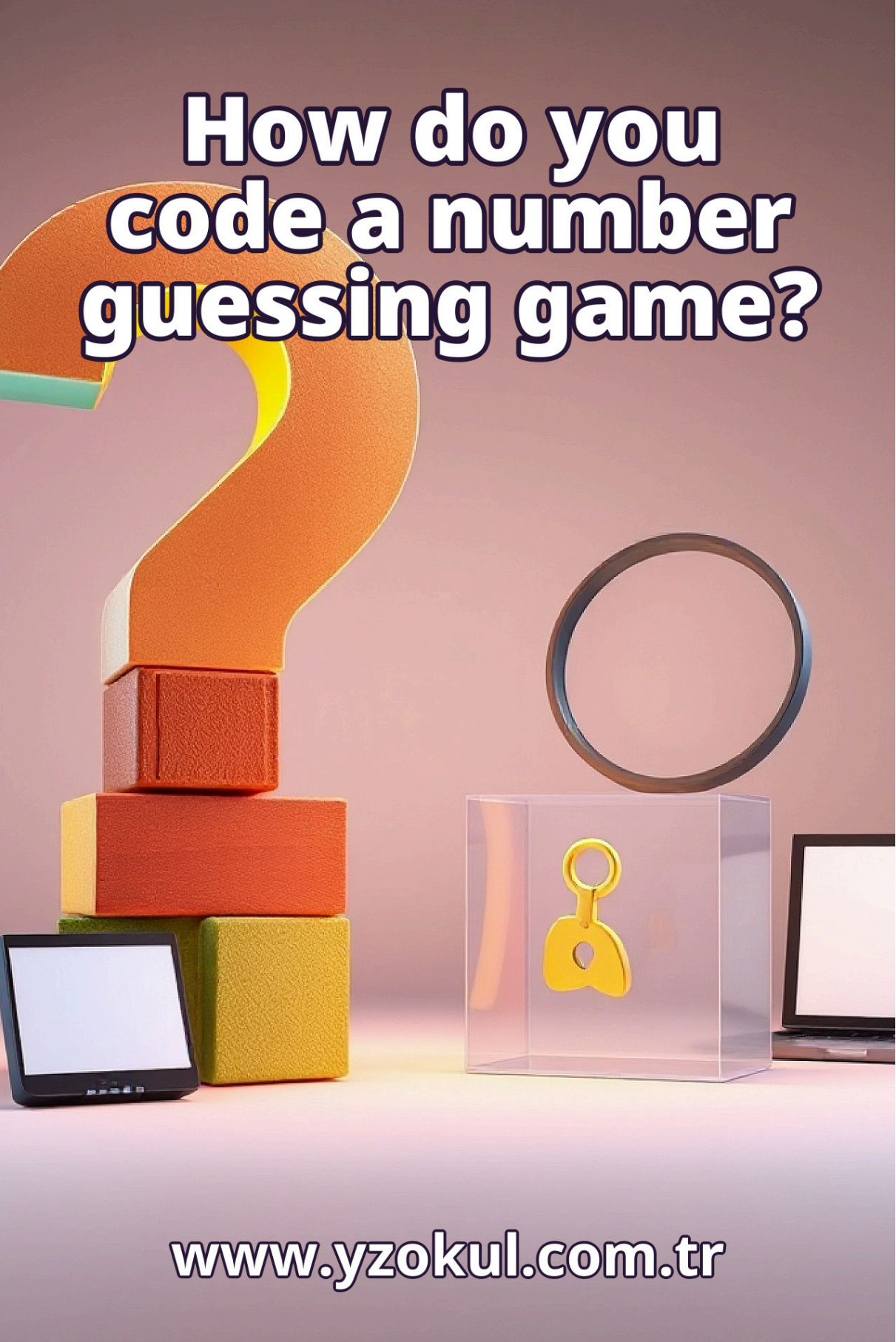
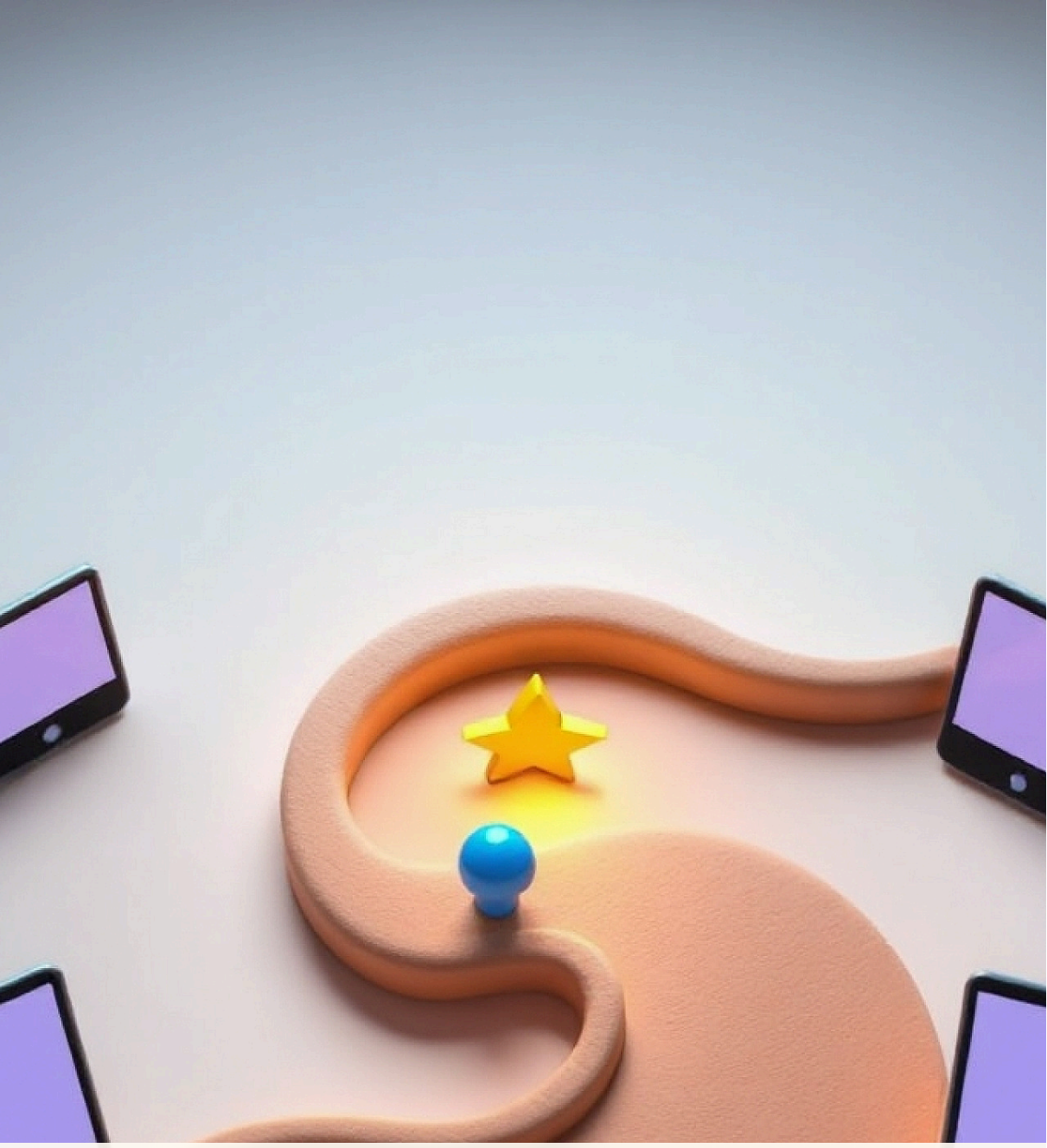
How do you code a number guessing game?



