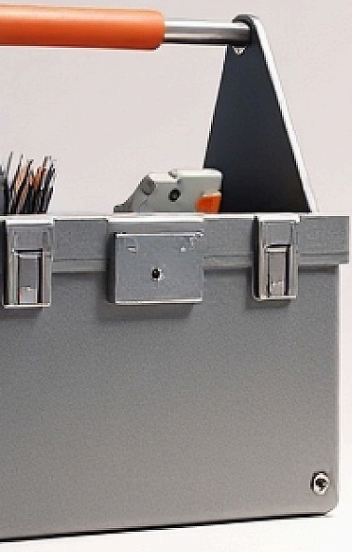
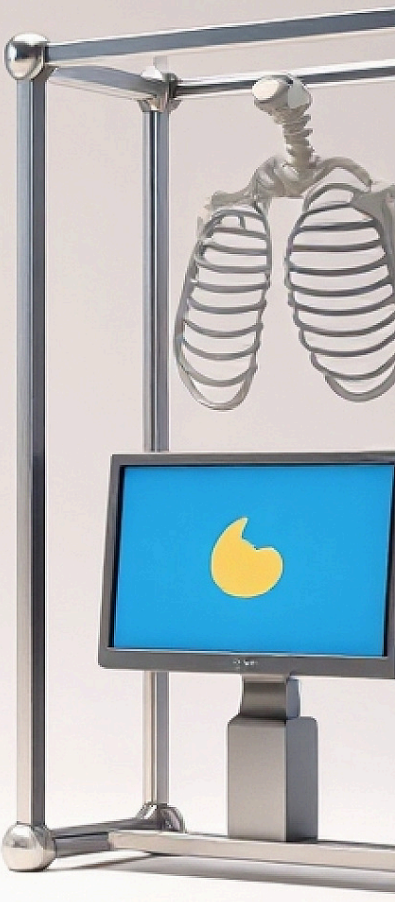
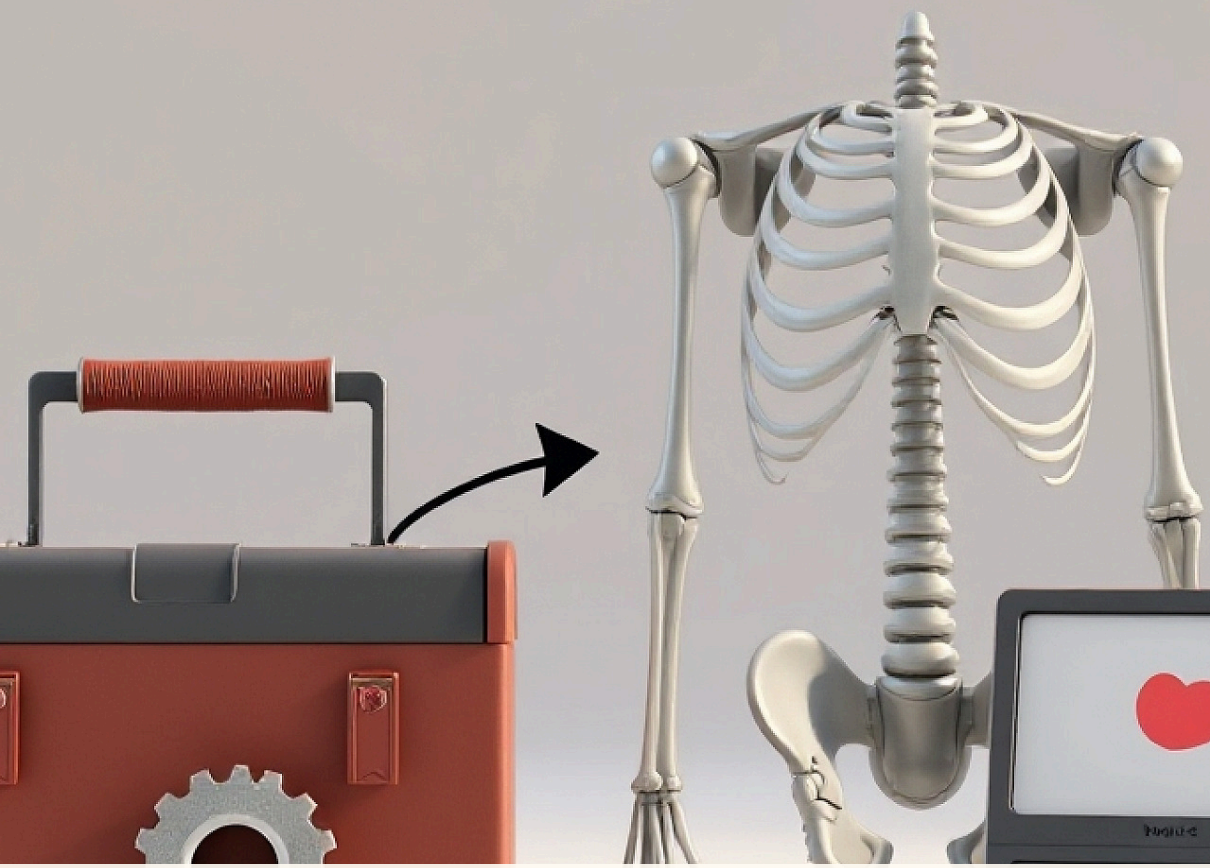
What is the difference between a framework and a library?





