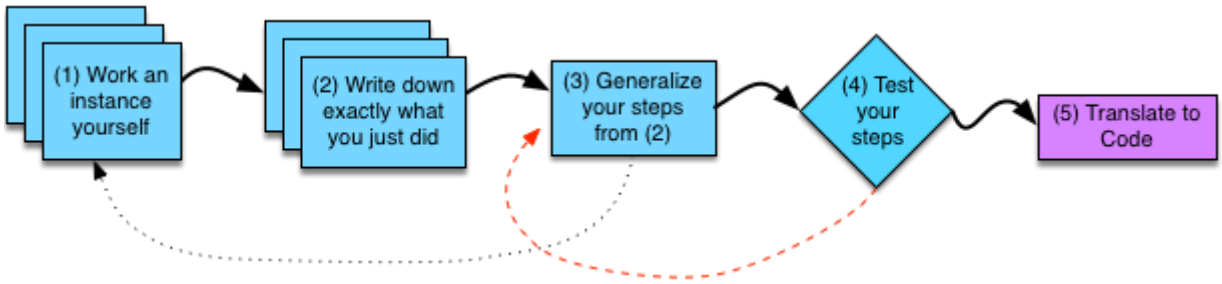
Step 1: Work an Example Yourself

The first step in trying to design an algorithm is to work at least one instance of the problem—picking specific values for each parameter—yourself (by hand). Often this step will involve drawing a diagram of the problem at hand, in order to work it precisely. The more precisely you can perform this problem (including the more precisely you can draw a diagram of the situation if applicable), the easier the remainder of our steps will be. A good example of the sort of picture you might draw would be the diagrams drawn in many science classes (especially physics classes). The figure shows multiple copies of the box for this step layered one on top of the other, as you may need to perform this step multiple times to generalize the algorithm properly.



One of the examples of an algorithm that we mentioned early in this chapter was determining if a number is prime. If you were trying to write a function to determine if a number is prime, your first step would be to pick a number and figure out if it is prime. Just saying "ok, I know 7 is prime," is not of much use—you just used a fact you know and did not actually work out the problem. For a problem such as this one, which has a "yes or no" answer, we probably want to work at least one example that comes up with a "yes" answer, and one that comes up with a "no" answer.

Another example would be if we wanted to write a program to compute x raised to the y power. To do Step 1, we would pick particular values for x and y, and work them by hand. We might try $x = 3$ and $y = 4$, getting an answer of 3^4 = 81.

If you get stuck at this step, it typically means one of two things. The first case is that the problem is *ill-specified*—it is not clear what you are supposed to do. In such a situation, you must resolve how the problem should be solved before proceeding. In the case of a classroom setting, this resolution may require asking your professor or TA for more details. In an industrial setting, asking your technical lead or customer may be required. If you are solving a problem of your own creation, you may need to think harder about what the right answers should be and refine your definition of the problem.

The second case where Step 1 is difficult is when you lack *domain knowledge*—the knowledge of the particular field or discipline the problem deals with. In our primality example, if you did not remember the definition of a prime number, that would be an example of lacking domain knowledge—the problem domain is mathematics, and you are lacking in math knowledge. No amount of programming expertise nor effort ("working

harder") will overcome this lack of domain knowledge. Instead, you must consult a source of domain expertise—a math textbook, website, or expert. Once you have the correct domain knowledge, you can proceed with solving your instance of the problem. Note that domain knowledge may come from domains other than math. It can come from any field, as programming is useful for processing any sort of information.

Sometimes, domain knowledge may come from particular fields of computer science or engineering. For example, if you intend to write a program that determines the meaning of English text, the relevant domain field is actually a sub-field of computer science, called Natural Language Processing. Here the domain knowledge would be the specific techniques developed to write programs that deal with natural language. A source of domain knowledge on English (an English professor or textbook) is unlikely to contain such information.