www.yzokul.com.tr

How do you code a number guessing game?



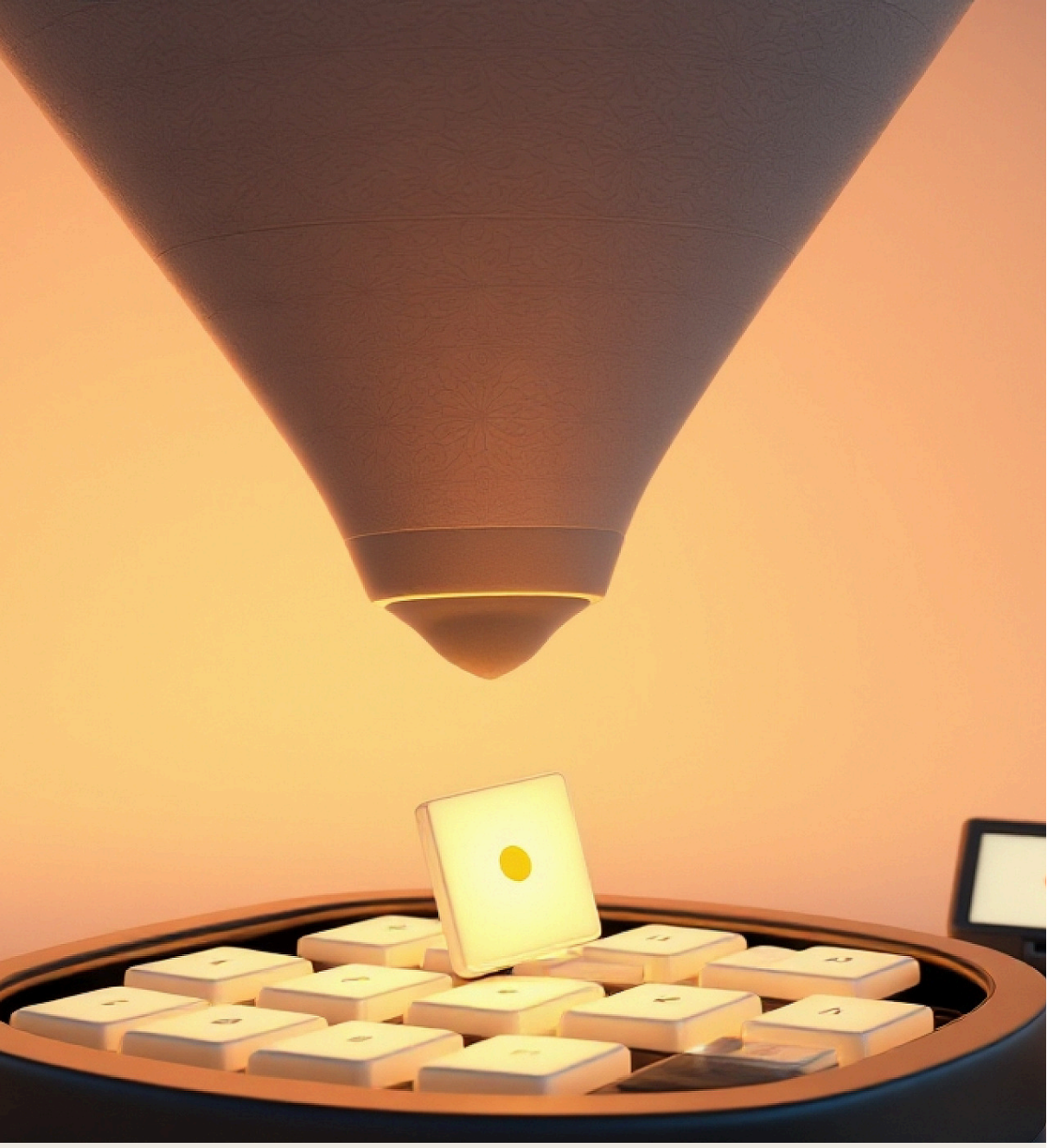
A number guessing game is a small program that plays a guessing contest with you. The computer picks a secret number, and you try to find it by typing guesses. After each guess the program tells you whether to go higher or lower. The game continues until you guess the secret number exactly. The short answer to how you code it: pick a secret number, compare each guess, give hints, and loop until correct.



