
```

System Design question

Design a TinyURL system. TinyURL shortens a long URL and returns a shorter URL. The system can also turn a short URL into the original long URL.

Solution (solutions to this type of questions may vary)

Solution approach

Generate a short, unique URL from the given long URL. The TinyURL should redirect to the original URL. Handle redirects, track statistics, delete expired URL, provide high availability and ability to scale up if there is an increase in the TinyURL requests. Use REST API to communicate with the service. The front of the service should support a load balancing feature. The application layer will be multi threaded and persisted in a database, where both the original URL and the TinyURL will be stored.

▶ ▶ ▶ Hands-on Activity

Create and follow a study schedule for the topics outlined in this chapter.

The timeline will vary depending on your background. In some cases, you might benefit from enrolling in a formal educational program, as discussed in a previous section of this book. Set aside an hour or two (or more if you can) to work on interview related technical topics. Remember that nothing beats hands-on practice and repetition.





Hands-on Activity

1. Work through each coding problem and try solving it on your own . You can use Repl for both Java and Python. To work on those problems on your local computer, you can use use IDLE (comes installed with Python) or Eclipse for Java. All software mentioned here is free.
2. If you are unable to solve the problem on your own, hand type the solution, run it and study it until you understand it. Can you think of additional test cases to add?
3. Using the guidelines included in this chapter, determine what is the Big O notation for each problem. What is your rationale?
4. Practice answering the non coding problems, such as the System design question.

▶ ▶ ▶ Hands-on Activity

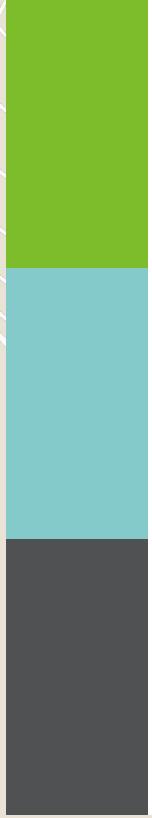
Interview Practice

Virtual interview practice is now possible with online platforms such as [prampt](#) or [leetcode](#). Try a mock interview and get ready for the real interview.

In Person interview practice normally takes place in front of white boards, in venues such as colleges or meeting rooms in libraries or companies. Get together a study group and take turns practicing being the interviewer and the interview.

Note: at the time of this writing in person mock or real interviews are generally not conducted due to Covid 19, so you might want to focus on the Virtual Interview.





Chapter Nine

Soft Interview Questions

Soft Interview questions

Soft interview questions generally fall in one of the following categories:

- ***Behavioral questions*** aim at assessing your personality, abilities and skills based on your prior experiences.
- ***Situational questions*** are similar to behavioral questions but are set in a hypothetical situation.
- ***Value based questions*** assess your personal values, and if you will be a good fit for the company.
- ***Broader questions about your technical knowledge and experience*** aim at assessing your overall technological literacy, and if you keep up with current technologies.
- ***Education related questions*** aim at uncovering more details about your educational background such as languages learned or projects you worked on.



A helpful framework for answering soft interview questions is **STAR**. **STAR** stands for:

Situation: Describe a specific situation that you were in or a task that you needed to accomplish. Provide enough detail for the interviewer to understand the context and your role. Ideally, the situation will be from a previous job, however, school projects or volunteer experiences are also acceptable.

Task: What was the project you were working on? What was your role on the team? Remember to address your role, not the team project at large.

Action: Describe what actions you took in regards to the situation. Again, remember to discuss your role, not the team as a whole. Use “I” not “we” when describing the actions taken. Be as specific as possible.

Result: What was the outcome? What was your part in achieving the outcome? What were the lessons learned? Was there a customer success story or process improvement to share? Be as specific as possible.

Additional tips on preparing for soft interview questions

Prior experience questions Your answer should focus primarily on your (not the team’s) actions, accomplishments, and decisions. Remember to use “I” rather than “we.” Your answer should be relevant to the question asked. Examples of prior experience questions:

1. Can you tell me about a time when a project you were working on fell behind schedule? What was the cause and would you do differently next time?
2. Was there a time when you and a colleague you didn't get along? How did you handle the situation?
3. Describe a time you didn't agree with a decision made by the team lead. How would you have handled the situation?
4. Describe a time you fell behind schedule.

Situational questions (hypotheticals) This type of question examines how you apply your past experience or skills to solving hypothetical issues. When answering hypotheticals don't go into too much detail unless you have studied the company and can include a specific company detail in the context of the question. Talk to the interviewer through your thought process. Hypothetical questions usually begin with: "If." "What if.", "You are asked to..."

