$10 M

The Pebble smartwatch raised more than 100 times its funding goal on Kickstarter.

CAREERS

The Google Technical Interview
How to Get Your Dream Job

Most successful students would not consider taking a final exam without preparation. For students—undergraduate and graduate—about to head into industry, job interviews benefit from the same level of consideration. There are many books and articles on interview skills, and most academic career centers offer additional training. What this article covers is more specific: A highlevel view of Google's engineering interviews.

The interview process at Google has been designed (and redesigned!) from the ground up to avoid false positives. We want to avoid making offers to candidates who would not be successful at Google. (The cost of this unfortunately includes more false negatives, which are times when we turn down somebody who would have done well.) The recruiters and engineers you will speak with want to see where you shine, whether you can do the job, and make sure you're someone they want to work with. This article is designed to help both you and Google achieve those goals—and help the interview be an interesting, even pleasant, experience, too.

You will meet at least two types of Googlers (Google employees) in the interview process. The first are our recruiters. Recruiters are nontechnical employees who are experts at both finding candidates and helping them through the interview process. The second are our technical interviewers; they are fulltime engineers who volunteer to help with the hiring process by interviewing candidates like you. All of our Googlers come from academic backgrounds and from industry, and can answer most (if not all) of the questions you might have, including whether Google is likely to be a good fit for you. So please ask away!

There is a standard format for most technical interviews. (Ph.D. students and more experienced candidates may be given one additional interview with a slightly different format, but similar advice applies.) For about 45 minutes you meet with a single technical interviewer, who will present a programming problem and ask you to work out one or more solutions to it. In some interviews, you will be asked to code up one of your solutions on a whiteboard. All of our questions have multiple solutions, and some of our questions do not have a single best answer, so if you have more than one solution, explain the tradeoffs or the benefits of your preferred solution.

Each interview day will have up to five of these 45-minute interviews, depending on your schedule, proximity to the nearest office, and interviewer availability. Candidates living farther away may start with phone interviews before proceeding to an onsite interview. The types of interviewers and which questions they ask are the same in both cases.

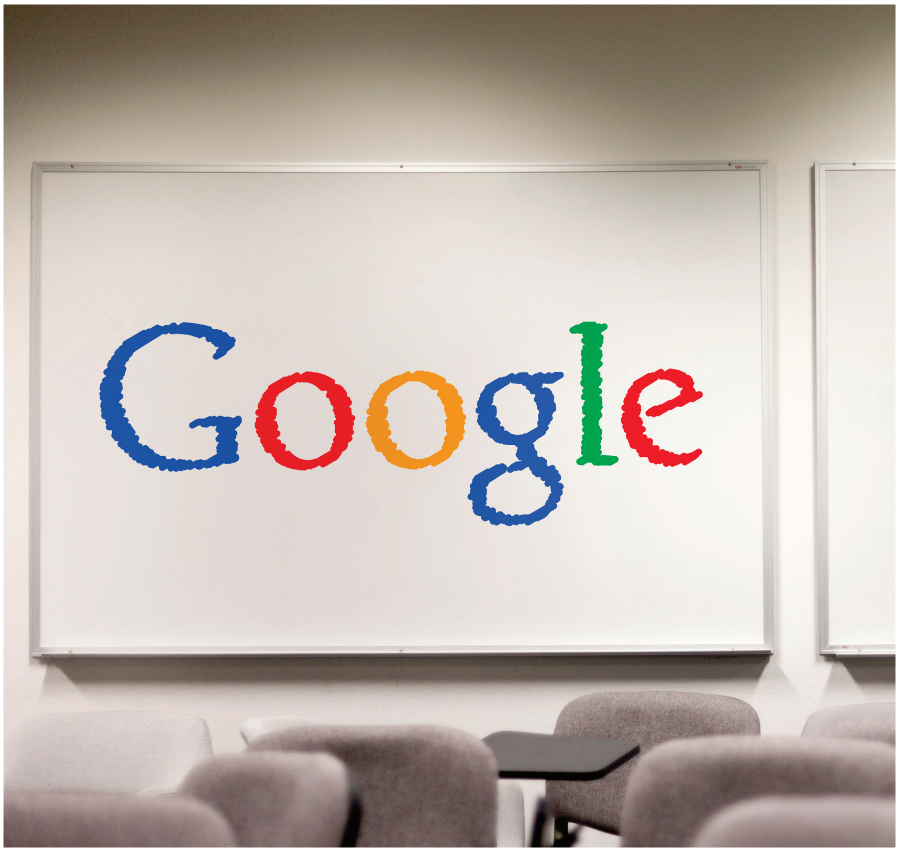
Overall, the interviewers are simply trying to decide one thing: Would you be a good fit for Google?

