Mental Debugging

Mental Debugging is the process of taking your practice efforts, so when you're actually practicing problems, and using them to identify conceptual errors. So, not just trying to memorize a solution pattern so you can figure out how to solve the question the next time, but figuring out why you didn't understand it correctly, why you didn't understand the problem correctly in order to solve it better in the future and for an entire class of problems, not just a specific solution pattern.

So, there are a few types of errors you can get when you're working on practice problems. Sometimes, they are just a typo, you've just screwed something up, it was very basic, and you do understand the underlying concept. Other times, it was because you forgot a piece of information, so you understand it when you look at the answer, but when you were doing the question, you had forgotten that little element of it. And again, that's a basis where creating more metaphors can help, because it really sears into your memory those vivid connections.

But, there's a third type of error, where you felt that you understood the idea and when you look at the answer, you're actually surprised. There's a little bit of a "Hmm" moment, "That's funny." about what you're learning because your mental model was actually incorrect. So, with these third type of problems, Mental Debugging is very important, because most students, when they don't understand the question and they didn't get it right, they will look to the answer and try to memorize the pattern, the solution pattern, for solving those types of questions. But, this doesn't really generate a deep understanding. Solution patterns can be good for doing the questions faster, but if you debug a conceptual error, it'll help you with a broad range of questions, not just one particular pattern.

So, an example of this when I was doing this is, I was working on programming and I was working on a programming problem, and this was again, partially to my whole lack of education, but I had this model of how arrays worked. So, an array in computer science is a, sort of a line of elements in a row, in a block. And so, if you want to access it, you say which cell you want to access. So, if there are a hundred elements, then you say I want to access the information that's in Cell 88. So, this is how an array works. It's a very basic computer science concept.

Now, when I started programming with computers, I had a bug in my mental model. And, my mental model was that there were sharp boundaries at the ends of each of these arrays. So, if I had an array of a hundred elements, and I accidentally tried to access the 111^{th} element, it would give me nothing because there's nothing past that point.

Now, when I actually was running my program, when I was trying to do this practice problem, I found out that mental model was incorrect, that I was actually getting numbers and they were sort of random. They weren't based on anything that I was expecting them to be and it was screwing up my program. What my mental model was that I could have just gone there and figured out, "Well, this is

the correct programming practice, is to never array elements outside of their bounds." That's a very logical thing. That's what the other programmers told me that I should be doing, that you shouldn't be accessing it that way.

But, in the certain sense, just learning the best practice is like solving a solution pattern. It gives you an answer to a particular problem, but it doesn't help you understand why your answer was wrong, or why the correct answer works the way it does. And, when I did a little bit more research and I did a little bit more understanding, I was able to understand that the reason is because the array exists side-by-side next to lots of elements and there are no boundaries. The boundaries are up to you to enforce. And so, when you go outside the elements, you're actually accessing other arrays of data which were stored right next to it, on the computer program hard disk.

So, this is something that you can do and apply to your own learning and your own learning efforts, is that, when you don't understand a problem, the naïve or easy way out, or shortcut is to just try to memorize the solution pattern. But, my challenge to you is to go beyond that and try to figure out why your answer couldn't be right or why it wasn't working the way you thought it was because your understanding was flawed. And then, second, figuring out what the correct understanding is, because if you can debug your mental models, that's far more useful than just memorizing a narrow solution pattern for a specific problem.

So, good luck with this tactic, and I'll see you on the other side.