4 Steps to Solve Any (CS) Problem

Adapted from George Polyá's *How To Solve It* by Nathaniel Granor

1. Understand the problem

At the end of this step, you should be able to restate the problem in your own words. Along the way you might need to draw a picture, research unfamiliar terms or syntax, observe a working demo, or ask clarifying questions.

Can you restate the problem in your own words?
Can you draw a picture or diagram to make the problem clearer?
What are you asked to find or solve?
What can you assume to be true when your code is called (pre-conditions)?
What must be true when your code returns (post-conditions)?
What must remain true throughout code execution (invariants)?
Are there any unfamiliar terms or pieces of syntax that you should research?
Do you need to ask a question to get started?

2. Make a plan

You can probably solve small problems in your head without thinking too much about them. But when it comes to bigger or harder problems, planning is the most important step! Your plan might be on paper or in a notebook. A complete plan usually takes the form of pseudocode or a diagram. Figuring out a correct plan sometimes seems like magic: you need that "special flash of insight" to see the way to a solution. But those flashes of insight come about by methodically applying questions and strategies like the ones below.

Can you find a special case of this problem that you know how to solve?
Can you find and solve a mathematical equation that represents your problem?
Can you find a pattern in the input, output, or code you need to write?
Can you work backwards?
Can you work around the problem by creating a "stub" function or method that you implement later?
Can you think of an analogous problem that you know how to solve, or that others have solved?
Can you think of a related problem that has already been solved?
Can you find a more general problem that you can solve?
Can you derive a solution to your general problem by looking at a few specific cases?
Can you change/vary the problem to create a new problem whose solution would help you with the original problem?
Can you break the problem into smaller sub-problems and solve/recombine those problems?
Would the use of a well-known data structure help you solve the problem?
What *won't* work?
Is there a CS concept that would help you solve this problem?
Is there a library or method that would help?

What variables do I need?
Do I need any IF statements or LOOPs?
What data-types should I use?
Should I build an object?
Should my methods be public or private? Static?
What if I inherit behaviors from a superclass?
Did you use all the data? Did you use the whole condition? Have you taken into account all essential notions involved in the problem?

3. Implement the solution

Once you have a correct plan, you can build the solution. Often you'll start building the solution only to learn that your plan wasn't complete or correct. That's OK, problem-solving is an iterative process. Sometimes you'll run into translation issues: trying to figure out how to do in code what you wrote in words; other times, you'll run into problems with your plan.

Did you consider boundary cases?
Did you cover every possible code-path?
Are you using the correct conditions in conditional statements?
Are you using the right conditions/bounds in loops?
Do recursive functions have a base case?
Have you captured/respected the pre- and post-conditions?
Are your variables using the correct data-type?
Will your solution run in an acceptable amount of time/with acceptable memory use?
Have you reinvented a wheel? (Is there something already implemented that can replace part of your plan?)
Did you implement all of the steps of your plan?
Does each piece of code actually do what your plan says it should do?
Can you explain what each line of code you've written does?
Did you include code comments to relate your code to your plan?

4. Reflect on your work

Even after you've finished solving the problem and verifying your work, there's an opportunity to learn more and prepare for the future by thinking about the solution you built and the process you used to get there.

Can you check the result?
Can you explain why it's correct?
Can you think of another way to solve this problem?
What other problems could you solve with this code?
What would you do the same/differently next time?