Let's break down a typical interview, piece by piece.

The programming problems are not "trick" questions, but they will always have aspects that require care and attention. First, make sure you understand the problem properly. It helps to clarify assumptions before diving in too deeply, and if you are confused about the question, do ask for examples, or for the question to be reworded. The interviewer will often not offer information until you ask for it. "How big could the input be?" "What happens with bad inputs?" and "How often will we run this?" are three common clarifications. If you're stuck, one thing to do is recheck your assumptions.

After you have the clarifications that you need, dive into solving the problem. There are usually several paths to a good solution. Much like math homework, it is essential to show your work; the way to do this is to talk to the interviewer, and explain what you are thinking. This is easy for some of us, but really hard for others, so it is important to practice this skill. If narrating your entire thought process will significantly reduce your ability to think on your feet, it is OK to be quiet for a minute or two—but then tell the interviewer what you considered, and why you chose what you chose. The more you can communicate your thought process, the better. If this might be hard for you to do, it is definitely worth practicing with a friend.

Those in the Quantified Self movement collect data about themselves with sensors,with the goal of making better life decisions.



Feel free to give answers you know are imperfect; explain briefly why they are not the best answer, and keep going. The first solution is rarely the best, and a sequence of answers very clearly shows your thought process toward solving the problem. It is also fine to start with a brute-force approach, as it gives an initial benchmark for the answers that follow. One trap to avoid is getting stuck thinking about incremental improvements to the worst algorithm; sometimes you will need to leap to another approach.

One of the most important things you should know is Big O notation and the analysis it represents. It is the common language for discussing algorithmic performance. While Big O is normally used to discuss how well an algorithm will scale, it is also good to consider disk, memory, network, and other needs for each solution.

At any point in an interview it is fine to ask if you can move to the whiteboard to take notes, draw diagrams, or explain what you are thinking. You are also welcome to use pen and paper if it will better help you keep your thoughts organized. In many interviews, the interviewer will ask you to move to the whiteboard, and then it is usually time to write some code.

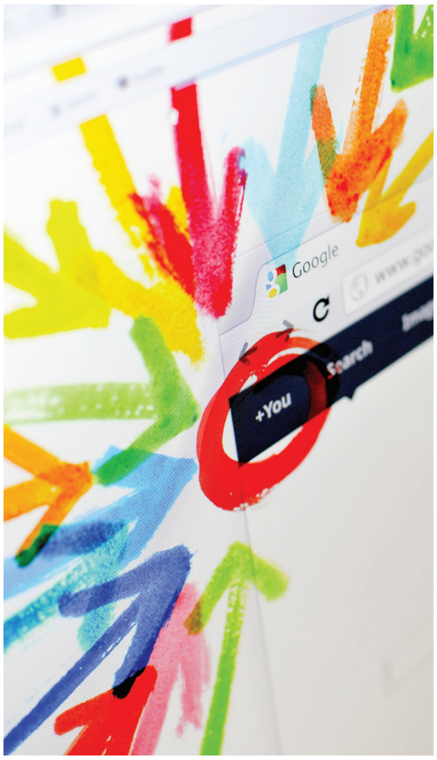
For our engineering positions, you will need to know a language like Java, C++, or Python. Knowing more than one is a nice touch, but you must know one well. For each interview with a coding component, you will usually write between 10 and 50 lines of code.

If you are rusty at coding, you should practice. If you code every day, you should still prepare by solving a few interview-like questions. Candidates who prepare do better. You should also practice writing code on a whiteboard, and explaining what you are doing while you do it. A whiteboard is significantly different than using your favorite editor or IDE. It also helps to have a friend or mentor review your practice code. As you know, code that works perfectly but is very hard for others to read is usually not a good idea in team-based software development. We are not worried about handwriting, but we do actively look for clean, maintainable code.