www.yzokul.com.tr

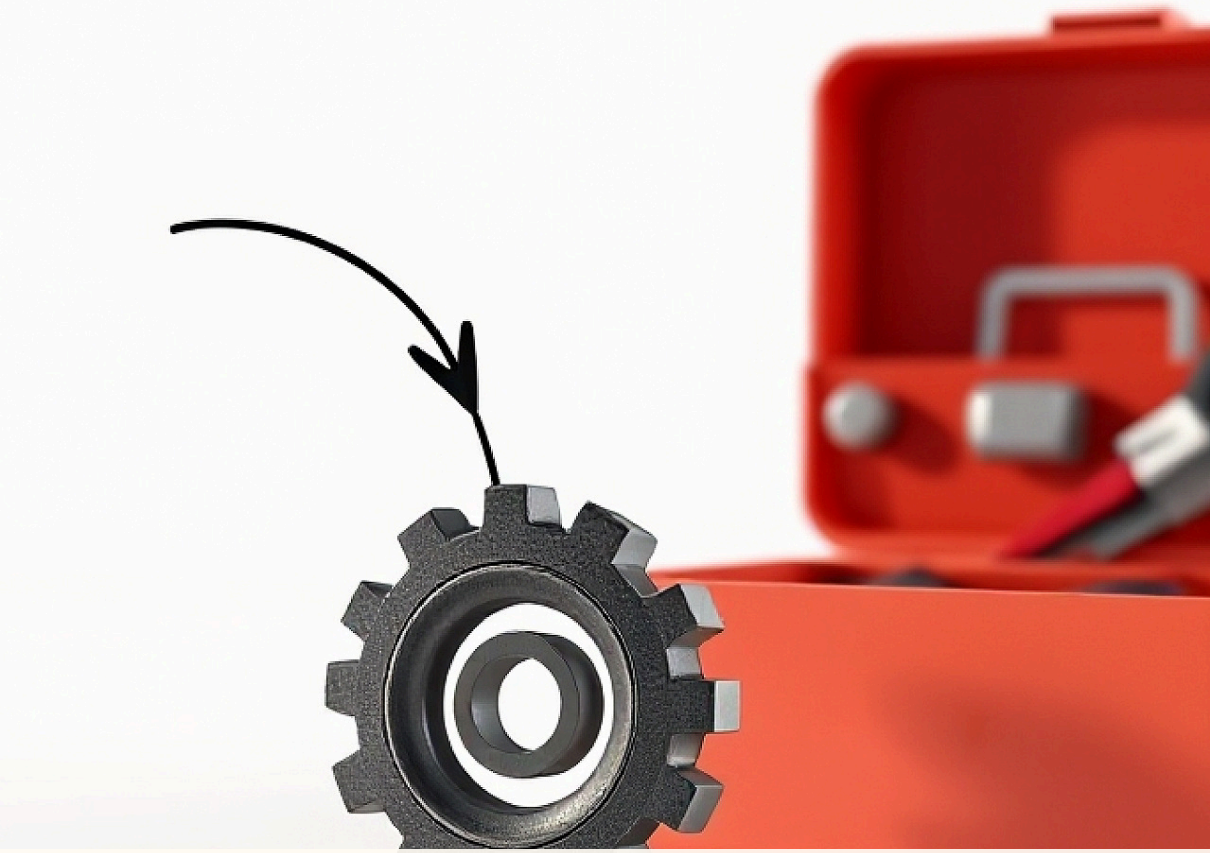
What is the difference between a framework and a library?



You may have heard the words library and framework and wondered how they differ. A short answer is that a library gives you tools you call, while a framework can set your program's structure and control its flow. This book explains that difference step by step, using simple examples and plain language. By the end, you will understand the key idea and be able to tell which is which in many cases.



