What is debugging?



www.yzokul.com.tr

What is debugging?



Debugging is finding and fixing the cause of a problem in a program. A program is a set of instructions a computer runs, and sometimes it does something wrong. Debugging is the careful work of locating why that happens and correcting it. You do not need prior knowledge to start, only patience and a clear way of looking at each step.



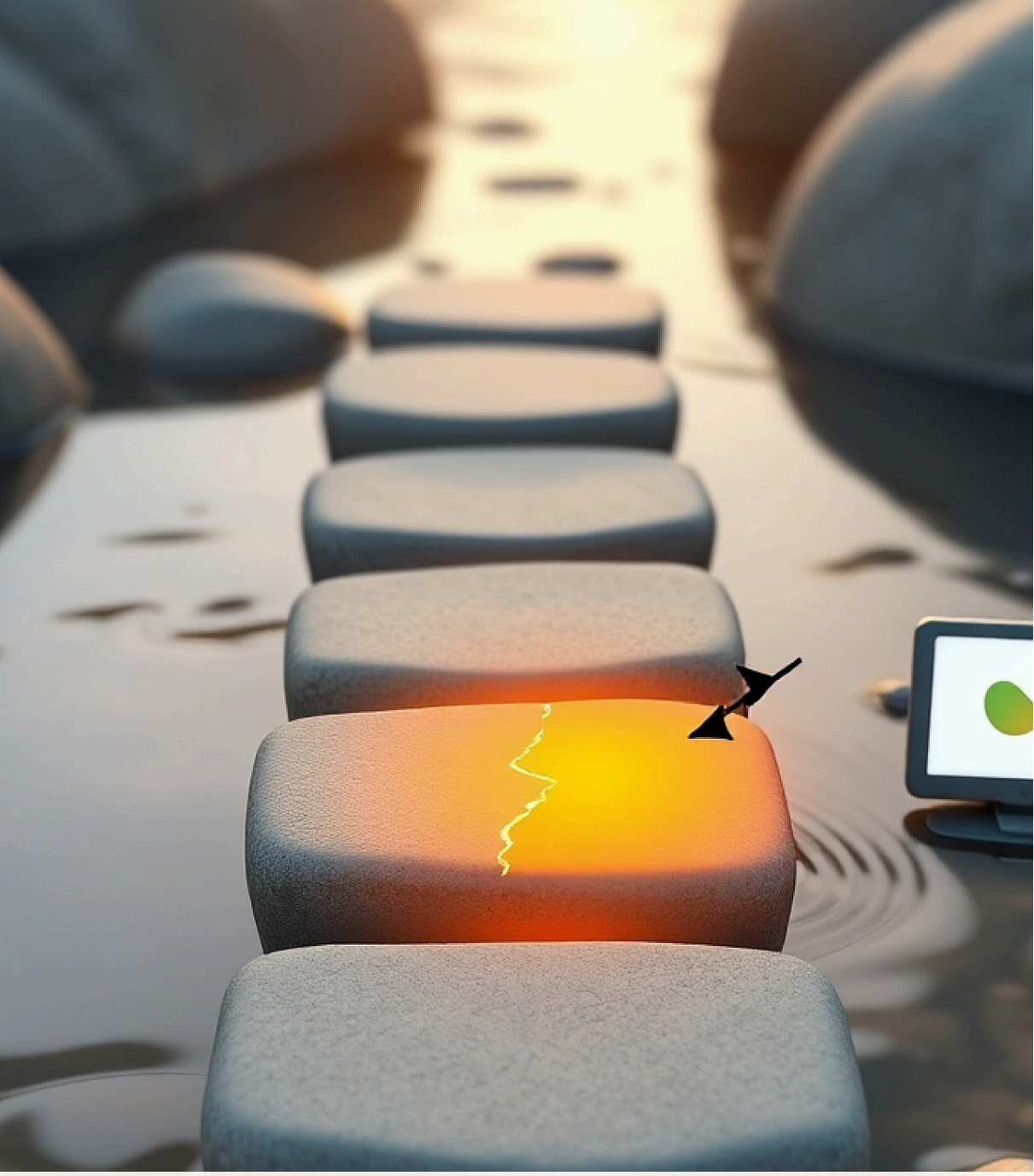
A program is a list of steps that a computer follows in order, one after another. Each step does a small job, such as adding two numbers or storing a value. Debugging only makes sense once you see a program as a sequence of steps, because a problem always sits at some specific step in that sequence. Read the steps from top to bottom.



