Apprenticeship Interview 101

**What is an apprenticeship?**  
An apprenticeship is a paid program that provides a pathway into Software Engineering for people from non-traditional computer science backgrounds (e.g. bootcamps, online learning, self-guided learning).

You can think of it as a 12 month long internship. The goal of an apprenticeship is to provide people with the support, project experience and structure to become productive software engineers at the company.

**Who are apprenticeships for?**

Most of the software engineering apprenticeships we are seeing these days at Tech companies are for:

* People who know how to code from non-traditional tech backgrounds (those without a Computer Science degree - this includes bootcamp grads)
* People re-entering the workforce after taking some time off
* Self-taught engineers

**Are apprenticeships paid?**

Yes, all of the apprenticeships we know of pay participants. You can find more information about software apprenticeships [here](https://www.apprenticeship.io/)

**How long do apprenticeships last?**

Most apprenticeships last anywhere from 6-36 months with most in the 6-18 month range.

# Interview Preparation

General Interview Tips

* Study the company before you go, this will help with questions you can prepare for them
* Listen carefully and ask for clarification if you don’t understand something
* Don’t rush into coding. Take your time to understand the question. Make sure to repeat the question in simple terms to the interview. Doing so will help convince yourself that you understand what you’re being asked.
* Ask questions for clarification if you don’t understand the question or a part of it or just want to make sure you understood the task correctly
* Be confident in the programming language you are using. Use a coding language you are very familiar with.
* Don’t be afraid to draw things out if that helps you understand the questions better
* Simplify the problem first. Think about solving the simplest cases of the problem first and then handle more complex, edge-cases afterwards.
* Talk. The most important thing in an interview is to understand how you think. By talking about what’s on your mind as you go through problem solving, we can better understand what you’re doing and why, as well as nudge you in the right direction if you’re heading astray.
* If you get stuck, try to verbalize what exactly is going on in your head. Saying something like *“I’m thinking about doing X, but it may not work because Y*” is much better than staying quiet because the interviewer doesn’t know what you’re thinking if you’re silent.
* Easier said than done, but don’t be nervous, take deep breaths. The people interviewing you are people too and they are probably a bit nervous as well.
* If you need to, draw out the problem and proposed solution or pseudo code (write your solution out in steps in plain english or break down the problem into just functions, and propose it before you start coding.
  + - • I.e. fun solveTheProblem() {

doThing1()

doThing2()

formatOutput()

}

* Don’t worry about perfect. We know it’s a stressful environment. It’s better to have a working solution than incomplete pretty code.
* Ask for hints if you’re stuck! Sometimes interviewers give you a question they know you will need to team up on, so think of your interviewer as your team member and ask for hints and/or clarifications if you get stuck
* Be open to feedback and criticism. Do not ignore suggestions your interviewer makes. Take them into consideration and then respond to it whether you agree or disagree.
* **Interviews aren’t black and white**. It is still possible to get hired without solving the whole problem. If you get stuck, the most important thing is to never give up. Keep trying for the duration of the interview even if it feels like you’re not making forward progress.

Resources for Practicing Technical Questions

* [**Leetcode**](https://leetcode.com/) - Classic algorithm and data structures questions.
* [**interviewing.io**](http://interviewing.io) - Free anonymous interviews with engineers at Google, FB, Pinterest, Uber,Airbnb etc.
* [**Pramp**](https://www.pramp.com/#/) - Mock interviews with other jobseekers
* [**Outco**](https://www.outco.io/) - a four week hands-on course that teaches technical and non-technical curriculum for acing the software engineering interview
* [**Big O Explanation**](https://www.bigocheatsheet.com/)
* [HackerRank](https://www.hackerrank.com/)
* [Interview Cake](https://www.interviewcake.com/)
* **Youtube** - [CS Dojo](https://www.google.com/search?q=cs+dojo&oq=cs+dojo&aqs=chrome..69i57l2j69i59j69i60l5.1460j0j1&sourceid=chrome&ie=UTF-8)

Coding Challenge

The first step is the coding challenge which is usually done by a Karat. They let you know it will be a shared screen coding challenge and after the challenge you also discuss a project you’ve worked on in the past . A great way to prepare for the coding challenge is by doing interview prep questions on HackerRank, Leetcode, Interview.io, etc.