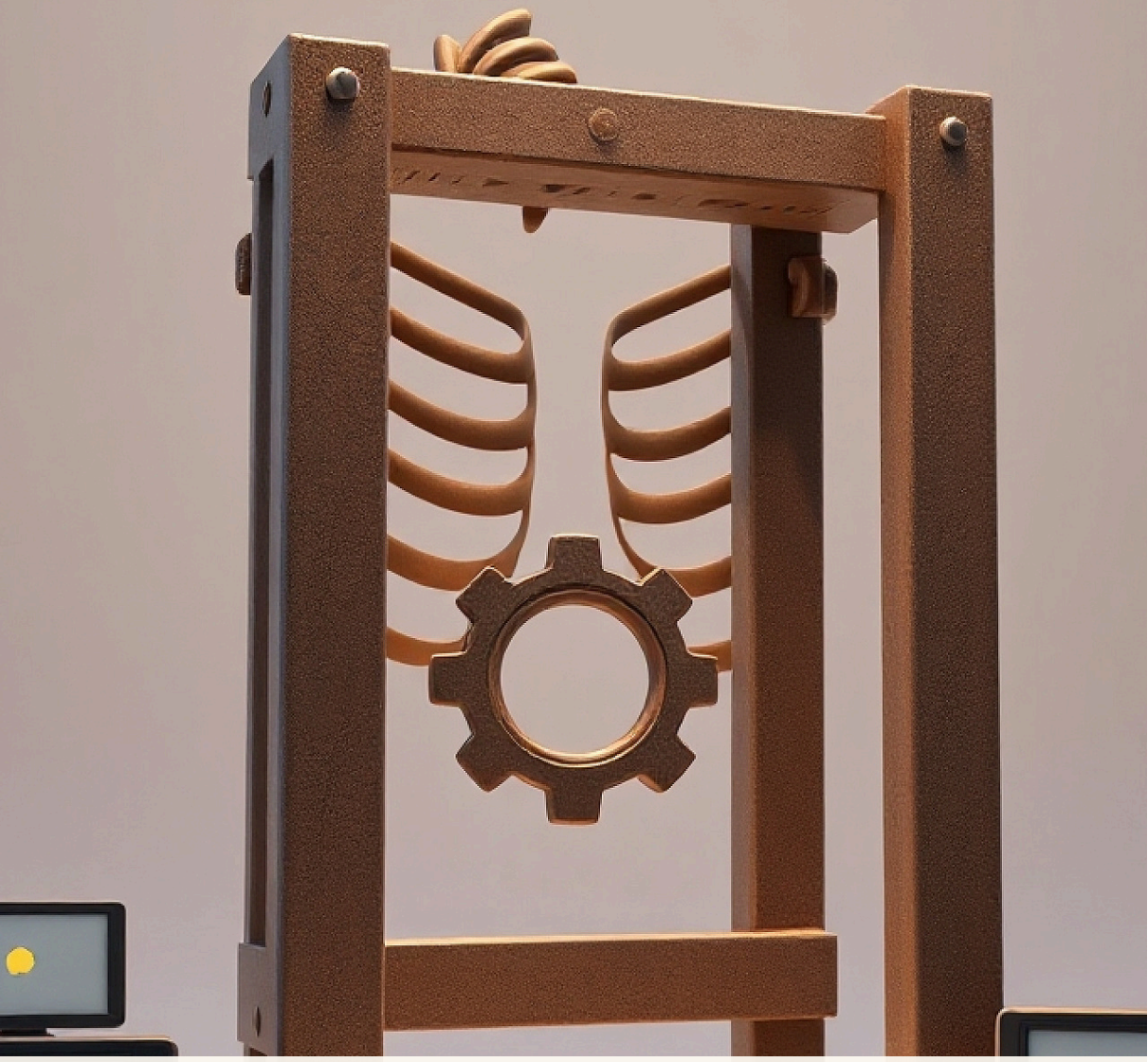
A library is a collection of ready-made tools you can use in your program. Think of it as a toolbox filled with useful functions that someone else wrote and tested. The library sits there waiting and does nothing until your code calls one of its tools. You decide when and how to use each tool, and your program keeps running its own steps. Libraries help you avoid writing common code from scratch.



