Troubleshooting Python

Click to schedule a meeting with the DITI Team

One of the most important rites of passage when it comes to learning any programming language is encountering your first error message. While error messages can be confusing, scary, or even demoralizing, they are entirely normal and part of learning how to code; even seasoned programmers frequently receive error messages. Think of it this way: a programming language is just like learning any language, and sometimes, you're going to misspeak or forget where to put the comma or how to spell a word. Reminding yourself that programming languages are foreign languages with their own vocabulary and grammar can make error messages feel less intimidating.

How Python Executes Code

Unlike **compiled** programming languages—like C++ or Java, which compile all lines of code together before attempting to execute, or run, the code—Python is an **interpreted** language, which means that the code is run one line at a time. While compiled languages have the benefit of speed, interpreted languages have the benefit of being much easier to troubleshoot. Chances are, if you have encountered an error in your code, you likely already know exactly where that error is (or at least have very few possibilities) since Python only produces an error message if it reaches a line of code that it cannot execute. There are a few approaches to troubleshooting errors in Python, but the most important first step is to actually identify where the error is in the first place.

Warnings vs. Errors

There are a number of different messages which you may encounter when running Python code. Some of these messages may appear to be very scary if you have never run into them before, but there are important differences between the two major forms of display messages that you are likely to encounter as you build your experience in programming. In either case, it is critical that you **read the message your computer shows you**, because the information that you need to debug the issue will be in the message.

Developed by: Avery Blankenship
Questions? Contact us: nulab.info@gmail.com

Warning Messages

A warning message is exactly what it sounds like on the tin: it's a warning message that the computer is producing in order to make you aware of something in your code—a method, library, syntax, etc.—that may not prevent the code from executing, but could introduce some other type of problem down the road, or may cause the code to execute in unexpected ways. Most commonly, you will see warning messages in cases of **deprecation**, where a library or module you are using in your code will soon become obsolete or in cases where your code may exceed the amount of working memory your computer is allowing it to use.

- For **deprecation warnings**, the updated syntax is typically included in the warning message itself
- For **memory warnings**, your code needs to be rewritten in order to limit the number of times that memory needs to be accessed
    - It is usually not a good idea to increase the amount of memory the code is allowed to use for the sake of your computer's functionality

Error Messages

An error message displays when your code is unable to be executed any further. An error message is indicative of a mistake in the code itself rather than a sub-optimal line of code or a slightly out of date library. The error message will include more information about the error as well as the line number where the computer encountered the error. For new programmers, the two most common types of errors are:

- **Type errors**: these errors arise when two variables of different types have been added or when a value of an incorrect type has been passed to a function
- **Syntax errors**: these errors arise when the syntax itself is incorrect. Remember that Python is a case sensitive language and a language where spacing matters; often, even a misplaced space can result in a syntax error

Debugging with Print Statements

Sometimes, when you read an error message, you know that something is wrong, but the message doesn't exactly make it clear **what** is wrong. Let's say, for example, that you are getting an error message for a long script of Python code you have written, but the error message is only being thrown on the final function call at the end of the file where all of the functions and loops you have written actually get called and executed. In this case, the line number provided in the error message and Python's interpreted style won't directly help you debug the code,

because you've written code that is inherently compiled. In this case, you need a way of determining where, in this pseudo-compiled code, the error is.

A useful way to debug code in this instance is to use a series of print statements placed throughout your code. While you may have written code that is compiling a bunch of code into a function and only running that function when you actually call it, once that function gets called, it is executed one line at a time. If a print statement at a specific line within the function definition fails to print, then you'll know that the error has to occur at least before that line.

For example, let's look at the code block below:

```
define sample_function(x):
     y = 2
     if x == y:
          print("first condition no problem")
          z = x
     elif x < y:
          print("second condition no problem")
          z = y
     else:
          print("third condition no problem")
          z = 0

     print("full conditional no problem")
     return z
```

In the sample code above, a function is being defined which will accept an integer, `x`, and use a conditional statement to compare that integer to a local variable, `y`, and then return the variable `z` if certain conditions are met. After each condition, a `print()` statement is added that describes what has happened prior to the print statement. If the print statement successfully displays, then that means the lines above the statement have all executed, given that Python is an interpreted language. If, for example, the print statements stop printing at the elif condition, then we now know exactly where the error is and can adjust the code.

While Python also has a built-in debugger which follows a similar process using what are called "breakpoints," the integrated interface can be clunky and difficult to use for beginners. If you are interested in learning more about the built-in debugger, you can read about it in the Python documentation as well as this walkthrough.

Developed by: Avery Blankenship
Questions? Contact us: nulab.info@gmail.com

Digital Integration Teaching Initiative

What To Do After Finding The Error

After you have located the line that is causing the problem in your code, the fix may be as simple as deleting an extra space or adding a missing parenthesis. The first step to fixing an erroneous line should always be to make sure that there isn't an issue with the syntax itself, even if the error message doesn't identify a syntax error. For example, if you are missing a parenthesis, this could cause whatever is within the parenthesis to be read in the wrong order; or, if you mistyped a variable name, you may be assigning a value to the wrong variable.

Once you have made sure that the syntax is correct, the second step should be to walk through the code yourself by hand. One of the best ways to identify what is going wrong with a particular line of code is to ask yourself what the computer is actually assuming—and don't just assume, do it yourself. It's a good idea to begin at the earliest line of code that makes sense (the beginning of the function definition, the beginning of the loop, etc.) and to mentally walk through the process of executing the code, even writing down values, until you arrive at the line that is giving you an error. Often, this process will reveal any logical errors in your code.

Finally, in cases where the error message doesn't make it clear what the error actually **is**, it is always a good idea to Google search the error message or to look the message up otherwise. Not only does the Python documentation provide a robust explanation of various error messages, but the community at StackOverflow, a popular coding forum, is often willing to chip in to help collectively debug errors.

Developed by: Avery Blankenship
Questions? Contact us: nulab.info@gmail.com