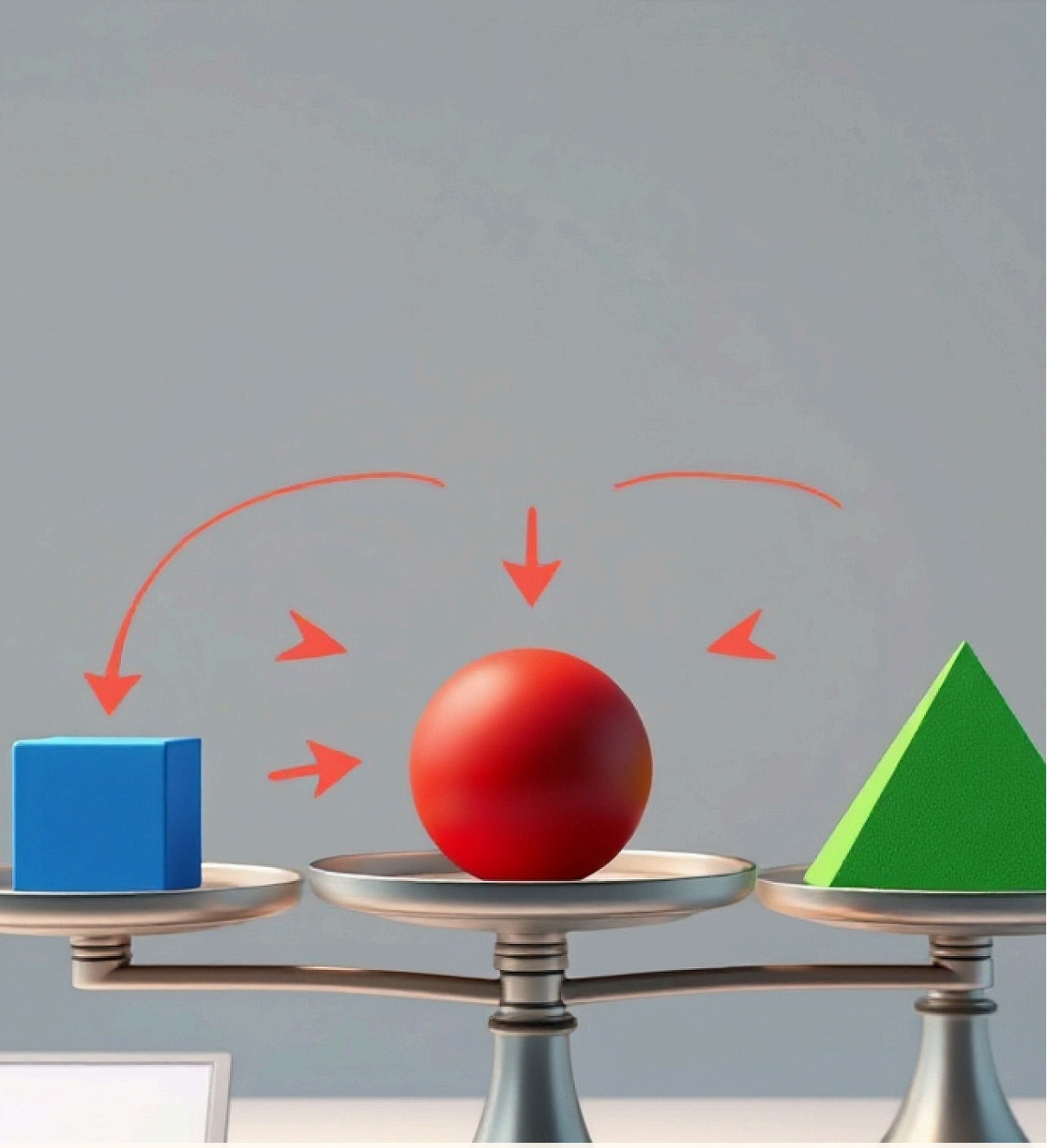
Start by making the program choose a secret number inside a fixed range, such as 1 to 100. The program uses a random number generator, a built-in tool that produces an unpredictable value, so the game is different each time. Store that secret number in a variable so the rest of the program can compare guesses against it. Keep the range small at first, like 1 to 10, so you can test the game quickly.



A variable is a named box in the computer's memory that holds a value. You give it a name, such as secret, and put a number inside it. Later you can read the value or replace it with a new one. In this game you use one variable for the secret number and another for the number of attempts. The name lets you refer to the value without remembering where it is stored.



The program now asks the player to type a guess and reads that input. Input from the keyboard usually arrives as text, so convert it into a number before using it. Many languages have a function for this, for example int(input()) in Python. Store the converted guess in a variable too, so you can compare it with the secret number. Without the conversion, the program cannot do number comparisons correctly.



Now the program checks the guess against the secret number. There are three possible results: the guess is equal to the target, less than the target, or greater than the target. Each result leads to a different message. This comparison is the heart of the game, because every hint and every win comes from it. A single comparison line can produce all three outcomes when combined with conditions.



