The Art of Debugging

One of the most important skills you will acquire is debugging. Although it can be frustrating, debugging is one of the most intellectually rich, challenging, and interesting parts of [programming or circuit design or experiment design].

In some ways, debugging is like detective work. You are confronted with clues, and you have to infer the processes and events that led to the results you see.

Debugging is also like an experimental science. Once you have an idea what is going wrong, you modify your [program or circuit or experiment] and try again. If your hypothesis was correct, then you can predict the result of the modification, and you take a step closer to a working [program or circuit or experiment]. If your hypothesis was wrong, you have to come up with a new one. As Sherlock Holmes pointed out, “When you have eliminated the impossible, whatever remains, however improbable, must be the truth.” (A. Conan Doyle, *The Sign of Four*. (quoted with slight modification from “Think Python, 2nd Edition,” Allen B. Downey, pg. 30, O’Reilly Media, Inc., Sebastopol, CA, 2016. A free PDF version is available at http://www.thinkpython2.com/thinkpython2.pdf)

Although these comments come from a computer programming text, they apply to any debugging process as denoted by the addition of “[program or circuit or experiment]” whenever the above paragraphs refer to the object being debugged. You may be debugging an electronic circuit, a computer code, a homework problem, or a complicated experimental setup. The fundamental processes and skills are very similar.

If a [program or circuit or experiment] you are building or testing doesn’t work the way you think it should, you need to go methodically through the [program or circuit or experiment] to locate the error(s). The error(s) could be in the design of the [program or circuit or experiment], how you [coded the program or wired up the circuit or assembled the experiment] , or faulty components. You should follow the following steps as you approach the [program or circuit or experiment].

If you would rather randomly change things to see if you can make it work, you will find that the probability of making the proper corrective change is fairly small (vanishingly small if you have a complex [program or circuit or experiment]).

Some of these steps may not apply to all of the choices (program, circuit, or experiment). Only skip the step if it obviously doesn’t apply.
1.0 Debugging electronics

1. Safety first! In classes, we usually don’t deal with high voltages or high currents, but you should be aware of possible dangers in working with electrical devices. Also, think about whether you might damage any components as you are working on the circuit. You should always power the circuit off when swapping out components.

2. Think analytically about the circuit. Should your design work? What do you expect to measure at different parts of the circuit if it is working properly?

3. Check that your measurement device is set to read what you expect. For example:
   (a) Multimeter settings (voltage, current, resistance, etc.) are correct AND the meter is connected correctly (i.e., parallel for voltmeter, series for ammeter).
   (b) Oscilloscope settings (AC/DC coupling, triggering, active inputs, vertical gain, time base, trace intensity, etc.) are correct.
   (c) Verify that the battery and fuses on portable devices like multimeters are good.

4. Check voltages to identify the problem(s).
   (a) Check the voltages at the power supply with and without your circuit attached.
   If the voltages are incorrect without your circuit attached, you need a new power supply. If the voltages are incorrect with your circuit attached, you have a problem with your circuit – fix it before you reconnect your supply.
   (b) Check whether you are getting the expected voltages or currents at various places in the circuit. See item 2 above – you should have already thought about the circuit and what you expect to see at strategic points.
   It is best to check voltages on the component leads so you also detect poor connections.

5. Check the wires (including scope and multimeter leads). Wires can easily become shorted or broken. Sometimes this will result in intermittent problems.

6. Verify that the wires connect the things that you think should be connected and don’t connect things that shouldn’t be connected (this includes power supply buses on breadboards or circuit boards).

7. Check the board for bad connections. Breadboards can develop shorts between neighboring rows. The connectors can also be damaged so that components inserted into the board don’t connect reliably.
   Circuit boards may have damaged copper traces, or you may have solder bridges. Cold solder joints will often result in intermittent problems until they just fail altogether.
8. Test the components.

**Note:** Don’t just randomly replace components. Sometimes the circuit itself is causing components to fail and just replacing a part will cause the new one also to fail.

