(/)

White boarding

In real-world projects, your brain should be able to write a quick efficient solution for complicated stuff and you can only do that when you practice a lot of coding questions. Understand that language and frameworks are just tools, they won't teach you problem-solving skills. You develop problem-solving skills when you practice a lot of coding questions.

Every developer has their own tricks and they follow their own pattern to solve coding problems but when it comes to new developers they are always uncertain about where to start. A lot of them understand the problems, the logic, and the basics of syntax, they also understand someone else codes and they can follow along with them but when it comes to solving the questions on their own, they get stuck. They don't understand how to turn their thoughts into code even though they understand the syntax or logic. This article is a result of mass research of how programmers effectively approach a coding problem.

Step 1: Understand and Analyze the Problem

It doesn't matter if you have seen the question in the past or not, read the question several times and understand it completely. Now, think about the question and analyze it carefully. Sometimes we read a few lines and assume the rest of the things on our own but a slight change in your question can change a lot of things in your code so be careful about that. Now take a paper and write down everything. What is given (input) and what you need to find out (output)? While going through the problem you need to ask a few questions yourself…

Did you understand the problem fully? Would you be able to explain this question to someone else? What and how many inputs are required? What would be the output for those inputs Do you need to separate out some modules or parts from the problem? Do you have enough information to solve that question?

Step 2: Go Through The Sample Data And Examples Thoroughly

When you try to understand the problem take some sample inputs and try to analyze the output. The sample inputs will help you to understand the problem in a better way. You will also get clarity that how many cases your code can handle and what all can be the possible output or output range. Consider some simple inputs or data and analyze the output. Consider some complex and bigger input and identify what will be the output and how many cases you need to take for the problem. Consider the edge cases as well. Analyze what would be the output if there is no input or if you give some invalid input. Step 3: Break Down The Problem When you see a coding question that is complex or big, instead of being afraid and getting confused that how to solve that question, break down the problem into smaller chunks and then try to solve each part of the problem. Below are some steps you should follow in order to solve the complex coding questions…

Make a flow chart for the problem at hand.

Divide the problem into sub-problems or smaller chunks. Solve the subproblems. Make independent functions for each subproblem. Connect the solutions of each subproblem by calling them in the required order, or as necessary. Wherever it's required use classes and objects while handling questions (for real-world problems like management systems, etc.)

Step 4: Write Pseudocode, make a flowchart

Before you jump into the solution it's always good to write pseudocode for your problem. Basically, pseudocode defines the structure of your code and it will help you to write every line of code that you need in order to solve the problem. Reading pseudocode gives a clear idea that what your code is supposed to do. A lot of people or experienced programmers skip this step but when you write pseudocode the process of writing the final code becomes easier for you. In the end, you will have to only translate each line of pseudocode into actual code. So write down every step and logic in your pseudocode. Step 5: Replace Pseudocode With Real Code Once you have written the pseudocode it's time to translate this into actual code. Replace each line of your pseudocode into real code in the language you are working on. If you have divided your problem into subproblems then write down the code for each subproblem. While writing the code keep in mind three things…

The point where you started * Where are you right now? * What is your destination (end result)?

Step 6: Simplify and Optimize your Code

Always try to improve your code. Look back, analyze it once again and try to find a better or alternate solution. We have mentioned earlier that you should always try to write the right amount of good code so always look for the alternate solution which is more efficient than the previous one. Writing the correct solution to your problem is not the final thing you should do. Explore the problem completely with all possible solutions and then write down the most efficient or optimized solution for your code. So once you are done with writing the solution for your code below are some questions you should ask yourself.

- Does this code run for every possible input including the edge cases.
- Is there an alternate solution for the same problem?
- Is the code efficient? Can it be more efficient or can the performance be improved?
- How else can you make the code more readable?
- Are there any more extra steps or functions you can take out?
- Is there any repetition in your code? Take it out.