**Preparation tips**

Coding challenges are usually intended to evaluate an interviewee’s problem-solving ability. So it helps to have a general approach for solving problems regardless of what the algorithm is. One such approach is to use this 4-step formula for tackling algorithm-style coding challenges. The steps are

1. Ask questions
2. Simplify and diagram
3. Pseudocode
4. Code

|  |  |
| --- | --- |
| **Step** | **Context** |
| Ask questions | The goal here is to establish an understanding of the question. For example, if you’re given an array (or list) of numbers as an input to the question, and asked to check for duplicate values, you can ask   * Is the array sorted? * Are the numbers integers or floats? * How large could the array be? * Will there at least be one duplicate? Or could there be a set of three copies (or four copies of a number etc)   You should also have a good understanding of the inputs of the problem and the outputs your solution should provide. For the example question above, you should confirm that your input is an array, and the response should be a Boolean.   It usually helps to have multiple examples of the inputs and problems. You can create these yourself or ask the interviewer. For example, you can write example inputs and outputs such as   [10, 2, 2, 0, 911] ===> True [] => False [9, 18, 9, 18, 11, 67, 1.5] => True  Each of those examples could provide a different insight into the problem. |
| Simplify and diagram | After getting an idea of the problem, you should break it down to the simplest piece. That means, you should try to figure out how to “solve” for the easiest case. You don’t need to write any code to do this. For our example case, you need to understand what a duplicate is. And then figure out how to check for a duplicate in a collection.  Then you can take your example inputs and “walk through” your ideas to see whether they give the corresponding example outputs that are correct.  For example, consider the case [10, 2, 2, 0, 911] ===> True  You should be able to walk through this list/array and explain how and why your solution yields True. You can draw this out to make this clear to the interviewer. |
| Pseudocode | This step involves converting your solution approach into concrete steps in words. Here are two examples of pseudocode that could solve this problem   Attempt 1   * I’m going to first check if the array is empty, if it is, then I’ll say False * If the Array is not empty, then I’ll iterate or loop through the array * I’ll create an object or dictionary to keep track of numbers I’ve visited before (during the iteration) * Everytime I visit a number, I’ll check the object to see if I’ve seen that number before. If so, I’ll return True. If not, I’ll add the new number to the object * If I make it to the end of the array without seeing a number that I saw before (i.e. a duplicate), then I’ll return False.   Attempt 2   * I’m going to create a Set of the values in the Array * And then I’m going to compare the size of the Set with the size of the input array. If they are different in size, then I know that there was at least one duplicate so I will return True. If not, then I will return False.   The goal here is to solidify the steps of your solution without writing code. It’s possible that you’ll have a bug in your code the first time you write it so it’s helpful to have a recipe or a list of steps to refer back to. |
| Code | Finally, you can write the code that translates the words you wrote in step 3 above into code. By this point, you should have already confirmed that your approach or solution works. You’re just translating that approach into your preferred programming language. It is possible to pass the interview with bugs in the code if your logic or approach is solid. |

You can improve upon this method by adding an optimization step at the end in order to fulfil the desired time and space complexities that your interviewer may want. Or you can repeat steps 2, 3 and 4 for the more complicated extensions of the problem. Here’s an [extended guide](https://static1.squarespace.com/static/58028c1e29687f239061ebb1/t/5c9ab0409140b799e030e5b4/1553641547360/Outco+-+Solving+Any+Algorithm.pdf) from Outco for problem-solving in general (some of this content came from there)

Researching the company

You can spend anywhere from a few minutes to an hour researching a company! The goal here is to be able to have an opinion on their product and be able to talk about the company as a whole. What do you like? What would you change? What drew you to the company?

Interviewers love to see interviewers who care about the company’s product. So make sure to do some research.

## Tips and Tricks:

* Use the product!
* Check the company’s social media accounts and Medium blog
* Use your connections
* Look for recent news
* Check the company’s profile on Glassdoor and the Muse

**Interview Process**

1. Apply online
2. Coding challenge
3. Recruiter prescreen
4. Phone screen
5. Onsite interview
6. Offer!🎉

Phone Screen

### Expectations

* + A recruiter will call you to talk about your experience, your interests and what you’re looking forward to doing next. This is also a great time to ask questions about open roles and/or the current team you are interviewing for.

### How to prepare

* + Look over your resume and think about all of the projects/teams you’ve worked on.
    - Make sure to talk about the coding languages, frameworks, tools and libraries that you have used throughout your time building things
  + Make sure to talk about passion projects and side projects
  + Think about why you want to work for this specific company, what makes them special and what you feel you can contribute
  + This is your time to shine and show the company why they should continue to invest their time into interviewing and hiring you.
  + Common Questions: <https://skillcrush.com/blog/answer-the-toughest-tech-interview-questions/>

Tech Screen

### Expectations

* + You will probably receive a link to a platform like Karat or Codepad to solve a technical puzzle
  + You will probably be asked some general platform specific trivia-like questions so that the interviewer can gauge your knowledge of the platform
  + **Example question:**
    - What is object oriented programming?
    - Where does JavaScript run?
    - How can you debug <language of choice>?
    - Question

We have a list of numbers, and another number. We want to know whether there are two

numbers in the list that sum to the other number.

