Lesson | Wednesday

Intermediate JavaScript (/intermediate-javascript) / Object-Oriented JavaScript (/intermediate-javascript/object-oriented-javascript) / Introduction to Whiteboarding

Text

Many jobs have a multi-step interview process that includes a technical interview where you'll **whiteboard** a solution to a coding problem. **Whiteboarding** is the process of solving a coding problem on a dry erase whiteboard. It can be a very stressful process, even for experienced developers, and the practice has fallen out of favor in some tech companies. Proponents of whiteboarding argue that it's a quick way to get a sense of someone's actual coding ability. However, there are a number of issues with whiteboarding as well:

- Whiteboarding doesn't accurately simulate a real-world environment where we have documentation, a code editor, and other resources at our disposal.
- Whiteboarding can disadvantage people from underrepresented groups that might already have to contend with going through an interview process at a company that likely is already lacking in diversity.

As a result, some companies don't do whiteboarding technical interviews. Instead, they might do an interview about technical concepts (with no whiteboarding involved) or you might get a take-

home test where you have a certain amount of time to solve the problem on your own. In the latter situation, you'll often have a follow-up interview where you'll be asked to walk through your code.

However, whiteboarding is still a common part of the interview process — and the best way to get better at whiteboarding is to learn how to do it and then practice until you're comfortable with it.

We provide whiteboards in person at Epicodus. For students working remotely, you may want to get a personal whiteboard to practice on. At Epicodus, students often use 9" x 12" whiteboards to practice, so if you want to get one for personal use, this size is available at a low price point online.

The act of whiteboarding is ***very*** different from writing code in a text editor. That's why we recommend that you practice on an actual whiteboard — though it's also fine to practice with pen and paper, a blackboard, or for that matter, even with chalk on a sidewalk or a dry-erase marker on a bathroom mirror. The point is to get away from your computer and to actually *write* out your code without referring to any documentation.

In this lesson, we'll walk through the process of how to whiteboard. Even if you can't solve the problem you're given, you can use this process to solve as much of the problem as you can.

We'll use a very simple problem as an example. Let's say an interviewer asks us to write a function that adds two numbers together. What follows is the process we'd go through to whiteboard this problem. This same process will apply regardless of the difficulty of the question. In fact, you'll never get a whiteboard problem as simple as the one we are using as an example. However, we want to focus on the *process* itself in this lesson, not the problem to be solved.

Whiteboard Interview Steps

Our interviewer has asked us a whiteboard question: write a function that adds two numbers together. However, we won't start coding yet. The first part of a whiteboard interview is to talk through the problem.

Talking Through the Problem

1. Ask clarifying questions.

First, we need to ask clarifying questions. Your interviewer may intentionally be vague about something or leave out information. Don't be shy about asking for clarification or more information.

Also, when asking clarifying questions we should be considering **edge cases**, which are extreme use cases. When you consider extreme use cases, you are considering what happens when we pass in an unexpected input to our function.

Here are some questions that we could ask about a function that adds two numbers together:

- Can we assume that both arguments are numbers or could they be strings or another datatype?
- Should the function be able to add floating point numbers (decimals)?
- How should the function respond if `NaN` is passed in as a number?
- What should the function return?

There's nothing wrong with being thorough here. For example, the fourth question above probably seems obvious — this function should just return the sum! But it's often not obvious in an interview. And what if we'd missed a small but crucial piece of information where the interviewer asked us to sum the numbers *and then* return whether the sum is positive or negative? By asking clarifying questions, we are actively communicating and thinking about the problem.

2. Talk through the most basic solution.

We're still not ready to start coding yet. At this point, we can talk through what our function will do. This should be the most basic implementation. We might say something like this:

> "I'm going to write a function named 'add' that will take two parameters. Since you mentioned that we can't assume that the arguments passed into our parameters will be numbers, we will use 'parseInt' to convert them to numbers if possible. If they can't be converted to numbers, our function will return a string that says 'please input a number'. Otherwise, if the arguments are numbers or are strings that can be converted into numbers, our function will add them together and then return the sum."

Here we've talked through the problem and how we'll solve it. Even if we have issues writing out the code, we already have a good start — and verbalizing the problem should help us write it down, too. Note that we added a wrinkle to our problem — our function might be accepting values that aren't numbers. That doesn't just affect the function itself — that affects what it returns. That means the clarifying questions we asked in step #1 were necessary to help solve our problem.

3. Optionally, break your function down into a series of steps that you list on the whiteboard.

You could write out these steps in pseudocode or just plain English. Doing this can be helpful if you are feeling particularly nervous, because it can give you a point of reference to refer back to if your mind goes blank or you freeze.

Otherwise, it's particularly helpful to do this if the problem you are solving is particularly challenging and complex. Listing out the functionality that your function must have in steps can help you brainstorm in a structured way before you begin to actually code the function.

4. Talk through a better solution if possible.

If we are interviewing for more advanced engineering roles, we'll likely be asked to solve a problem that has both inefficient and efficient answers. That's beyond where we are currently at right now! However, we can always look for opportunities to make our code better. Here's an example for this problem:

> "The solution I've already suggested is the most basic implementation but we can improve it further. A function for adding numbers shouldn't really be returning a string. So let's have it return `NaN` instead if something isn't a number — and we can always have another function that returns a user message if needed. Alternatively, we could have a separate function that checks whether the parameters are numbers first before passing them into our add function. Would you prefer one or the other or should we just stick with our most basic solution?"

Even with this very basic example, we found a way to think about improving our code. It's a best practice to keep our functions simple and focused on one thing. That makes our code more modular and easier to scale up and refactor. This best practice is called **separation of concerns**.

If possible, we should always talk through a better implementation. Even if we are only able to code the basic implementation, the interviewer knows that we have ideas for improving our code.