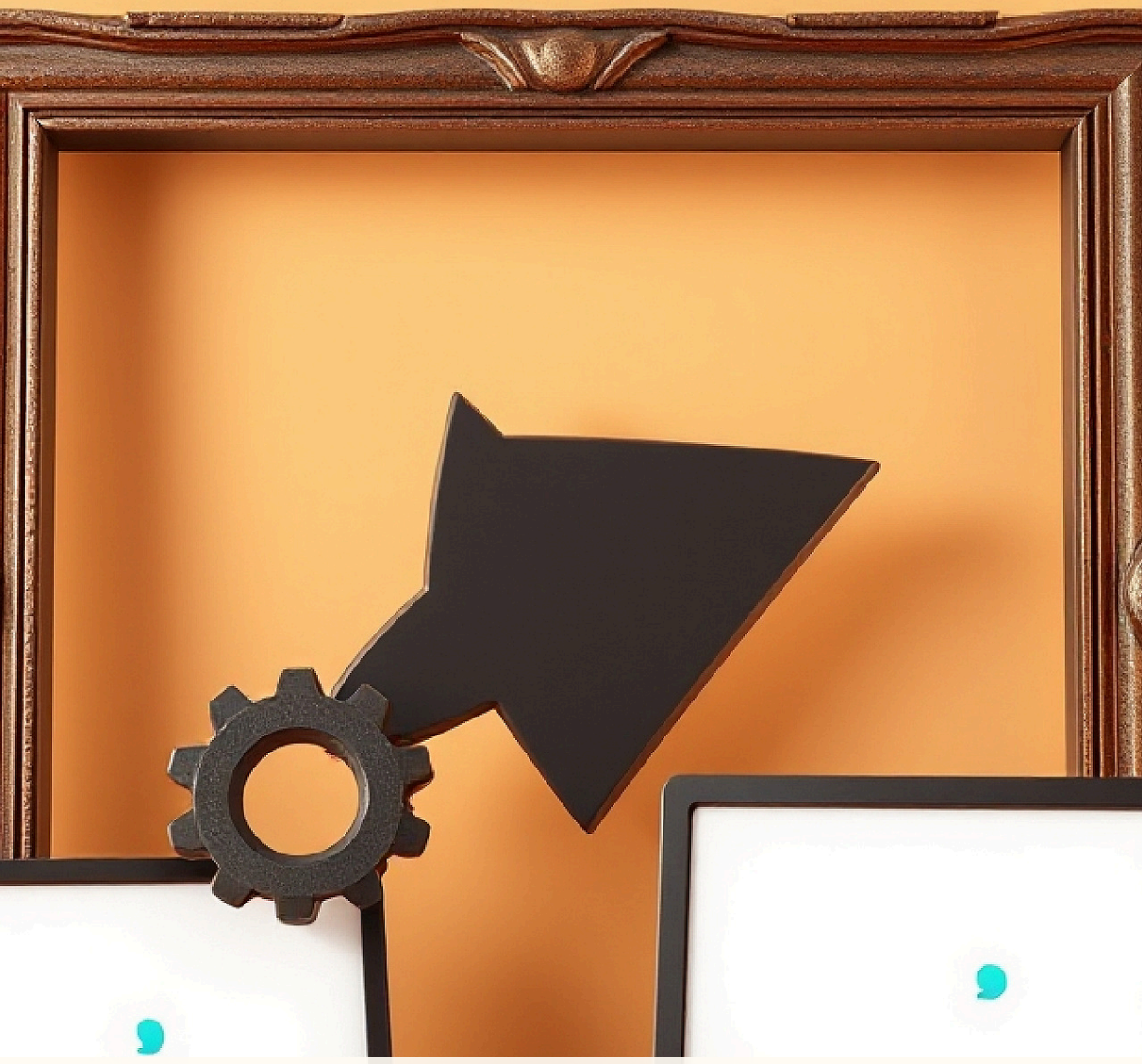
In a library, your program is in charge and decides when to use a tool. Picture your code as the boss that reaches out to the library only when it needs something. The library does not run your program or tell it what to do; it simply responds when called. This means you keep full control of the order of actions in your program. You choose which tools to use and when to use them.



Imagine your program needs to do some math, so it calls a math tool from a library. Your program keeps running its own steps and only borrows the tool for a moment to get a result. After the tool returns the answer, your program continues with its own logic. The library never decides what your program does next; it just provides the calculation. This is how libraries work: they are passive helpers.