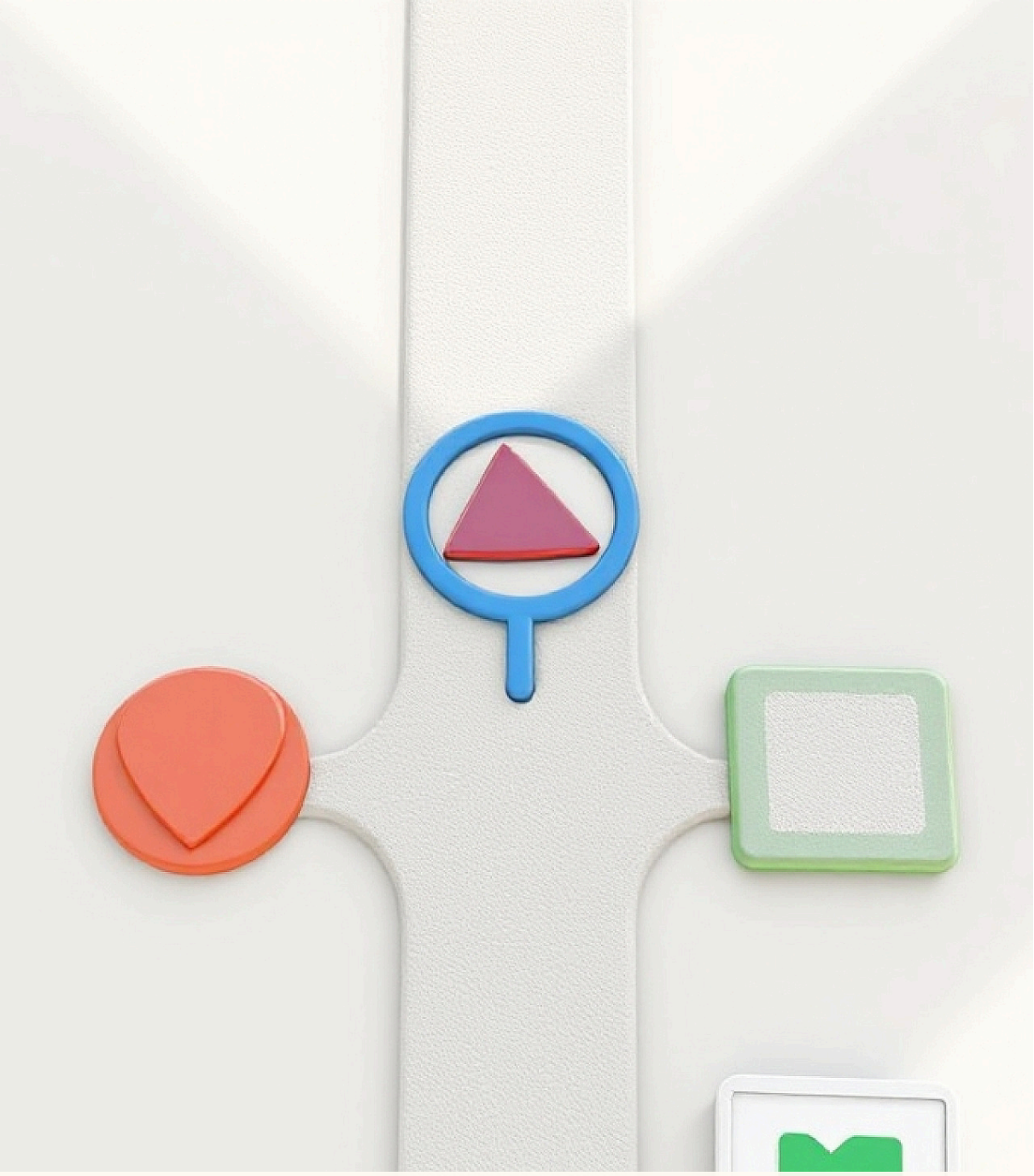
When the guess is smaller than the secret number, the program prints a hint such as higher. This tells the player that the secret number is larger, so the next guess should be bigger. The hint is just a printed message; it does not change the secret number. Giving a clear hint after every wrong guess is what makes the game playable and fair.



When the guess is larger than the secret number, the program prints a hint such as lower. This tells the player that the secret number is smaller, so the next guess should be smaller. Together with the too-low hint, this narrows the range of possible numbers step by step. The player uses these hints to home in on the answer, much like a search.



