

## 10.9: DEBUGGING



Contributed by [Chuck Severance](#)  
Clinical Associate Professor (School of Information) at [University of Michigan](#)

Lists, dictionaries and tuples are known generically as *data structures*; in this chapter we are starting to see compound data structures, like lists of tuples, and dictionaries that contain tuples as keys and lists as values. Compound data structures are useful, but they are prone to what I call *shape errors*; that is, errors caused when a data structure has the wrong type, size, or composition, or perhaps you write some code and forget the shape of your data and introduce an error.

For example, if you are expecting a list with one integer and I give you a plain old integer (not in a list), it won't work.

When you are debugging a program, and especially if you are working on a hard bug, there are four things to try:

### reading

Examine your code, read it back to yourself, and check that it says what you meant to say.

### running

Experiment by making changes and running different versions. Often if you display the right thing at the right place in the program, the problem becomes obvious, but sometimes you have to spend some time to build scaffolding.

### ruminating

Take some time to think! What kind of error is it: syntax, runtime, semantic? What information can you get from the error messages, or from the output of the program? What kind of error could cause the problem you're seeing? What did you change last, before the problem appeared?

### retreating

At some point, the best thing to do is back off, undoing recent changes, until you get back to a program that works and that you understand. Then you can start rebuilding.

Beginning programmers sometimes get stuck on one of these activities and forget the others. Each activity comes with its own failure mode.

For example, reading your code might help if the problem is a typographical error, but not if the problem is a conceptual misunderstanding. If you don't understand what your program does, you can read it 100 times and never see the error, because the error is in your head.

Running experiments can help, especially if you run small, simple tests. But if you run experiments without thinking or reading your code, you might fall into a pattern I call "random walk programming", which is the process of making random changes until the program does the right thing. Needless to say, random walk programming can take a long time.

You have to take time to think. Debugging is like an experimental science. You should have at least one hypothesis about what the problem is. If there are two or more possibilities, try to think of a test that would eliminate one of them.

Taking a break helps with the thinking. So does talking. If you explain the problem to someone else (or even to yourself), you will sometimes find the answer before you finish asking the question.

But even the best debugging techniques will fail if there are too many errors, or if the code you are trying to fix is too big and complicated. Sometimes the best option is to retreat, simplifying the program until you get to something that works and that you understand.

Beginning programmers are often reluctant to retreat because they can't stand to delete a line of code (even if it's wrong). If it makes you feel better, copy your program into another file before you start stripping it down. Then you can paste the pieces back in a little bit at a time.

Finding a hard bug requires reading, running, ruminating, and sometimes retreating. If you get stuck on one of these activities, try the others.