A bug is a step where the program does something different from what was intended. It is not a mystery or a curse; it is simply a mismatch between expected behavior and actual behavior. Once you accept that a bug is a concrete mismatch, you can search for it calmly. Every bug lives at a findable place in the steps.



Expected versus actual is the heart of debugging. Write down what the program should do, then observe what it actually does. The difference between the two tells you that a bug exists and roughly where to look. Debugging starts from this comparison, so always keep the expected result in front of you while you inspect the actual result.



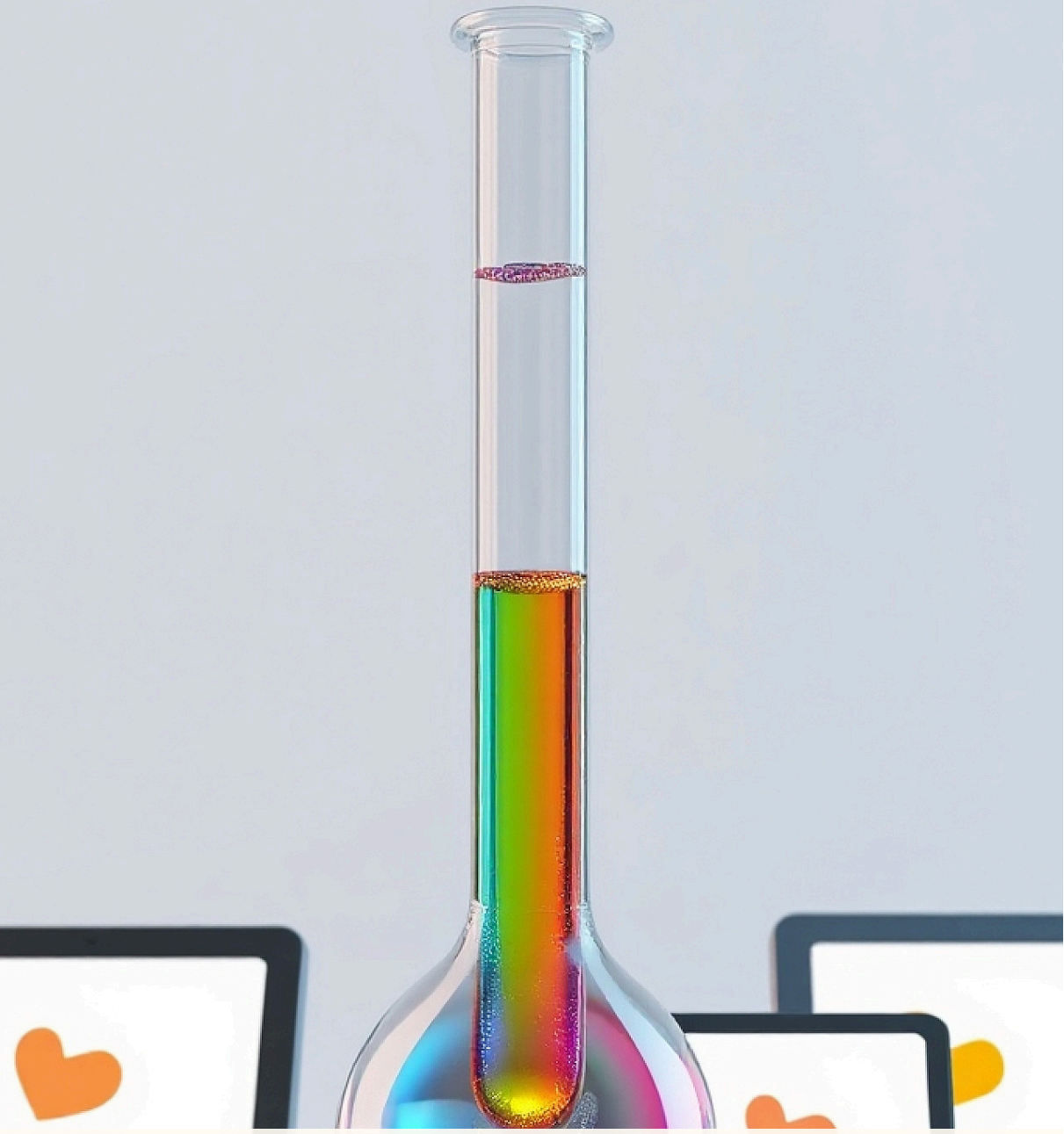
Find the step where behavior first differs from what you expect. Look step by step, not at the whole program at once, because the whole is too much to hold in your head. Check the first step, then the second, and so on, until the result stops matching your expectation. That first differing step is your starting point for testing causes.