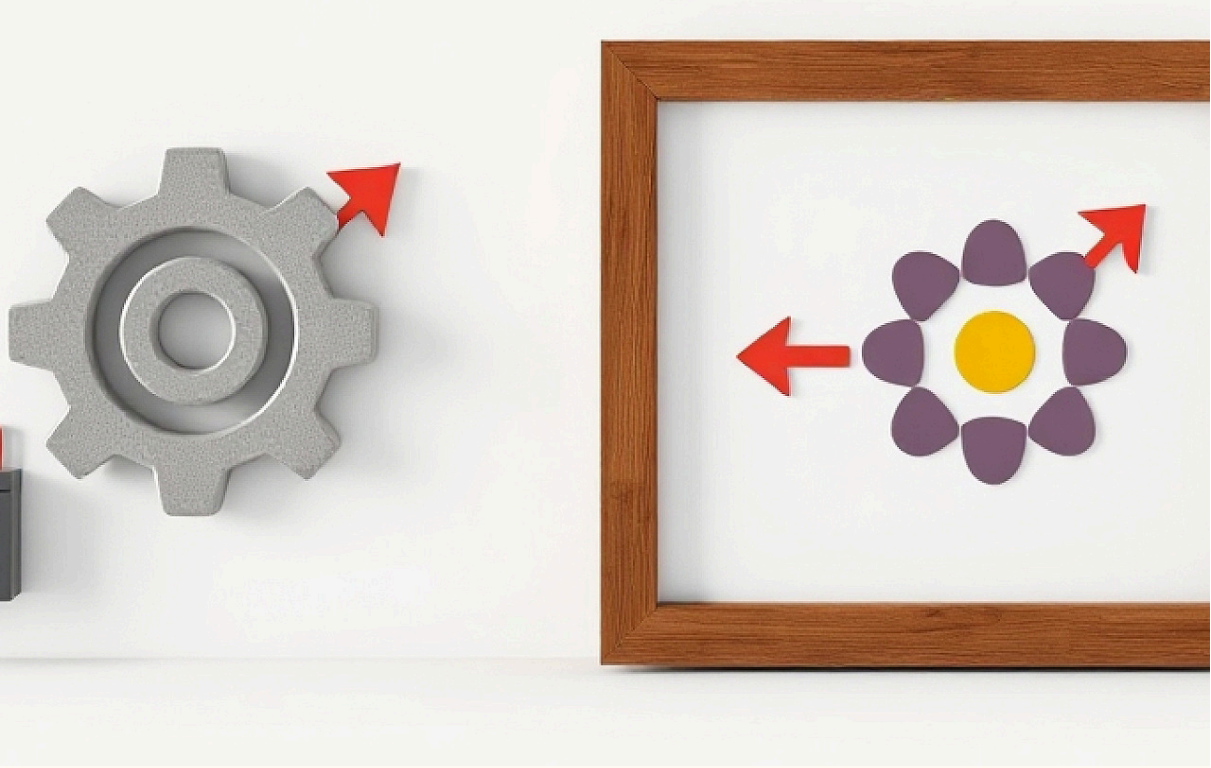
A framework is more than a set of tools: it can provide the general structure of your whole application. It offers a ready-made skeleton that your code fits into. The framework defines where your code goes and how different parts connect. You fill in the details, but the overall shape is already set. This gives you a lot of structure and can speed up development, especially for common types of apps.



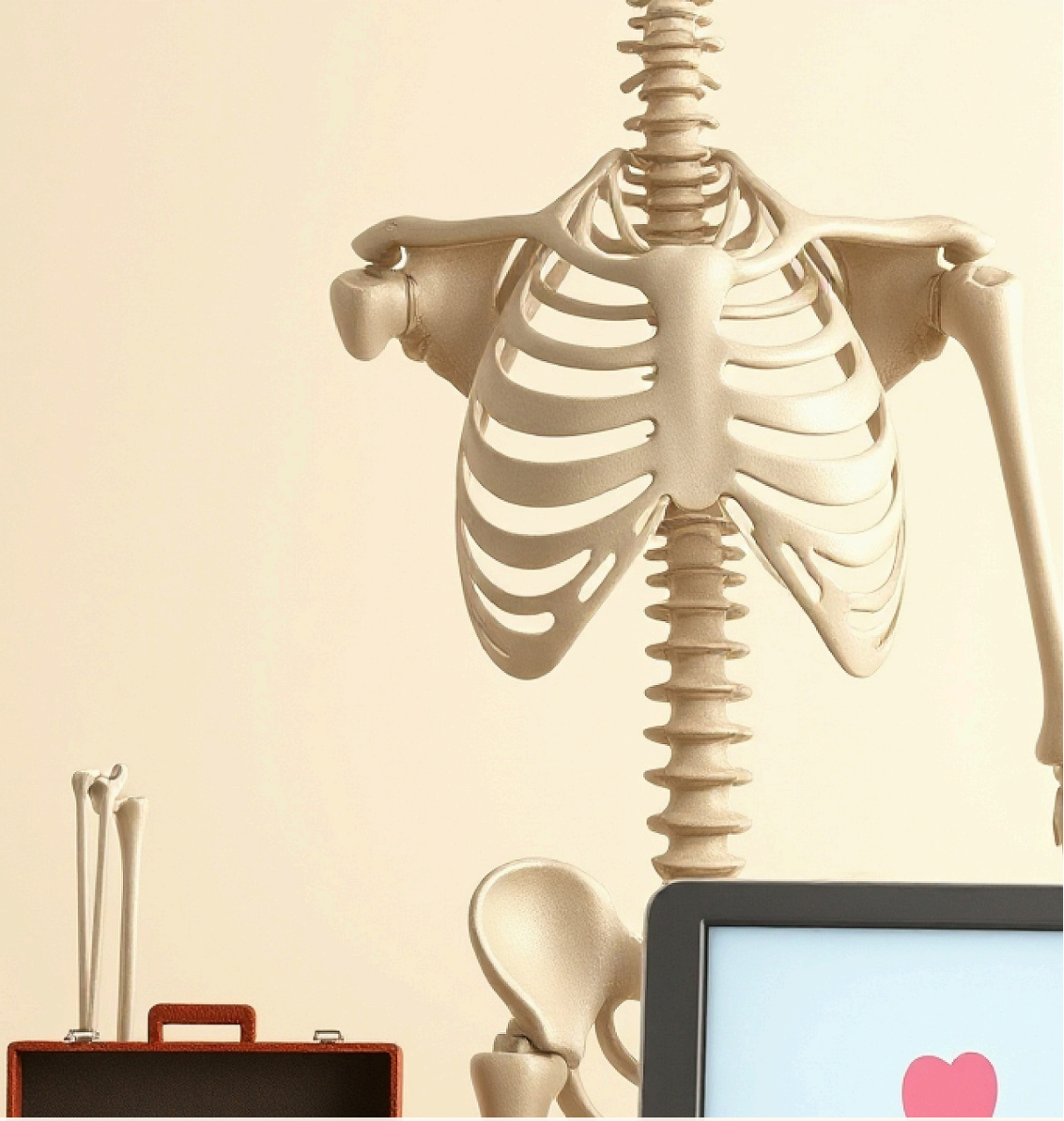
With a framework, the key flip is that the framework is in charge of the flow and calls your code when needed. Your code fills in the parts the framework asks for, such as specific functions that handle certain events. The framework decides when to run those functions and in what order. This is often called inversion of control: you do not call the framework; the framework calls you. Your code becomes a set of pieces that plug into the framework's flow.