1. You are asked to cut your team's spending budget by 20%. How would you do it?
2. What if you were tasked to design an ATM for children? Would you do it, and if so, how would you approach this problem?

Values-Based Questions allow the interviewer to assess your personal values and whether you are a good fit for the company.

1. What resources do you use to keep technology skills current?
2. What are your favorite and least favorite technology products? Discuss why.
3. How do you think technology advances will impact your job?
4. What are the qualities of a successful team leader?
5. How do you handle tight deadlines?
6. Why do you want to work for us?

Broader questions about your technical knowledge and experience or Education related questions aim at assessing your overall technological literacy and if you keep up with current technologies. Frame an appropriate example from your prior experience and answer it in the STAR format.

Do your research on the company Study the company's mission statement, information about the company culture or any news in the media. For example, Amazon interviews are centered around Amazon's Leadership principles, which you can see [here](#).

Dress Code

While most tech companies have a casual dress code, you can research a company's dress code and company culture by looking at photos on the company website or social media.

Another consideration is the position you are applying for. Customer facing account managers will likely wear a more formal attire than software engineers.

A rule of thumb is to dress one level higher than the company dress code.. For example, if the dress code in the company is t-shirts and jeans, you might want to wear dark jeans or slacks and a casual button down shirt.

Should you wear a skirt to the interview? This is a personal question that most likely will not (and should not) have an impact on the outcome of your interview.

In general tech companies are more interested in your skills and if your personality is a good fit rather than in your outfit.

If you are still unsure how to dress for your interview, reach out to your recruiter and clarify.

▶▶▶ Hands-on Activity

Choose 5 of the interview questions listed in this chapter and write down your answers. Use the STAR method to frame the non hypothetical questions.





Chapter Ten

Negotiating the Offer

How to successfully negotiate your job offer

Negotiating a job offer might feel uncomfortable, but it is worth it. It could cost you a substantial amount over the course of your career if you choose not to negotiate. Note that every company and every situation is different and should be approached as such.

What can you negotiate?

Focus on 2-3 items from this list that are most important to you and prepare for negotiations.

- Salary
- Job title
- Job start date
- Relocation expenses
- Vacation
- Signing bonus, bonuses
- Stock options
- Training costs
- Virtual work availability
- If contracting, terms of contract

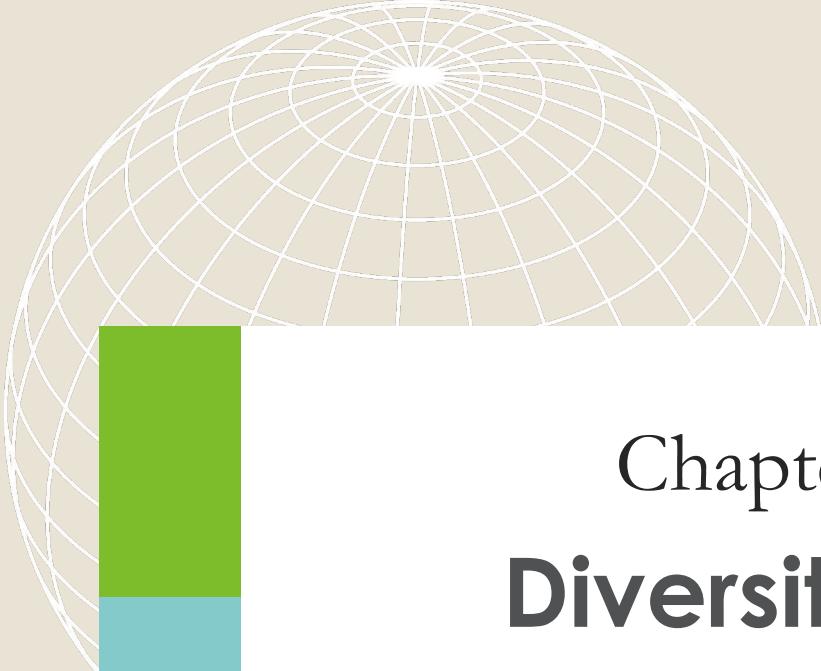
How to negotiate the offer

1. Start negotiating after you have received an offer. The offer should be presented to you in writing and include the salary and other offer details. Once you have received an offer, express your enthusiasm for the job and ask what is the timeline for you to evaluate the offer.
2. Receiving an offer makes it official that the company wants to hire you. It takes time and resources for companies to hire new employees, so they have an incentive to complete the process. Based on the original offer or the negotiated terms, it is up to you to decide whether to accept the job or not.
3. Start the negotiations with your salary. Knowing if you won the salary negotiation will tell you how to approach the rest of your negotiation. If you did not get the salary increase, push harder for the other items on your list.
