# The Framework

## 0. Read

Read everything pertinent to the question / project at hand.

*Example:* Read the project prerequisites, man pages, documentation, books, and articles.

► Try all the examples in the docs.

## 1. Think

Pause, take a breath, and think of the solution. Make sure you understand all of the required concepts and algorithms before you proceed to solve tasks.

Once you have found your algorithm or solution - and only then - can you start to code.

**Repeat:** Do NOT write any piece of code before you have the solution. Coding should only be translating your solution to computer code.

To think of your solution, try to break things down; a big problem is easier to solve when you break it into smaller, easier problems.

*Example:*

First, rephrase the problem. “What am I trying to accomplish here?”

Then, go through different examples: “if I understand correctly, then with this input/context/parameters I should output/print/return/draw/… this”… “What could be the edge cases I have to think about to make sure my function/program will always work correctly and never crash?”

Next, focus on the basic/regular use cases. “What should my algorithm be?”

When you found it, try to “run” it through different examples on a whiteboard or pen/paper and see if the input/output match your expectations.

Finally, once you get that, try to see what changes you have to make to take care of all possible edge cases.

► Sometimes, you will be able to “guess” solutions or complete tasks without totally understanding all the concepts, especially if a task is very small. Do NOT do this, because if you do not develop your critical thinking

skills with smaller learning opportunities you will not have the ability to solve harder questions. You need to start thinking like an engineer, and this is how to begin.

► Use a whiteboard or pen/paper until you are 100% sure your code will work; this maintains focus, and keeps your mind active.

# 2. Think more + Whiteboard

Write or draw out your problem on a whiteboard or paper. It's easier to visualize your problem when it isn't pure code.

Can't find the solution? Patiently think more, it's out there. Also, ensure you read the resources thoroughly.

# 3. Ask why?

Whether or not your code runs when you input it, make sure you know why it doesn't (or does) work without looking at the error message.

You must also understand the vocabulary, concepts, and tools you use, and question why they exist so you ultimately know.

*Example:* Let's imagine we're doing the bit manipulation project.

Why do you think this is important? Is it important? Why binary? Where do bits play a role? How is it connected to the machine, memory, bytes and values of types?

If you haven't asked and answered those questions, then you did the task wrong.

Also, if someone comes to you with this question and you don't have the answer, stop everything and work on answering this question right away.

This is your future, so ensure you have a strong foundation of understanding.

► Never blindly work on tasks without first understanding why it is important to spend time on the topic of the day and how it plays a role in the system or comprehension of CS. Understanding the big picture is very important to your education and future career.

# 4. Read the error messages

Still can't understand why it doesn't work? Most of the time, the answer is in the error message.

# 5. Google

If you still can't find the solution or understand why your code doesn't work, google it *

*error message, specific question, specific concept, etc.

► **NEVER TRUST THE INTERNET**. A lot of the content on the Internet is wrong or incomplete. Always make sure what you are reading does make sense and is correct by challenging and testing every assumption.

► You should only google something specific. This step is **NOT** about looking at how to do the project. It should only help answer a specific question or understand a concept.

► You are **NOT** allowed to look at the solution / other people's code. Even for "analysis". Looking at other people's code / a solution is considered cheating.

► **NEVER** copy and paste something from the Internet. You need to understand everything before you implement something you read. Typing yourself - instead of cp/paste - ensures that you better understand and not miss any details.

# 6. Google again

If you can't find it, then Google again using more context in your search or by rephrasing your question.

# 7. Ask a peer

If you are still stuck on a problem despite doing the steps above, don't wait too long before asking to a peer.

It is part of your job to ask for help. You are expected to do so, just like you will be expected to do so when you will have a job.

> Repeat: It's your job to ask and it's your job to help - especially at ALX SE.

**7.0 If you ask for help**

Respect the time of others.

Make sure you spent enough time going through previous steps.

Explain what the problem is and what you tried (i.e. how you followed The Framework). Give as much context as possible; this helps your peers help you, and will prepare you to be an effective software engineer.

**Poorly asked technical question**

"Hey, my code isn't working. Can someone help me fix it?"

This question is vague and doesn't provide any context. It's difficult for anyone to understand what the problem is or how to help.

**Well-asked technical question**

"I'm working on the bit manipulation project, and I'm trying to set the nth bit to 1 in a given number. Here's the code I've written so far:

```
unsigned int set_bit(unsigned int n, unsigned int index)
{
    if (index > 31)
        return (0);

    return (n | (1 << index));
}
```

However, I'm getting incorrect results when I run my tests. I've gone through the ALX framework and tried to solve the problem myself, but I'm stuck. Can someone help me understand what's wrong with my code and how to fix it?"

> This question provides specific details about the problem, including the project name, the code the student has written, and the issue they're facing. It also demonstrates that the student has gone through the ALX framework and attempted to solve the problem themselves before asking for help. This context makes it easier for someone to understand the problem and provide targeted assistance.

#### 7.1 If you are asked for help

Respect every question; there is nothing more demotivating than having someone laughing at your question or being asked "what? you still don't know that?"