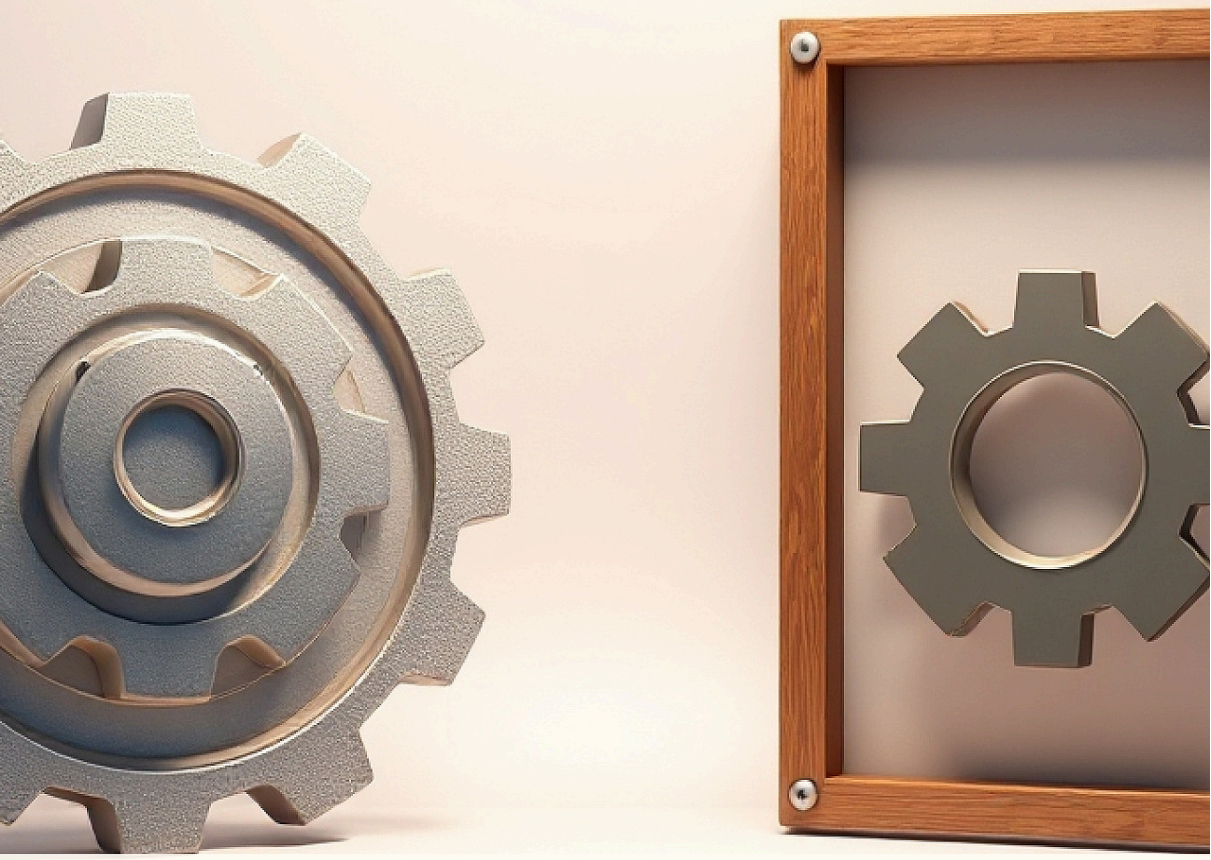
Picture a web framework receiving an incoming request from a browser. The framework looks at the request and decides which of your functions should handle it. Your function runs only because the framework sent the request to it. After your function finishes, the framework may take over again to send a response. You write the handler functions, but the framework controls the overall flow of handling requests.



