You notice your program sometimes gives you an error when uploading an image. Based on the scientific method, what would be the next step to take in order to solve the problem?

- Form a hypothesis
- Apply expert knowledge
- Gather information
- Ask a question

Correct

Continue

Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript

English

0:02
In the previous lesson we learned about the seven step process to write a program.

0:08
The seventh step was to debug failed test cases. In this lesson, we're going to talk more about step seven, debugging failed test cases. In particular, how would you go about doing this? This is an important step of the programming process because you aren't typically going to write your program correct to the first time. Everyone makes mistakes, and it's easy to have small problems that cause your program to work incorrectly. We need a good way to do this and we're going to use the scientific method. Now you may or may not remember the scientific method from elementary or middle school. Or you may have learned it a little bit differently, so we're going to review it here. The first step is that you observe a phenomenon. In the case of your program, this is generally that it crashes or exhibit some incorrect behavior when you test it.

0:56
The next step in the scientific method is to ask a question. In other fields of science, this might be something like, why does an apple fall to the ground? Or why do these birds have slightly different beaks? In programming, it's generally, why is my program broken? And how is my program broken so that I can fix it?

1:14
The next step is to gather information and apply expert knowledge. You may think you want to jump straight to forming a hypothesis but forming a good hypothesis is often very difficult. If you were Newton, you might not just figure out the theory of gravitation from seeing one apple fall. You might have to go do mini experiments with many different apples or other objects to come up with the equations of physics that he discovered. For programming, we're going to gather information about the inner workings of our program. We may need to do multiple experiments to see what's happening inside of there. We're also going to apply expert knowledge. This is the knowledge of the particular field in question, in this case programming to help us think about the data that we gather, analyze it, and understand what's wrong. This step is an iterative process. As you gather information, you're going to realize other information you want to gather, and other things you want to think about. You keep doing this step until you're ready to form a hypothesis. When you form a hypothesis you're going to make at some statement which predicts behavior about your program. I think that when my program does this, the following happens.

2:23
Once you've formed a hypothesis you're ready to test it. You're going to conduct some experiments to either reject the hypothesis if the behavior of your program contradicts it, in which case you're going to return back to gathering more information and applying your expert knowledge again. At this point, you've learned something new from the fact that your hypothesis was incorrect, so you have more knowledge to work from and are ready to form a better hypothesis. The alternative is we can become convinced that our hypothesis is correct because all of the evidence agrees with it, in which case we accept our hypothesis, we understand what's wrong with our program, and we're ready to act on it and fix our program.

3:05
To dive a little more deeply into the gathering information step, since this is one of the most time-consuming steps of the process. How do we actually go about this? Well, we need to look at the inner workings of our program to see what's going on inside of everything and it's doing. We could add statements that print things out. Print the values of variables. Print when we reach certain lines of code, or any other information we might find useful in this process. We may also use debugging tools. There are specific programs written to help people debug their programs. They let you step through the execution line by line, inspect the value of variables, even change the value of variables, or watch to see when variables will change. What particular tools are available depend on the language that you're working in, but mastering these tools can be incredibly powerful and helpful.

3:57
In some cases you may also find it instructive to execute your code by hand. Stepping through it line by line, writing down the effects of each line and figuring out what's going on which will let you see exactly what's happening in your program, which may lead you to some insights into its behavior. As you do all of this you're going to apply expert knowledge. You're going to gain more expert knowledge with experience as you become a more experienced programmer and debugger. Things will come more naturally to you because you'll have seen similar situations. You'll recognize what certain symptoms of a program mean and you'll be able to debug with more ease.

4:35
Speaking of our hypothesis, we'd like to talk a little bit more about those. What makes a good hypothesis? Well, it should be testable. That is, we should make a specific prediction about the behavior of our program, something that we can either refute if the program's behavior does not match that prediction, or something that we can become convinced is the case. It should also be actionable, that is something where once we've convinced ourselves that it's true, we can go fix our program because we now know what's wrong with it.

5:07
Being specific, as specific as possible, informing our hypothesis, helps with both of these aspects. Let's see a couple examples. First, we're going to start with an example of a very poor hypothesis which provides us no useful information. My hypothesis is my program is broken. While this may be true, and in fact likely is, if we've observed a failed test case, this does not tell us anything useful about how to fix our program.

5:35
A slightly better hypothesis would be, the problem is on line 5. This tells us a little bit about where something is wrong in our code. We might be able to act on it in some ways. We could go look at line 5 and see what we can figure out is wrong, but it doesn't really tell us what's wrong.

5:53
An even better hypothesis would be that the problem is division by 0 on line 5 of our program. This tells us specifically what's wrong, and where it's wrong. We could test this and see if we're dividing by 0, and we might be able to act on it, but we can do even better.

6:11
Here we have a very good hypothesis. The problem is division by 0 on line 5, specifically for inputs that need certain criteria. In this example maybe red is less than 30, and green is greater than 245. So this hypothesis is testable. We can go and craft specific inputs that tell us whether it's exactly these criteria, and exactly this behavior, and it's very actionable. Once we've convinced ourselves this hypothesis is true, we know exactly what to do to fix our program. We go back to steps one, two, and three of the seven step process, and we see what's special about this particular class of inputs, and how they need to fit into our algorithm as special cases, adapt their algorithm, and then fix our code.

6:57
When we want to test our hypothesis, the way we're going to do this is we're going to run our program. We'll take whatever test cases we find appropriate, use them as inputs to our program and run it and see what happens. If the program's behavior matches our prediction, or not is going to be the outcome of the test case. If it does not match the behavior, we're going to reject the hypothesis. Anytime we get contradictory information it indicates our hypothesis was not correct and we need to go make a new hypothesis.

7:27
If it is the same behavior that we predicted then we're not ready to accept our hypothesis immediately, we just become slightly more confident in our hypothesis. And we continue testing until we become confident enough that we're ready to accept our hypothesis.

7:43
Checking our hypothesis is very similar to gathering information. We could potentially just run it and see the output, but sometimes we need more information. We might want to print the state of internal variables or examine other things about the behavior of the program.

7:59
Now, in doing this, we've taught you a good way to debug a program. However, there's this temptation that many programmers often fall into, to make ad hoc changes. Maybe, if I just change this one thing, add a plus 1 here or a minus 1 there, maybe it will work. If I just tweak some code, hopefully it will fix it. I might get lucky and save some time because going through and gathering information and forming hypothesis can be kind of hard work. Well, this is a tempting, but it's a really poor idea. And we'd like to make a metaphor to going to the doctor for this. So, suppose your sick and you go see your doctor. Doctor, I'm coughing and don't feel well.

8:44
And so the doctor's job is to diagnose you much in the same way that your job is to diagnose and fix your program.

8:50
Does your doctor just randomly try things? Take this medicine and see what happens. If so, I would get a new doctor.

8:58
No, your doctor uses the scientific method. He or she is going to gather information running some tests, apply expert knowledge, all of the things he or she learned in medical school and has seen from experience in working on other patients, and figure out a good hypothesis. Test that hypothesis and then once he or she's convinced of what's wrong with you, take a corrective course of action based on that diagnosis.

9:26
So to sum up this lesson, we've seen the scientific method and discussed it as a basis for debugging your programs.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt
Lecture Slides pdf

Would you like to help us translate the transcript and subtitles into additional languages?