(a) Some resistors have not learned the color codes (especially the tolerance portion). Verify the resistance with an ohmmeter *with the power off*. It is also best to at least disconnect one lead on the resistor so you don’t measure the resistance of everything else in the circuit at the same time. If there is a discrepancy between the value measured with the ohmmeter and the color code, believe the ohmmeter.

(b) Capacitors should be removed from the circuit and discharged (*i.e.*, short the leads together) before they are measured.

(c) Check diodes, LEDs, transistors, etc. Most multimeters can be used to test these components. If you are not sure how to test them, you should ask for help.

9. Ask a classmate or TA to have a look at what you might be missing.

10. Invoke the “authority effect.” Circuits sometimes behave better if they are aware that you know what you are doing.

Forcefully reciting Ohm’s Law may be effective. Be sure you have it right – if you get it wrong, the circuit knows you have no authority, and your best bet is to abandon all hope.

11. As a last resort, a gentle blow to an offending piece of equipment may help. Generally, a three-foot drop onto a concrete floor is adequate. *This technique is best left to the professionals.*

If you feel this is necessary, indicate on the offending piece of equipment the problem as nearly as you can determine. This will assist the professional in determining the proper positioning for the gentle blow.

### 2.0 Debugging software

The process of debugging software is very similar to that for electronics. The required steps may also differ depending on the programming language being used.

1. Think analytically about the program. Should your design work? What do you expect to happen at different points of the program if it is working properly?

2. Can you design a “toy problem” for which you know exactly what all the outputs should be? Does it work correctly?

Be very careful with testing. In many cases, it is not possible to test all possible conditions, and you may miss the one that matters.
3. Choose an appropriate “measurement” technique or tool. You need to evaluate what is happening (e.g., the values of appropriate variables) identified in the previous step. That may mean using a debugger, writing out some values, or inserting a graph in the code.

4. In step 1, you identified what you expect to have happen at different points. Is it happening?

5. Are the input values correct? If you are reading data from a file, is it formatted correctly? Are you reading it correctly?

6. Are variables that are supposed to be local actually local? Are variables that are supposed to be global actually global? Are global variables consistently named?

7. If you are not using a strongly-typed language, do you have any mistyped variable names?

8. If you are using LabVIEW, is your data flow between modules correct? Do you need to do something to control module execution that is not properly handled by the data flow?

9. Are your argument lists for functions or submodules correct?

10. Are you correctly using functions or submodules? Did you check the documentation?

11. Check low-level functions or submodules individually to ensure they are working properly. Work your way up through the program from there.

12. Ask a classmate or TA to have a look at what you might be missing.

13. Don’t bother with the “authority effect.” The computer already knows that we are all amateurs at telling it what we want done.

14. As a last resort, shutdown and restart the computer. Don’t just reboot. You want the power off to be sure there is nothing hiding in the RAM. It is best to avoid the use of a hammer on the computer.

You still need to be methodical about the process. Random changes will result in premature gray hair and nervous tics.

3.0 Debugging an experiment

It is not possible to write a generic process for debugging a complex experiment. There are too many parts, and the interactions are too complex to generalize. However, carefully thinking through the design, connections, hardware interactions, software interactions, etc. may lead you to an idea of where to start checking. Isolating the problem to a single subsystem or instrument will greatly simplify the process.

This is why you are studying physics – to be able to solve really hard problems!
4.0 A few things to remember

The following list of the six stages of debugging has been observed on the back of a t-shirt. (I don’t know where the t-shirt came from so I can’t properly attribute it – it was observed on a student wandering the halls of the ESC. A Google search on the title of the list reveals at least 20 vendors selling t-shirts with this list as well as several mugs, a couple of backpacks, a one-piece for an infant, and the cover on a spiral notebook.)

You really should remember these stages as you go through any debugging process.

Six Stages of Debugging

1. That can’t happen.
2. That doesn’t happen on my [machine or circuit or program or experiment].
3. That shouldn’t happen.
4. Why does that happen?
5. Oh, I see.
6. How did that ever work?