The main difference is who organizes the flow of the program. In a library, your program organizes the flow: you call the library's tools when you need them. In a framework, the framework organizes the flow: it calls your code at the right times. This is the heart of the whole question. Remember: who is in charge of the sequence of actions? That tells you whether it is more like a library or a framework.



A framework offers much more ready-made structure than a library. A library gives you tools, but you build the overall structure yourself. A framework gives you a whole frame to build inside, with predefined places for your code and predefined ways for things to connect. This extra structure can save time but also means you must follow the framework's rules. The amount of structure is the second part of the difference.



Compare two simple pictures side by side. On one side, your code is the main program; it calls a library tool when needed, and the library responds. On the other side, the framework is the main program; it calls your code at specific points, and your code responds. Seeing both at once makes the difference clear: who calls whom? In the first picture, you call the library; in the second, the framework calls you.



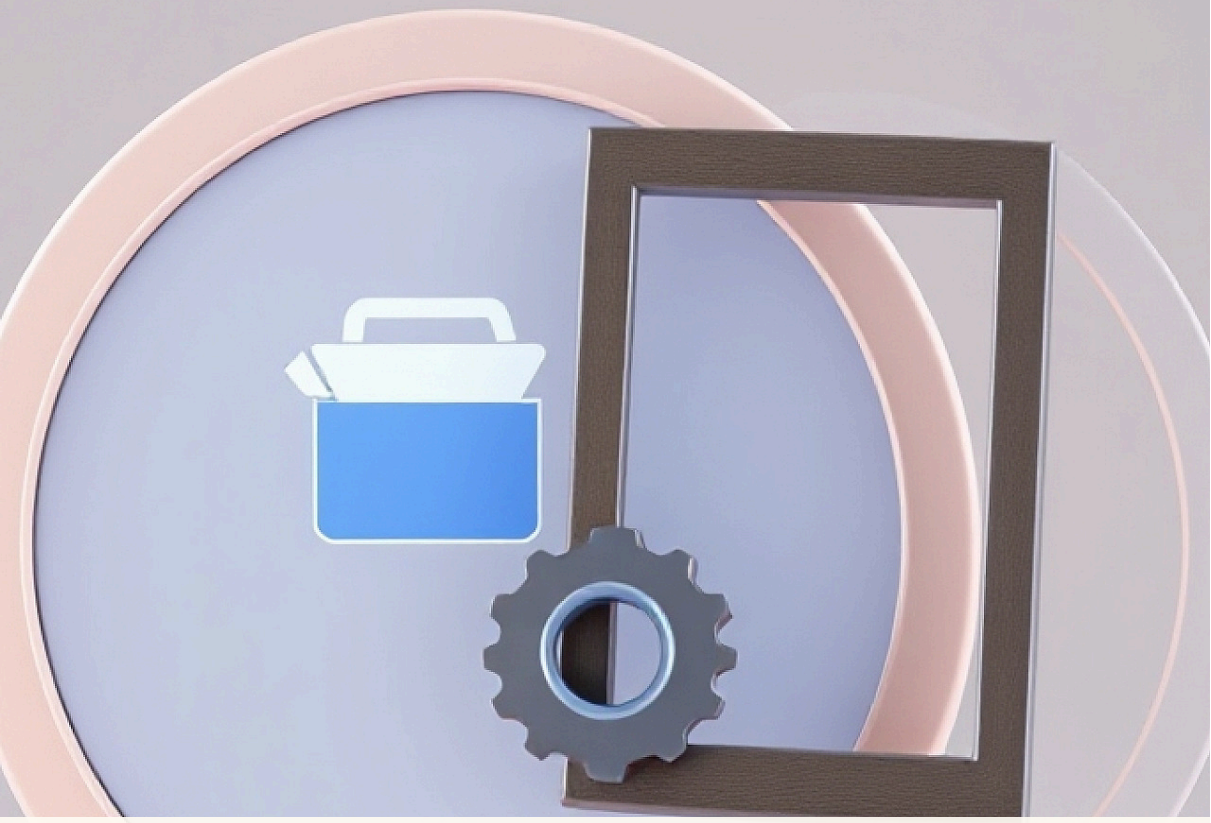
The line between library and framework is not always clear in every product. Some products sit in between: they provide tools but also impose some structure or call your code in certain ways. So the labels are not perfectly strict. You may find products that are called frameworks but act partly like libraries, or vice versa. It is more useful to look at how the product is used than to rely only on its name.



A common beginner mistake is to think the difference is about size: that a framework is just a bigger library. But size is not the point. A small framework can control flow, and a large library can still be just a set of tools. The real difference is who organizes the flow and how much structure is offered. So do not judge by the number of tools or lines of code; judge by control.



