How to Interview Well

The most important thing for you to prove in a technical interview is that you are a **strong problem-solver**.

Your knowledge of programming obviously plays a crucial role in your ability to solve problems, but your ability to articulate your thought process is equally important. Following the below guidelines will help you demonstrate that you can both **think through hard problems** and **effectively communicate your thought process**.

When your interviewer poses a question, be sure to:

Restate the problem

Translate the problem statement to plain English, and give a concrete example that demonstrates the problem. If you're asked to write code for a given task, coming up with an example input and the corresponding output is a good way to do this.

If possible, take this as an opportunity to interact with your interviewer. Questions are often ambiguous, and asking for clarification shows that you're able to think them through.

Sketch your solution

Before writing any "real" code, explain the high-level steps you need to take to solve the problem. Think of this as pseudocoding, and explain each step to your interviewer.

Discuss your approach

Take a moment to explain why you approached the problem as you did. If your solution isn't as efficient as possible, this is a good time to point that out. Don't get hung up in producing the best solution at this stage—focus on producing a solution that you can improve later.

Implement your solution

This is where you write your code. Don't use psuedocode. Instead, write syntactically correct Python.

Be sure to use expressive, well-considered names for your variables and functions. Just because you're coding doesn't mean you can stop communicating. Explain what you're doing each step of the way.

If and when you run into issues, approach solving it the same way you approached the original problem—explain it, sketch a solution, etc.

Discuss shortcomings and improvements

When you're done, take a moment to reflect on your solution and ways to improve it.

Can you think of any ways to make it more efficient, more readable, or more robust? State them here.

If you’re familiar with time complexity, now is a good time to mention it.