Imagine a small program that should add three prices and show a total of 30. Instead it shows 27. The goal is not to guess but to find which step produced the wrong number. Check the prices as they are read, then the addition, then the display. One of those steps is where 30 quietly became 27.



Reproduce the error first, before changing anything. Make the error happen again on purpose by running the program with the same inputs in the same way. If you cannot reproduce it, you cannot test whether your fix works later. A reproducible error is a reliable target, so write down the exact steps that trigger it.



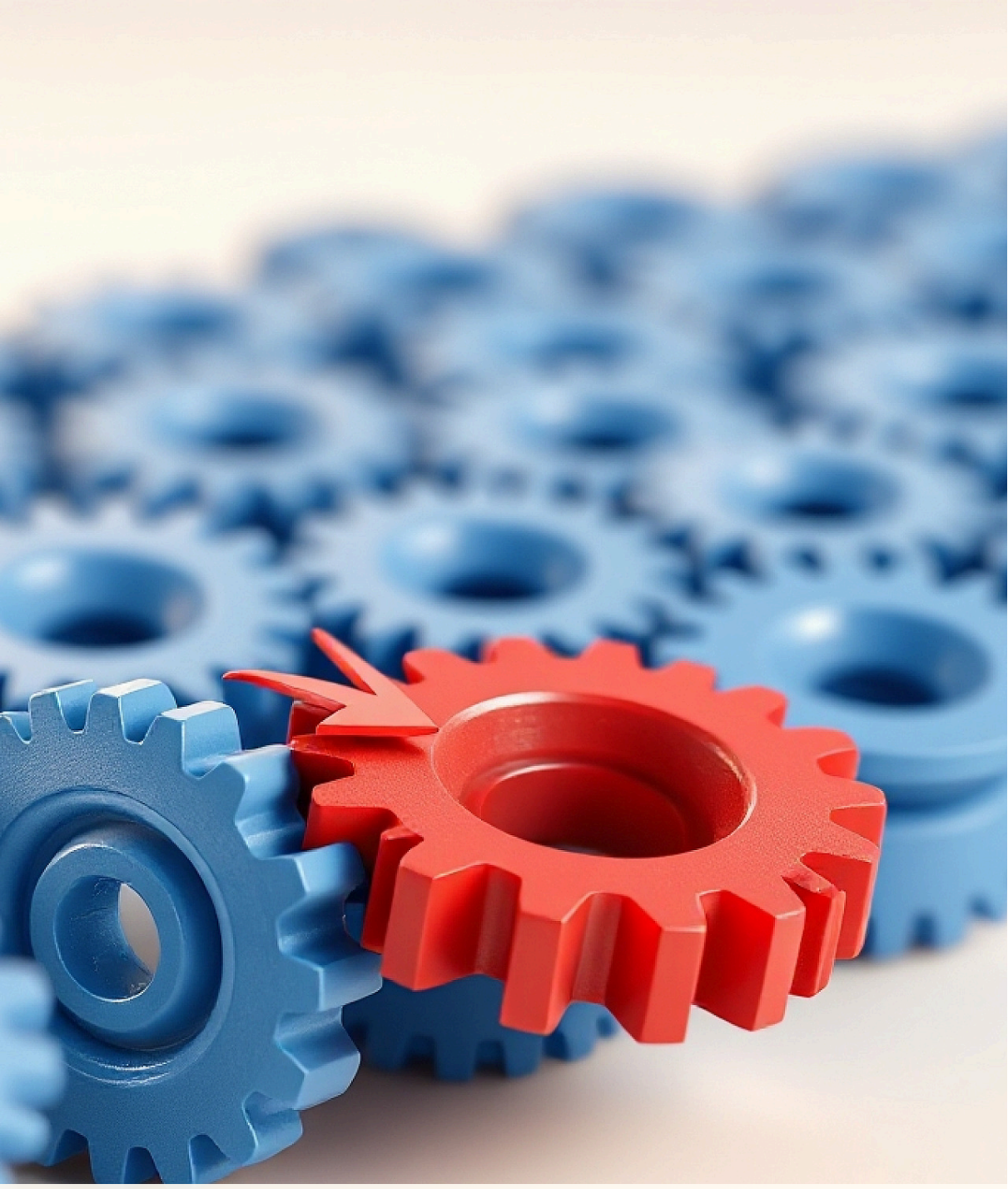
Print variables to look at their values at different steps. A variable is a named box that holds a value, such as a total or a count. Add a print line after each step and run the program again. Seeing the values shows exactly where a number first goes wrong, which narrows your search quickly.



Programs can write logs, which are records of what happened while the program was running. Each log line usually shows a message and sometimes a value at a certain moment. Reading the logs shows the path the program actually took, including steps you did not expect. Look for the last correct entry and the first wrong one.



Special debugging tools exist that let you pause a program and inspect it while it runs. A breakpoint is a marker that stops the program at a chosen line so you can look at the values there. These tools help you see the state at each step without adding print lines. Use them when printing becomes messy or slow.