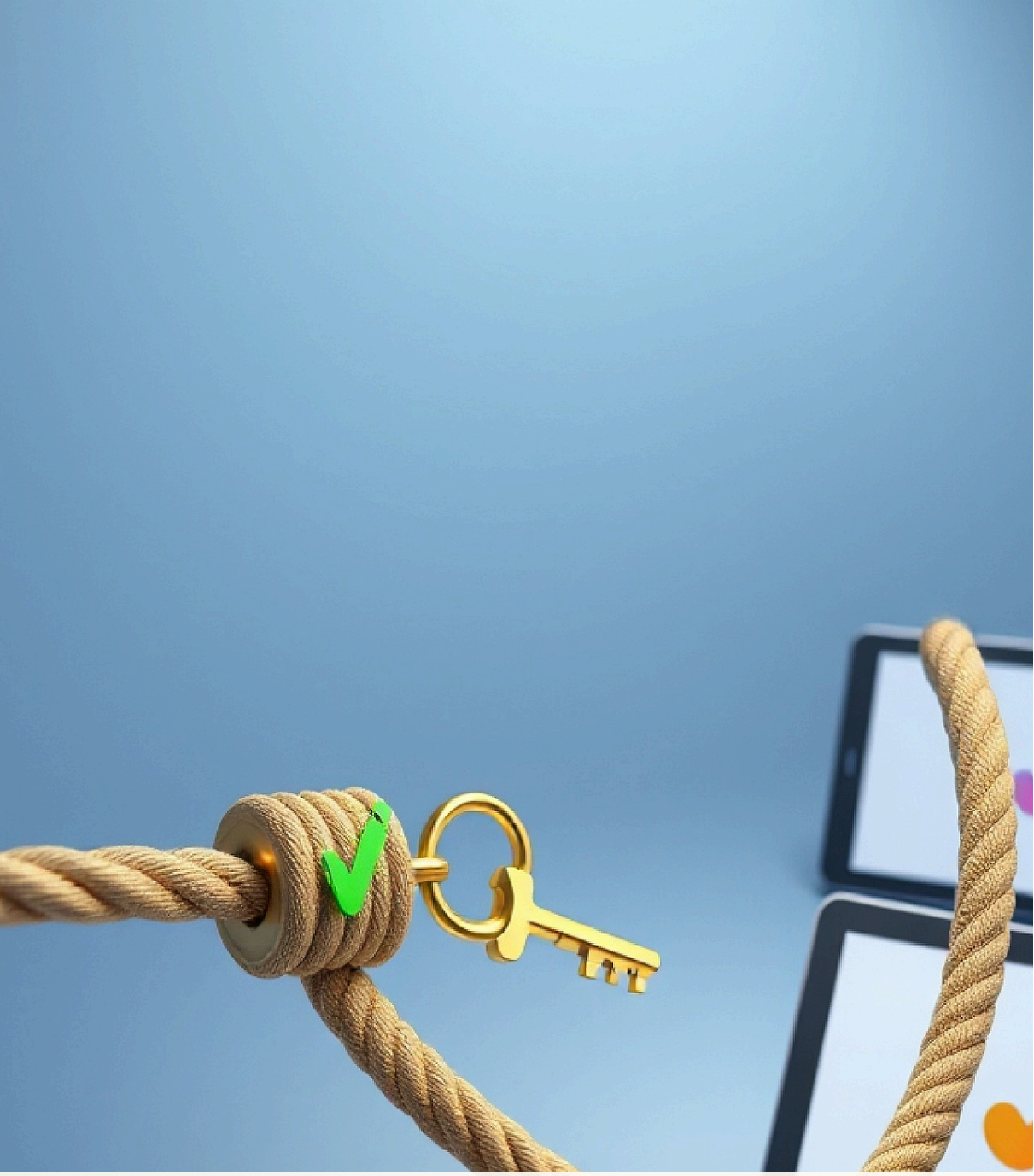
When the guess equals the secret number, the program tells the player they won. This is the win condition, the test that ends the game. Print a short message and, if you like, show the secret number so the player can confirm it. Nothing else should happen after this point except ending the loop or stopping the program.



A condition is a test that is either true or false, such as guess is less than target. The program uses conditions to decide which hint to show and whether the player has won. Conditions are written with comparison operators: equal, less than, and greater than. You chain them so exactly one branch runs for each guess: too low, too high, or correct.



A loop repeats a block of instructions again and again. Wrap the ask-compare-hint steps inside a loop so the player can keep guessing without restarting the program. Each pass through the loop is one round of the game. The loop is what turns a single question into a real game that lasts until someone wins.



