Carnegie Mellon University
Computer Science Academy

Bugs!
A guide, written by students of CMU CS Academy

Bugs. They are inescapable and the bane of every programmer's existence, even experienced ones! Sometimes they are bugs at compile time, and other times they are in the logic of the program. Regardless, here are some processes, tips, and strategies to exterminate those bugs as efficiently as possible!

Compile-time Issue Steps
Nothing is showing up and there are scary words in the console!
A compile-time issue means that the computer was unable to understand the code that you wrote. This can mean anything from syntax issues, type errors, invalid calls, etc.

1. Find the console.
    a. The console is the grey box with the words "Console" and "clear"; typically located underneath the canvas window. This is where error messages will be written, often giving clues on what and where it went wrong.
2. Read the message in the console.
    a. If there is no message and the console is empty, that means it's likely a logic issue! See the next page for tips in that case.
    b. Go to the line number specified and check for potential issues based on the message. Common issues include syntax errors i.e misspellings, capitalization errors, indents, missing colons, and mismatching parenthesis.
3. If no issue can be found on that exact line number, check the lines above that line number.
    a. Oftentimes, the issue occurs on a previous line but Python doesn't notice and report the error later.
4. Take a break, then return to it.
    a. Sometimes the issue is so small that, if you stare at code for too long, your eyes will pass over the issue since your brain automatically assumes what's missing is there. By taking a break and then coming back to it, it helps reset your brain so that you can find that small single character that is off.
5. Worst case scenario: retype the code (if not too long)
    a. Sometimes if you've combed through the code several times and can't find anything, if you retype it, you subconsciously fix your error.

Logic Issues

Something isn't working correctly!

Logic issues occur when the program runs but does something that you didn't intend. This can be caused by a large variety of things, from having a small accidental logic issue like using a negative instead of a positive to not understanding the problem to begin with.

1. Check to make sure there's no compile errors.
   a. Check the console and make sure that there are no error messages. Sometimes it's not working because the computer can't even understand the code!
2. Attempt to narrow down a potential area of code that may be going wrong.
   a. What is going wrong with the program? Is it related to something specific? Can you think of a variable, or area of logic where something might be going awry? (It's okay if you can't though)
3. Do a preliminary skim check over that area from part 2.
   a. Check for basic, obvious errors such as code that is indented incorrectly, wrong variable or property names, etc. For graphics, make sure the values match whatever is in the canvas exactly.
4. Walk through the area of the code carefully and reason through what every line does.
   a. Use a whiteboard or paper! Draw or notate down what is happening in every line to see where something might be going wrong or unexpected. If you aren't completely certain what the result of the line is, then note that down as a potential error spot! Perhaps you may have made an incorrect assumption along the way. If there are variables involved that are used elsewhere, look at where those came from in case they were used improperly elsewhere.
5. Use print statements.
   a. If you still can't find the issue, try selectively putting print statements around the code. Print out relevant variables at key points to get information such as what values these variables hold to see if they are what you expect them to be.
6. Check to make sure you understand what is supposed to happen
   a. If everything is matching and everything seems correct, check to make sure you understand how everything is supposed to act. Perhaps you misunderstand the question or the intended behavior of a variable.
7. Once you find the error, if you know how to fix it, you've debugged your code! Otherwise, if it still doesn't work, it may mean that something else is wrong so continue repeating the process over again.

8. If you aren't sure how to fix it, then that's where the problem solving comes in! That could mean writing out the problem on a whiteboard, brainstorming what to do with friends, referring back to the notes, or even asking for help!

Other Tips

1. Talk to a rubber duck.
    a. Sometimes, if you're stuck on what to do and what should happen, talking through the problem is enough to help you solve it. When you talk about it out loud to a rubber duck or a friend, your brain starts to process it and sometimes it will just click what to do.
2. Murphy's law
    a. Think of worst case scenarios. Sometimes in most situations it works, but remember there are things like edge cases! Edge cases are situations that are unlikely but still possible and may require extra attention to take care of them properly.
3. Test Cases
    a. Test cases are another great way to test your code as well as debug it. This can involve breaking the problem into pieces, and then writing test cases to see if those pieces are acting as expected. By breaking it down, you then might be able to find the issue faster! It can also let you see certain situations that the code fails and therefore know what kind of logic you may have overseen.
4. Imagine you are explaining to a young child
    a. Sometimes, you find your problem, and know what to do, but not sure how to code it correctly. In that case, imagine you're speaking to someone really young. How can you break down what you want to do as basically as possible? Then use those instructions and translate it into code!
5. Breaks
    a. As mentioned before, breaks can be very important in debugging. Just like when you revise a paper, it's good to put time in between when you initially write it and when you revise it, the same goes for debugging. Your brain sometimes automatically assumes things are correct even if they're not. By taking a break and going back to it, you can better see what you really wrote and from there often catch bugs.

6. Google
    a. You might think it’s cheating or you shouldn’t need to google your issues, but even professional software developers who have been in the field for years google their problems sometimes! Google is our friend and there for a reason! Googling your error will sometimes be enough to help nudge you in the right direction of where to look and how to fix it.
    b. In general, be wary of copying bits of code from an online source directly! Oftentimes an issue can appear in many different contexts, and code that works to fix the issue in some contexts will break more things in another. So try to understand how the shared code is fixing the issue and make sure it works for your situation.

Helping Others Debug:

Now you’ve learned how to debug your own code, here comes the even harder part! How do you help others debug their code without directly giving them the answer? Debugging is an extremely important skill to learn so it’s important to have students learn to do it themselves! Here are some tips and tricks for doing so!

1. Questions
    a. Ask guided open ended questions to help lead them to look in the right place.
        i. If it’s a syntax or compile error, remind them of the console. Ask them to consider what the message might imply and see whether that pinpoints the issue. If they can’t seem to spot it, try giving them a little hint like “Look at line x”, or “Check how functions are formatted in python”
        ii. If it’s a logic error, ask them what the program is doing and what their code does. Based on their answers, sometimes you can naturally narrow the questions to help them hone in on their issue!
2. Notes
    a. The notes are there to be used! If you spot a logic issue from a misunderstanding, a small nudge to check that section can be a big hint to help students without giving away any

answers. It reinforces the knowledge by having them reread the notes, allowing the student to have a better understanding of the issue!

3. Trace It Through
    a. If asking questions isn't helping the student identify what's wrong, then trace it through on paper with them. Write out every variable, and its value on each step when applicable, and challenge them to explain how everything changes and why. If they can't explain how something works, refer them back to the notes or have them try just that piece of code! Trial and error is another great way for some students to understand how things work
4. Think Out-Loud
    a. Sometimes all of the above won't work and they still won't understand what's wrong, and that's okay! Especially for students who are new, it can be a new way of thinking and approaching a problem. In that case, it can be good to talk out loud and debug it. In other words, go through the process of debugging, and talk through every step as you reach it! By seeing your process, students can start to mimic it and get a sense on how to debug themselves! It can also be a great activity to do in front of the class to show students the process of debugging.
5. Have the Student Type
    a. A little more of a tip than a method, but have the students type everything! Even if you sometimes have to tell them exactly where their bug is, by having them type and fix the bug, they still have to think about it, at least a little!