Testing code is important when working as an engineer, and it helps in interviews, too. After writing code, you should test it. Run a normal input, try the edge cases, and see if what you wrote behaves properly. This helps many candidates bump a mediocre answer into a significantly above-the-bar performance. When I interviewed for my position at Google, this was the difference between code that worked and a likely failure

If you are unsure about something, you should be open about your uncertainty. That will help you, not hurt you. We value honest communication. Likewise, if you are stuck, it is OK to ask for help if you need it—but fishing for answers is a bad tactic. If the interviewer gives you a hint, it's always a good idea to listen to it, and consider what they said: Their comments aren't random, and they are trying to help you. If you are working on an idea, and the interviewer is silent (or just taking notes), do not worry that something is going wrong. They are waiting for you, and

A smart electricity grid can adapt to prevent small failures from producing large blackouts.



most importantly, they are finding it worth the wait.

Data structures and algorithms are the most important classes in the undergraduate curriculum. (Really!) If your school offers higher-level algorithms classes, take those as well. The majority of successful applicants took these courses, enjoyed them, and studied the material recently. Steve Yegge wrote about this a few years back in his blog post, "Get That Job At Google." I owe my job to reading his post; you should read it too.

Overall, the interviewers are simply trying to decide one thing: Would you be a good fit for Google? Determining that involves answering several other questions. Are you someone they want to work with? Are you someone who would make their team better? Are you someone they want writing code they will use and depend on? Can you think on your feet? Can you explain your ideas to coworkers? Can you write and test code? And are you friendly enough to chat with every day?

At the end of each interview, there will usually be a few minutes for you to ask questions. Have a few ready! The interviewers can tell you what it is like to work here, what we love, what we hate, how often we travel (or don't!), and are open to answering pretty much anything you remember to ask. (With one exception: We can't tell you how you did.) This is a great time to make sure that Google is the right place for you.

A typical 45-minute interview will consist of 35 minutes of programming problems, five minutes of questions, which leaves only five minutes for everything else, including an introduction, discussing your resume, and asking questions about your prior work experience. That said, your resume and work experience do matter; that and any references are what get you into an interview, so please make sure to polish your resume. Every interviewer you will meet will have been given your resume several days in advance, which is part of what helps them choose their questions.

Each interviewer has a limited amount of time to convince themselves that you will be a great hire, and they want to spend that time in the most efficient way. Therefore once you are in a technical interview, our interviewers will mostly focus on programming problems, not the resume, which we find to be the best use of your time.

Although it's unlikely to be the focus of an interview slot, be prepared to discuss what is on your resume. You should be able to talk about your experiences (especially the technical bits), explain your areas of focus and why they interest you, and be able to describe your contributions to the projects you list. The typical interview questions also apply: Why do you want to work for Google, and which types of projects are the most interesting to you?

A question that comes up time and again is: What should I wear? For Google, the advice I hear repeatedly, and seems to hold true, is: "Wear something that makes you feel comfortable." The specifics are up to you. That said, I feel it is still worth spending a bit of time on grooming. This will be your first time meeting meeting people you may work with for years, and making a decent impression helps convince the interviewer that they want to work with you.

In summary: Refresh your knowledge of data structures, algorithms, and writing clean code on a whiteboard. Come to the interview well rested, and feel free to ask the recruiter questions ahead of time. Be able to talk about your experience, and be ready to spend most of your time on the programming problems. Once in the interview, feel free to ask questions about the problem you are working on. During the interview, make sure to "talk out loud" enough. When you make decisions on how to solve something, make sure the interviewer knows about it. And be sure to ask questions that will help you find out if Google is a good fit for you.

Finally, be who you are, and be the best version of yourself. Our recruiters liked you, and the odds are our engineers will like you, too. Good luck!

(Parts of this have appeared in "Anatomy of the Google Interview," a talk given at Google by Carl Evankovich. Without his work there, this article wouldn't have happened; thank you!)

Biography

Dean Jackson's a member of the ACM and an engineer working at Google Pittsburgh, focused on Google Ads, and a frequent contributor to Google's recruiting programs.

Photograph by Ahmad Faizal Yahya