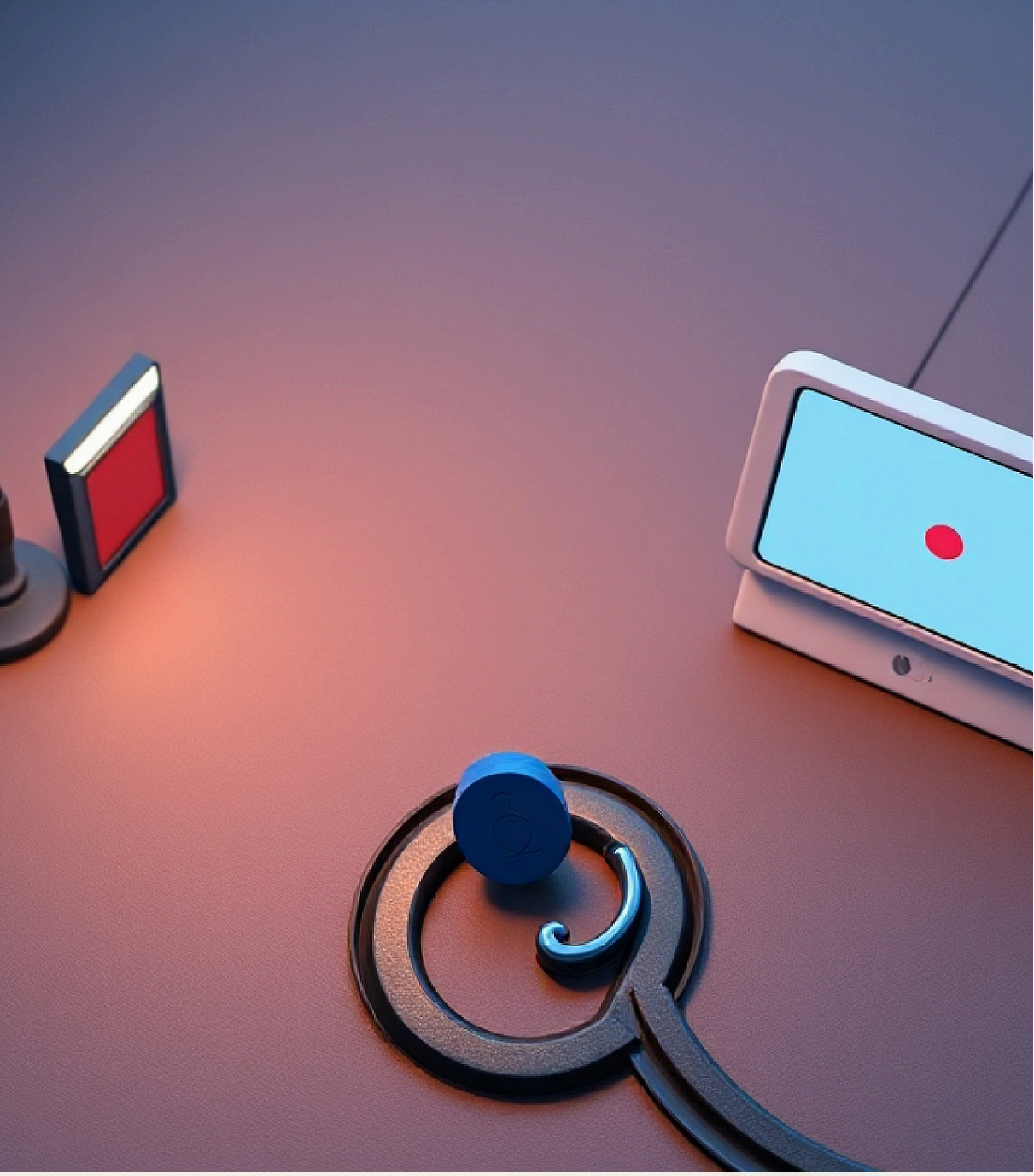
The loop must stop as soon as the guess matches the secret number. Connect the win condition to the loop: when the guess is correct, break out of the loop or set a flag that ends it. Without this link the program would keep asking forever, even after the player has won. Ending at the right moment is what makes the game feel finished.



