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 JavaScript 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—write syntactically correct JavaScript.
  - 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.

As an example, consider the following question

- Given an array of numbers, write a function that outputs a version of the array with its contents in descending order.

- **Restate the problem**

    - **Explain in plain English with an example**: "If I'm given an array of numbers, my function needs to return an array with those numbers, sorted from largest to smallest. So, if I'm given an array like `[1, -2, 4, -5]`, my function has to return `[4, 1, -2, -5]`."
        - **Ask for clarification**: "Can I assume that the array of numbers I'm given is sorted?"

- **Sketch your solution**
    - **Sketch an approach**: "If the array I'm given is *un*sorted, I would first sort it and then reverse it."
    - **Explain the steps**: "Sorting it would give me an array whose contents are in *ascending* order, and reversing an array whose contents are in ascending order gives me an array whose contents are in *descending* order, which solves the problem."

- **Discuss your solution**
    - "I'm sorting the array and then reversing it because that solves the problem for *any* array, whether sorted or unsorted."
    - "It would be faster to just reverse it, but that only works if I can assume the input is already sorted."

- "Also, while you said I'll be given an array of numbers to sort, in real life it would make sense to check that the array really does contain only numbers."
- **Implement your solution**
    - " First, I'll create a function named `sortAndReverse`, with a parameter I'll call `array`. Since all I need to do is sort and reverse `array`, I'll just `return`

`array.sort().reverse()`."

- "It looks like I get the correct answer if I run `sortAndReverse` on the example input I came up with earlier, but the wrong answer if I run it on `[1, 2, 4, -5, 100]`."
- "If I break down my solution into two steps—first, `varsorted = array.sort()`, and then, `varreversed = sorted.reverse()`—I see that `array.sort()` isn't doing what I expect it to."
- "According to the documentation for `Array.prototype.sort`, I have to pass `sort` a function that compares each element in the array."
- *Etc.*—use the same approach to solve the problem of writing your `compare` function.

- **Discuss shortcomings and improvements**
  - "If I were using this in production, I would want to make sure that `array` only contains numbers."
  - "Since `sort` changes the input array, It might be better to use a different approach to avoid side effects, unless we explicitly want `sortAndReverse` to mutate the input array."