For example:

* + - * sum\_in\_list([3, 4, 5], 8) -> True
      * sum\_in\_list([3, 4, 5], 10) -> False
      * sum\_in\_list([3, 4, 5], 11) -> False
    - Write a program that takes a filename as input and prints the number of words in the file as output.
      * Estimate how much memory is used by your program (relative to the file size).

### How to prepare

* + Run through some practice problems, there are plenty of sites that will have interview-style problems that you can work on (HackerRank and Leetcode are some options)
  + Set a timer while you’re practicing - this will help you get used to a real interview’s constraints
  + Practice talking out loud while solving problems. It may feel silly to talk to yourself but it’s a great way to get into the habit for the real day!

### Tips

* + Make sure to have a quiet place with internet connection for the day of the screen
  + Prepare a quick introduction speech - who you are, what you’re excited about
  + Use headphones with microphones or put your phone on speaker - you’ll need your hands for typing!

On Site

### Expectations

* + Pair programming on a computer
    - Make sure to play with the product of the company you are interviewing with to think of what sorts of things they may ask you to build when they bring you on site. A good way to think about this is to consider what their major surfaces are (for social media sites this is usually a feed) and what problems they may encounter.
  + Algorithms and Data Structures (whiteboarding)
  + As you prepare, play around with the product you’d be working on if you were hired. Think about what their major surfaces are, how they are built and what sorts of problems they’d have to solve on a regular day. Think about the foundations of the product in order to come up with possible questions they could ask you during the interview.

### How to prep

* + Practice whiteboarding during your free time. Find a friend who already works in the tech industry to help you practice if you can.
  + Know the in’s and outs of data structures. This [Youtube playlist](https://www.youtube.com/playlist?list=PL2_aWCzGMAwI3W_JlcBbtYTwiQSsOTa6P) is extremely helpful.

### Tips

* + Checkout the [Pinterest Guide on Technical Interviews](https://medium.com/pinterest-engineering/a-pinterest-engineering-guide-to-technical-interviews-1c2471c2d139) and [this newer guide](https://medium.com/pinterest-engineering/what-its-like-to-interview-at-pinterest-e40f05a018f9)
  + Or this guide on [interviewing at Google](http://steve-yegge.blogspot.com/2008/03/get-that-job-at-google.html)
* Example question:
  + You are on line at the barbershop/salon. There are n people before you and x barbers. Each barber cuts at a rate of r which is represented in an array of barber times in units of time. Ie; barber 1 cuts at a pace of 1 unit of time per person, barber 2 cuts at a pace of 3 units of time per person, etc. Write an algorithm that tells you exactly how long it will take before you sit down.

public int getTimeBeforeYouSitDown(int numOfPeopleBeforeYou, int NumOfBarbers, int [ ] barberTimes)

Asking the right questions

**Why is it important to ask questions?**

• It’s an opportunity for you to get your questions answered.

• It also demonstrates your curiosity and interest in the company!

**What kind of questions should I avoid?**

• Anything you can find online or ask your recruiter

• Personal questions about your interviewer e.g. the projects they work on, technologies they use etc

**What if I don’t have any questions?**

• Please have questions!

• Some interviews can end early, but not asking questions can demonstrate a

lack of curiosity in the company and you will miss the opportunity to present

yourself well.

**How do I come up with questions for an interview?**

• Think about what you would like to know about the position in particular, or the roles in general

* How many people will you work with?
* How can you get support from others?

• Think about what is important for you to know!

* What sort of work environment do you want to be in?
* Does the company have documentation for you to read or will it be more hands on?

The questions you ask will be personal to you, different people have different needs and priorities and the questions you ask will reflect you in that way.

Try to come up with a core set of questions you ask every interview to make them easier to compare - this will change over time till you find what’s best for you!

Further Reading and Resources:

## **Playlist to get good at understanding algorithms and data structures:**

* + <https://www.youtube.com/playlist?list=PL2_aWCzGMAwI3W_JlcBbtYTwiQSsOTa6P>
* **Live list of apprenticeships:** 
  + <https://apprenticeships.me/>