Now we're ready to actually start coding!

Whiteboarding the Problem

1. Write example inputs and outputs.

Listing inputs and outputs will state the data types that should go into and be return from the function. Later, you can use these inputs and outputs to manually test the function that you wrote. Make sure to list inputs and outputs for every case that your function should handle. We could start with something like this:

```
input: 7 and -5
output: 2
```

We might say: "When I pass 7 and -5 into the function, I expect them to be added together and return 2."

We could be more thorough:

```
input: 3, -8
output: -5

input: "7", 5
output: 12

input: "wow!", 0
output: NaN
```

Then we might say:

> "Here are a few different arguments the function should take. In the first example, our function should return -5. In the second, it should parse the string 7 and return a sum of 12. In the final example, we can't parse 'wow!' so we'll want our function to return NaN."

Being more thorough in this process is helpful, because it can help you identify more cases that your function should handle. Think of this process as a structured brainstorming session that can help you prepare for actually writing the function.

2. Use space wisely and write clearly.

This is more challenging when working with small whiteboards. However, you'll likely be working with a full-sized whiteboard in an interview. Even so, it's easy to run out of space! You should do the following when writing out your code:

- **Start at the top left corner of the whiteboard.** That way you'll have plenty of space to work with. Too often, we see students starting in the middle of the whiteboard — and then they run out of space.
- **Give yourself ample space between each line.** Think of this as making each line double-spaced — with about the same height of whitespace as the code you are writing. This is because you may well need to go back and add additional code as you refactor the problem — or even if you realize you've forgotten to write a line of code or have made a mistake. This way, you'll have the space to add that code in.
- **Write clearly.** This is hard for many people — especially since we spend so much more time typing than handwriting these days. Make sure your handwriting is clear and not too small. If you think it might be difficult for your interviewer to read the code, that's not a good sign. How will they be able to tell if you've written good code or not?
- **Use descriptive variable names, and proper syntax and indentation.** This demonstrates your attention to detail and it is an absolute must.

3. Verbalize your process.

This is very challenging at first, but you should be talking through the problem *while* you are solving it. **More than anything your interviewer wants to understand your problem solving process.** Verbalizing what's happening can feel a bit like patting your head and rubbing your tummy at the same time — but it does get easier with practice. So we might be writing something like this:

```
function add(num1, num2) {

}
```

Meanwhile, we might be saying this:

> "We want our function and parameter names to be clear and concise. Add seems like a good name, and num1 and num2 are concise but clearly state that the function takes two numbers. And our function needs to take two arguments because it's adding two numbers together."

4. Test your solution.

Once you are done, walk through your solution by passing in the example inputs you listed earlier in this process. You should describe each line of code as the input passes through the function. Make sure to work through all of the example inputs you previously listed to ensure that you've tested for every case that your function should handle.

The goal of this process is to verify that your solution is correct. It gives you an opportunity to catch any issues before your interviewer does, and in the process, you may also find ways to refactor your code. If your input doesn't arrive at the expected output, well that's a sure sign that you need to fix something in your code.

5. Admit when you don’t know something.

If you don't know how to solve part of the problem, admit it. Don't try to fake your way through it — your interviewer knows how to solve this problem and different approaches. It's okay to ask questions as well. Interviewers *will* collaborate with you and will give hints as needed. Getting a hint or two doesn't mean you won't get the job. Of course, if the interviewer needs to write out most of the solution, that wouldn't be a good sign.

If you know the name of a method or tool you want to use in your code, but you are not 100% sure if you are using it correctly, or remember the exact syntax, that's okay. Go ahead and use the tool, but make sure to communicate to the interviewer about what you're unsure of and how you are intending to use the tool in your code.

Sample Whiteboarding Prompts

The next step is to practice. You will get many chances to practice with pairs and groups at Epicodus but we also recommend practicing on your own. When you do, make sure you follow the steps above even when you're by yourself. This includes writing out the problem by hand and verbalizing your process every step of the way. The point is to write the code by hand and explain what you're doing *while* you're doing it.

Here are some sample challenges to work on from CoderByte (https://coderbyte.com/). You can also find more challenging prompts in Project Euler's archives (https://projecteuler.net/archives).

- Using JavaScript, create a function called `flip(str)` that takes the `str` parameter being passed and returns the string in reversed order.
- Using JavaScript, create a function called `factorial(num)` that takes the `num` parameter being passed and returns the factorial of it (i.e.: if `num = 4`, return(4 * 3 * 2 * 1)). For the test cases, the range will be between 1 and 18.
- Using JavaScript, create a function called `cipher(str)` that takes the `str` parameter being passed and modifies it using the following algorithm.
    - Replace every letter in the string with the letter following it in the alphabet (ie. c becomes d, z becomes a).
    - Then, capitalize every vowel in this new string (a, e, i, o, u) and finally return this modified string.

Recommended Resources

- Check out this video (https://youtu.be/XKu_SEDAykw) of two software engineers at Google performing a mock interview. Notice how the interviewee asks questions and makes a plan

before starting to write. Notice how he is *always* communicating with the interviewer. At the 21:05 minute mark, the interview also recaps important parts of the process.

- The articles Rock Your Next Whiteboard Test (https://skillcrush.com/blog/rock-your-next-whiteboard-test/) by Debbie Chew and How To Pass a Programming Interview (https://triplebyte.com/blog/how-to-pass-a-programming-interview) by Ammon Bartram are excellent resources to learn more about whiteboarding.

Previous (/intermediate-javascript/object-oriented-javascript/game-of-choice-two-day-project)
Next (/intermediate-javascript/object-oriented-javascript/switch-cases)

Lesson 27 of 33
Last updated March 23, 2023

disable dark mode

(http://www.epicodus.com)