Test one cause at a time. Form a single guess about what is wrong, such as a wrong price being read, and test just that guess. Change one thing, run the program, and observe the result. Changing one thing at a time keeps the results clear, so you always know what caused the change you see.



Fix the cause, not the symptom. The symptom is the visible wrong total; the cause is the step that produced it. If you only overwrite the wrong number with the right one, the bug stays and will appear again with other inputs. Repair the step that is actually broken so the correct result comes out naturally.



A common mistake is changing code randomly and hoping the problem goes away. Random changes hide the real cause and often create new bugs in other steps. You lose track of what helped and what did not. Resist the urge to edit quickly; test a clear cause instead, one change at a time.



Another common mistake is fixing without first reproducing the error. If you never saw the error happen on purpose, you cannot confirm that your change removed it. You might be fixing something that was never broken, or missing the real cause. Always reproduce the error first, then fix, then reproduce again.



Check that the fix works by rerunning the same steps that caused the error. Use the same inputs and the same actions you wrote down earlier. Confirm that the behavior now matches what is expected, with no new wrong values. This closes the debugging loop and tells you the cause is truly gone.



Remember that debugging is finding and fixing the cause of a problem by comparing expected and actual behavior. Reproduce the error, find the first differing step, and test causes one at a time using prints, logs, or debugging tools. Avoid random changes and always check the fix by rerunning the original steps.