Debugging

At the highest level debugging has two main steps:

1. Define/find the problem
    a. Where is it?
    b. Why is it happening?
2. Fix the problem

What is the problem?

Start by defining **how** you know something is wrong. Clarify:

- What did you expect to happen?
- What happened instead?

It is helpful while you are doing this to reproduce the error. Run the same steps a second time and make sure the error stays the same.

Generally, the answer to "what happened instead?" comes in two flavors:

1. An error
2. An inaccurate result

Where is the problem?

Errors

Understanding errors takes time and practice. Real Python has a nice article about understanding errors and slack traces, including some descriptions of common Python error types.
The main takeaways from looking at the error should be:

- The error type/message
- Finding the last line of **your** code referenced in the slack trace

A word of caution: error messages and slack traces are a great place to start, but they aren't always the final word on the source of your problem. Errors can sometimes be red herrings, so take them as useful information, but use them as one of many pieces of information you will use to find your bug.

Rubber ducking

Your error gives you a place to start, but there is likely a lot happening before that error that could have contributed to the problem. Part of figuring that out involves clarifying to yourself what your code is doing and how it is doing it.

Talking this process through to yourself is often called “rubber ducking”. It is a reference to the idea that you may be sitting alone at your desk working, but when going through this process you might pull out a small rubber duck and explain to the duck out loud what your code is doing and why. It might sound goofy, but stopping and explaining out loud your process can be really clarifying.

Some steps you can follow while you are doing this:

1. Start by restating the goal of the code, what you expected to happen, and what actually happened.
2. From a logical starting point, start explaining to the duck what each line of code does. Consider at each step what assumptions you might be making.
3. Every few lines (or every line, as appropriate) add a print() statement to confirm to yourself that your code is behaving as expected. If your data is large you can use a few sample data values to check that each process is doing what it should.

Why is the problem happening?

At a certain point you may have put in a print statement and, lo and behold, you find an output that isn’t what you expected it to be. The line that is the problem. But why is that line not doing what you expected?

- If you have a function or method in that line confirm your understanding of what you think it should be doing. Take that method out of your code and run it in the console below with some small fake data that you can manually calculate yourself. Is the .sum() actually summing all the values? Is it operating on the axis that you expect?

Fix the Problem

Once you have found the source and reason for your problem you can turn to fixing it. Don’t be fooled, finding the problem is at least half the work, so even though it can feel like you haven’t made much progress at this point you are a good chunk of the way towards removing your bug.

How to fix the problem depends a lot on what the problem was. Try googling a generic version of your problem (Ex. “dictionary not copying with .copy() python”). You can also rubber duck here to brainstorm solutions.

When in Doubt: Pair Programming

At any point in this process if you are stuck, reach out to a friend and try pair programming. More pair programming tips can be found in this other doc [TB-created]. As a general rule, though, always go into pair programming or asking for help with a few things clear in your mind:

1. What are you trying to accomplish?
2. What is the problem and how did you find it?
3. What are your guesses about what is happening?
4. What have you tried so far and what was the result?