Adrian Prieto Follow
Lead Engineer at Futurism
Oct 11, 2016 · 3 min read

Stuck on a Coding Problem? Here are 5 Steps to Solve It

Solving problems is a programmer's bread and butter. While everyone has their own tricks to employ when they're stuck, I've personally found five surefire steps that, more likely than not, will help you solve any programming problems you encounter—*and* do it faster and more efficiently.

1. Read the problem several times until you can explain it to someone else

Read Read Read!

This is by far, the **most important** step. Read the problem several times until you fully understand it. If you don't understand it, you simply won't be able to solve it. And the best way to know if you understand the problem is by being able to explain it to someone else.

2. Solve the problem manually

> *Nothing can be automated that cannot be done manually!*

Any code we write has a foundation: the manual process. So before you start automating, before you start writing code like a maniac, *solve your problem manually first*. That way, you'll know exactly what you want to automate as you move forward. This will save you a lot of time.

Test your process with more than one input and some corner cases to validate it; pay close attention to every single step you take in your head and write them down—each step counts.

3. Make your manual solution better

Now, see if you can make your process better, if there is an easier way to do it, or if there are some steps you can cut to simplify it (like loops). This step is very important—remember that it's much easier to reconstruct your process in your head than it is in your code.

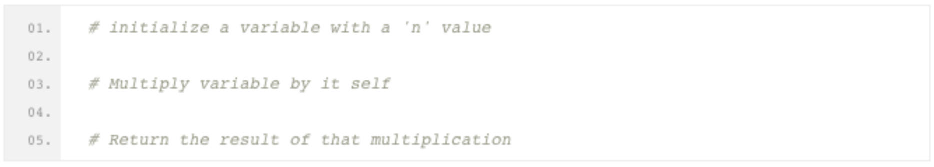
At this point, you will be tempted to write some code. Don't do it yet! We have one more step to cover, and I promise you it will make your final code easier to write.

4. Write pseudocode

"Pseudocode" is a detailed description of what a program must do; and writing it out will help you write every line of code needed in order to solve your problem.

Experienced programmers sometimes omit this step, but I can assure you: no matter how experienced you are, if you write some pseudocode, the process of writing your final code will be much easier since you only have to translate each line of pseudo code into actual code.

Here's what that might look like for "square (n)":



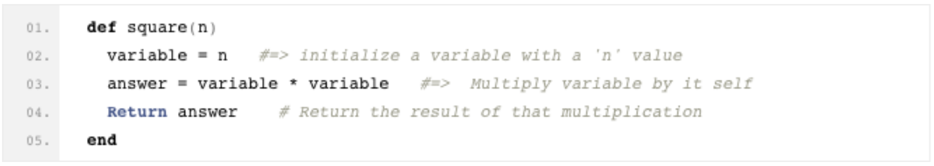
```
# initialize a variable with a 'n' value

# Multiply variable by it self

# Return the result of that multiplication
```

Now that we know exactly what our code is supposed to *do*, we have one more step… can you guess what it is?

5. Replace pseudocode with real code

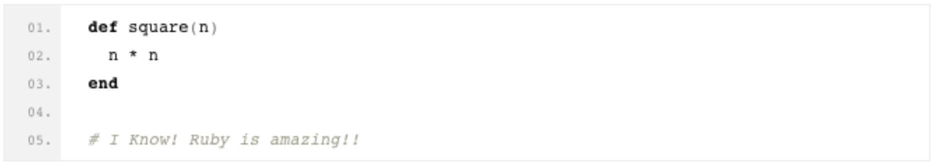
Here's the fun part. Now that you know for sure what your program should do, just write some code and test it. Remember, you can always make your code better along the way.

Let's take a look at how we'd do this with our square example:



```ruby
def square(n)
  variable = n   #=> initialize a variable with a 'n' value
  answer = variable * variable   #=>  Multiply variable by it self
  Return answer    # Return the result of that multiplication
end
```

Then, we can further optimize it:



```ruby
def square(n)
  n * n
end

# I Know! Ruby is amazing!!
```

Following this five-step process has helped me out of so many programming binds. No matter how complex your problem is, I assure you, these steps will help you solve it in less time and with fewer headaches.

Note: If your problem is too complex, divide it into small problems; it's a technique called "Divide and conquer".

. . .

Interested in learning how to code? I'm learning at Flatiron School, a coding bootcamp with campuses online and in NYC.