Do **NOT** touch others' computers / keyboards. Explain, and let the person who asked for help type and correct their bugs/code.

Do **NOT** give the final answer. You are here to guide, help them find the solution by themselves.

Sometimes, you won't be able to help, but you are still responsible to find someone who can. Stay with the person until it's resolved. Bonus is that you will also learn how to solve the problem!

Always start by confirming they went through the previous steps by asking "How did you follow The Framework?". If you help someone without making sure of this, you are NOT helping them.

When explaining something, always start with whiteboarding the algorithm. Again, coding should come at the end, only to translate the solution to code. When the student understands the solution, then clean the

whiteboard, and ask the student to do it again on their own. They need to go step by step from zero to understanding to showcase that they deeply understand.

Give hints

*Example of hints:* Try rephrasing the question: "So what you are really looking for is…

Point to the right documentation: "You should read the manual of this command…"

Again, coding should come at the end, only to translate the solution to code. When the student understands the solution, then clean the whiteboard, and ask the student to do it again on their own. Repeat until the student can do it.

► It's ok to code it in front of the student and explain the code, BUT then delete the file and ask the student to do it again on their own. Repeat until the student can do it.

► There are no stupid questions!

# 8. Ask more peers

If those you've asked haven't had the answer, ask someone else.

99% of the time, someone knows and has the answer.

*Examples:*
Ask several students - from your cohort first, and then from previous cohorts. Ask specific, clear questions in your cohort Slack channel, topic specific channels, or reach out to students in other locations.

► Different people will have different ways of explaining, since we all communicate differently. If at first you don't understand, continue to ask others and don't be discouraged.

► Once you understand something on the whiteboard, wipe it and do it again on your own, since you only truly understand when you are able to do it on your own.

# 9. Ask ChatGPT

ChatGPT is a large language model trained by OpenAI, designed to engage in natural language conversations and provide intelligent responses. It can provide personalized explanations, feedback, and guidance on programming concepts, best practices, and common errors, as well as answering your questions and helping you troubleshoot issues in real-time.

Before you use ChatGPT, you need to understand the context in which you are right now. You are here to learn. This is a very different context than when you will be working for a company. **You need to focus on learning**, while an employee needs to focus on shipping and be as effective and fast as possible doing so. As a result,

the usage of AI has to be different in the two different contexts. Remember, you are here to LEARN, not to get green checks. Green checks are here to measure your learning. The check assumes that you are following the framework and that you are never cheating.

In that context, here are some examples of what to ask ChatGPT (remember, only AFTER you have tried the above 8 steps):

► Explain concepts: Ask ChatGPT to summarize concepts or explain them to you in a different way or in simpler terms. Try the following prompt for instance: “Explain C pointers to me like I am a 12 year old”

► Ask additional, more detailed questions about concepts or examples. Try “Explain to me what is the difference between an address and a pointer in the C programming language”

► Find a bug: You have been working on a task for hours, and nobody can help you. You can ask ChatGPT to help you find the bug, and get this final green check. This should be done in two phases.

**Phase 1:** Ask ChatGPT to explain to you why there is a bug, without giving you any code to fix it. Once you understand the reason, go back to your code and fix it yourself.

**Phase 2:** You have done Phase 1 and have spent at least 30 minutes on it. You can’t fix it. You can ask ChatGPT to help you with fixing the code. **BUT you should NEVER copy and paste any code that ChatGPT has produced. N.E.V.E.R. (you can do that in a different context, but not in the context of learning).**

You want to take the time to read the explanation and understand the code. And then, you close the window, and you code it again on your own, without going back to ChatGPT. This is the ONLY way to know that you have actually understood: when you can do it on your own, without the help of anyone or anything.

At the end of the project or task, when you get to 100%, we encourage you to ask ChatGPT if they see a way to improve your code. Try “I am a software engineering student and I had to create a function that {describe the function} This is my code {paste your code here>}. Do you see a way to improve it? If so, can you explain how and why?”

# 10. Ask your Technical Mentors ™

Each non-inaugural cohort has a dedicated Technical Mentor. TMs are to offer guidance in situations where your peers couldn’t assist you. They are here (in-person or on Slack) to help.

► Remember to go through the previous 9 steps before asking a TM. The first thing that they will ask you is: “How did you follow The Framework?”, and you need to be able to answer that efficiently.

# 11. Ask other TMs

If your TM is unavailable or was unable to find the solution, you can also ask TMs from other cohorts.

# 12. Ask Staff/Alumni/Mentors

If no one up to this point was able to help, ask someone on staff. We are here to help on-site or via Slack!

If staff cannot help, flesh out your general concerns into specific and actionable questions that you'd not been able to resolve with steps 0-10 of The Framework.

Once you have your list, ask staff for an alum or mentor recommendation of who would be qualified to answer your questions.

Then, ask the alum/mentors for answers/advice to your specific and actionable questions.

# General Tips:

Getting a high score is less important than understanding and being able to explain things.

> Always remember one thing: your goal is not to become great students. Your goal is to become great software engineers, because no one wants to hire students.