4. Approach the negotiation by asking questions rather than making demands. For example, you could ask “Based on my knowledge of SQL, I was expecting a higher salary. Is there anything we can do about that?” This brings us to the point that you need to do your research and find out what is the average salary for the job role and geographical region.

5. Remember that the recruiter is on your side: they have a financial incentive to get you hired.
6. Treat this negotiation as a business transaction. Don't let your emotions get the best of you.
7. Be polite and positive during the negotiation.
8. Get the new offer, if any, in writing. Be sure to review it carefully before accepting.
9. Be prepared to walk away. If your counter offer is not accepted, you can still accept the original offer or walk away. The choice will ultimately be yours.
10. Do you have to negotiate an offer? No, you don't, but it may be in your benefit to do so. You can certainly accept an offer as is if it meets your criteria.

What if you didn't get an offer?

1. Don't take rejection personally and keep a positive outlook. Remember, it is an experiment, not a failure, unless you stop trying.
2. Reflect on how the interview went. You can ask for feedback, although some companies do not provide feedback. Reflect on the technical content of the interview, as well as what message did you convey about yourself as a potential employee and team member.
3. Practice mock interviews.
4. Re-evaluate the positions you are applying for. Should you consider a different role as your target?
5. If you think that you are aching the interviews, but are not getting an offer, consider getting help from a mentor or tech career coach.



Chapter Eleven

Diversity in Tech

How to Increase Diversity in Tech

Lack of diversity in tech is a known issue which impacts the tech workforce as well as the software and algorithms produced by tech companies. This lack of diversity can be addressed by educational institutions, companies and the tech workforce. Here we offer some suggestions how to do so:

1. Rethink education systems to better reflect the technological landscape.
2. Incentivizing lifelong learning.
3. Include (give voice) to diverse demographics in the building of various technologies such as AIs.
4. Analyze and adjust job description wordings, recruitment strategies and interview processes that might defer qualified candidates from underrepresented groups.
5. Develop leadership and sponsorship programs to help the retention and advancement of under represented groups.
6. Prioritize creating an inclusive company culture which is friendly to women and underrepresented groups.
7. Include and engage majority groups in company efforts to increase diversity.
8. Quantify diversity and inclusivity efforts and representation in the company with the use of data and metrics. Share the findings.
9. Offer equity, inclusivity and diversity training.
10. Offer flexible work policies and examine performance reviews and promotion processes.

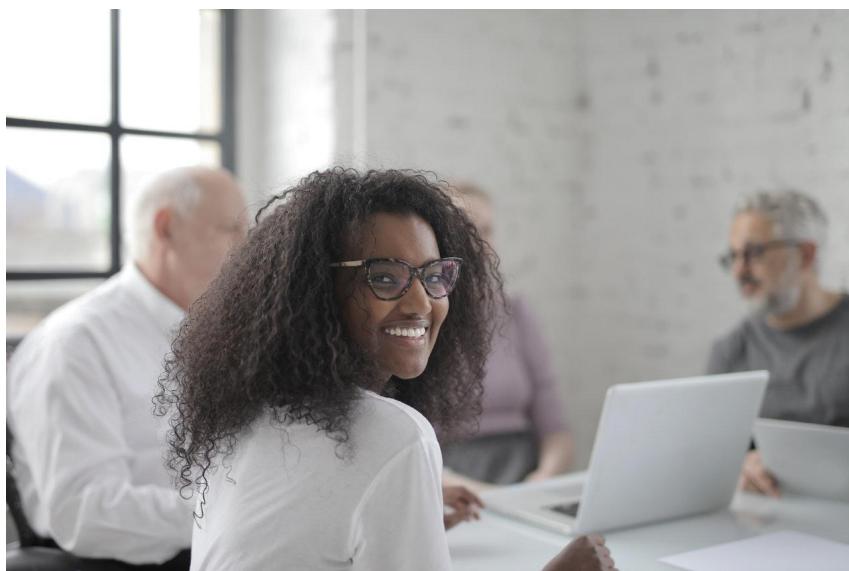
This visualization by informationisbeautiful.net demonstrates the employee breakdown in some of the top tech companies in the US.

Conclusion

By now you are familiar with the steps involved in landing a job in tech. Hopefully you have also completed the Hands-on Activities and set a study schedule for yourself.

With practice, consistency and reliance on a process that has worked for many other tech job seekers from traditional and non traditional backgrounds, you are on your way to achieving your professional goals and landing a job in tech.

Best of luck on your journey!



Images used in this publication are either public domain or used under the Creative Commons Zero (CC0) license.

While every effort has been made to ensure that the live links in this material are up to date, we cannot guarantee that this is the case, since those links are maintained by external entities.