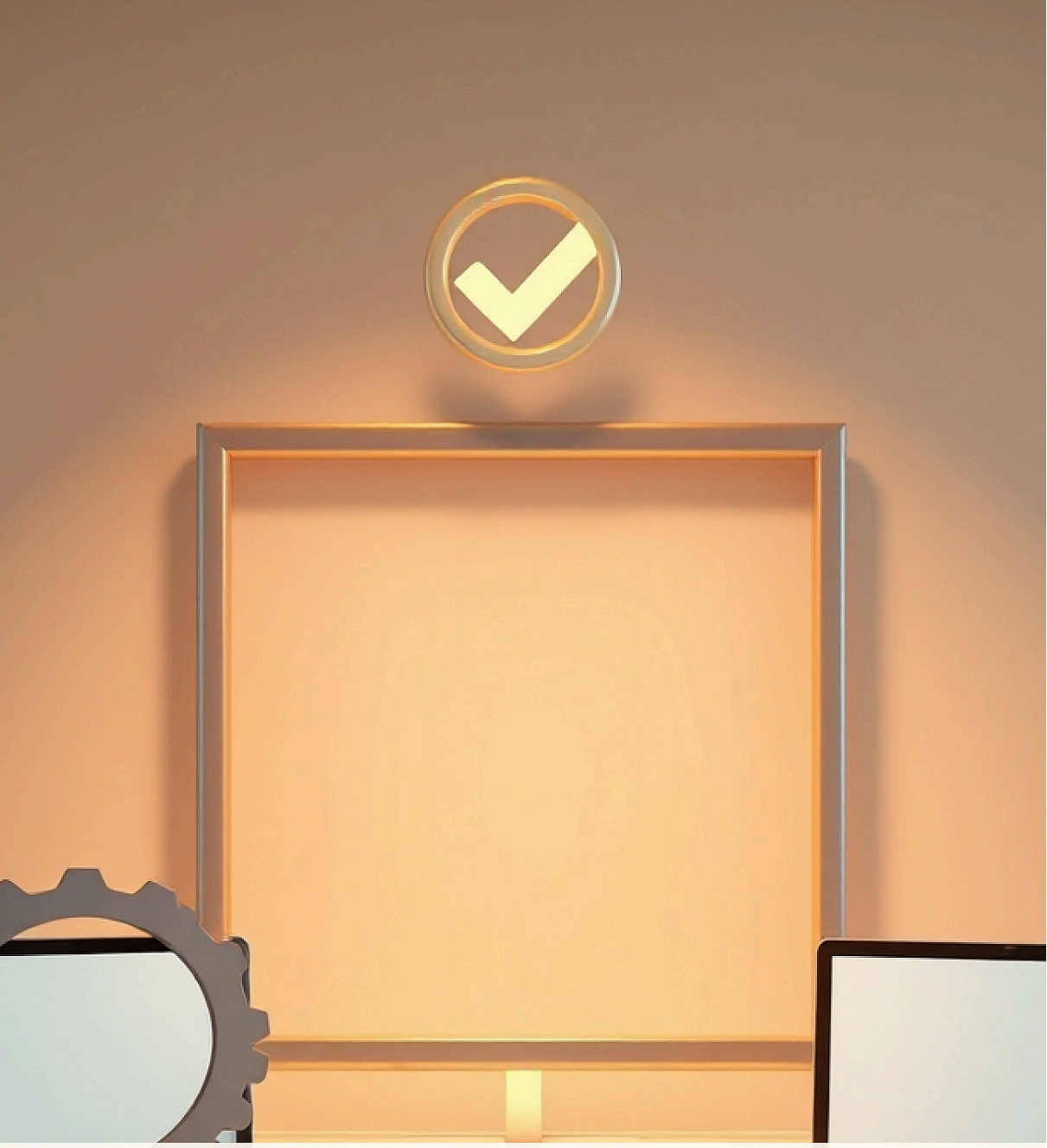
Another mistake is to think a framework is simply a library with more tools. That is not the real change. The key change is that control of the program's flow can move to the framework. In a library, you stay in control; in a framework, the framework takes the lead and calls your code. So a framework is not just extra code; it is a different way of organizing how your program runs.



Treating the two words as perfectly separate in every case is a mistake. The boundary can blur, so you should judge by who controls the flow, not just by the name. Some products may be called frameworks but only provide tools, and some libraries may offer some structure. Always ask: does my code call the product, or does the product call my code? That question is more reliable than the label.



A simple test for any product is to ask who calls whom. If your code calls the product to use its tools, then it acts like a library. If the product calls your code at specific points, then it acts like a framework. You can often tell by looking at examples: do you write functions that the product will call, or do you write code that calls the product's functions? That answer reveals the role.



In summary, a library offers tools that your program calls, while a framework can set the structure and control the flow of your program. Remember the key idea: who organizes the flow, and how much ready-made structure you get. Use the who-calls-whom test to decide in practice. With this understanding, you can better choose the right tool for your project and know what to expect.