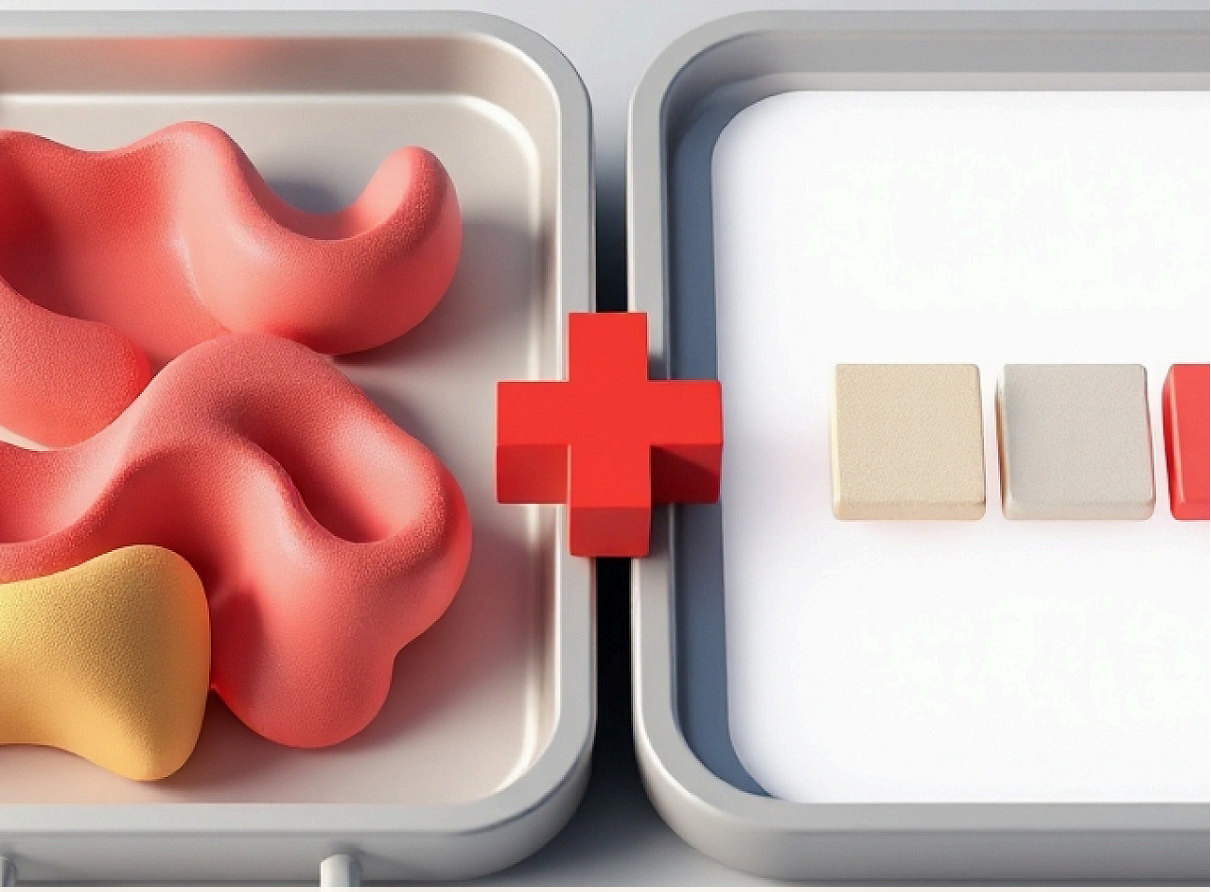
Add a variable that starts at zero and increases by one after each guess. This counts the attempts the player needs. At the end, print the count together with the win message, for example: you found it in 7 tries. The counter is a second variable that changes inside the loop, showing how variables and loops work together.



Now combine the pieces into one small program: pick a secret number, start the attempt counter at zero, then loop. Inside the loop, read a guess, convert it to a number, compare it, print higher or lower, and stop when correct. Each piece does one job, and together they form the complete game. Test it with a small range first, then widen the range.



A common error is asking for only one guess and then stopping. Without a loop the player gets just one try, so the game feels broken and unfair. The fix is simple: put the input, comparison, and hint inside a loop that repeats until the guess is correct. Check that your loop actually surrounds all three of those actions, not just the input.



Another common error is comparing typed text with a number directly. Input arrives as text, so convert it to a number before comparing, or the results will be wrong. In Python, wrap the input in int(), for example guess = int(input()). If the player types letters, the conversion fails, so a beginner version can simply assume numbers are typed.



The game shows how variables, user input, comparison, conditions, and loops work together in one small project. Remember the pattern: pick a secret number, compare each guess, give higher or lower hints, loop until correct, and count the tries. Once this works, you can add a limit on guesses or replay the game. Each new feature uses the same building blocks you already know.