Debugging strategies

As a security analyst, you might be required to read or write code.
One of the biggest challenges is getting it to run and function properly.
In fact, fixing complex errors in code can sometimes take up just as much, if not more, time than writing your code.
This is why it's important to develop this skill.
Now that you've learned the basics of coding in Python, it's important to learn how to deal with errors.
For that reason, we'll focus on debugging your code.
Debugging is the practice of identifying and fixing errors in code.
In this video, we'll explore some techniques for this.
There are three types of errors: syntax errors, logic errors, and exceptions.
Syntax errors involve invalid usage of the Python language, such as forgetting to add a colon after a function header.
Let's explore this type of error.
When we run this code, we get a message that indicates there's a syntax error.
Depending on the Python environment, it may also display additional details.
We'll typically get information about the error, like its location.
These syntax errors are often easy to fix since you can find exactly where the error happened.
They are similar to correcting simple grammar mistakes in an email.
Since their message tells us the problem is on the line that defines the function, let's go there.
In this case, we can add a colon to the header and resolve our error.
When we run it again, there's no longer an error message.
This is just one example of a syntax error.
Other examples include omitting a parenthesis after a function, misspelling a Python keyword, or not properly closing quotation marks for a string.
Next, let's focus on logic errors.
Logic errors may not cause error messages; instead, they produce unintended results.
A logic error could be as simple as writing the incorrect text within a print statement, or it might involve something like writing a less than symbol instead of less than or equal to symbol.
This change in operator would exclude a value that was needed for the code to work as intended.
For example, imagine that you reach out to a response team when the priority level of an issue is less than three instead of less than or equal to three.
This means all events classified as priority level 3 could go unnoticed and unresolved.
To diagnose a logic error that's difficult to find, one strategy is to use print statements.
You'll need to insert print statements throughout your code.
The print statements should describe the location in the code; for example, "print line 20" or "print line 55: inside the conditional".
The idea is to use these print statements to identify which sections of the code are functioning properly.
When a print statement doesn't print as expected, this helps you identify sections of the code with problems.

Another option for identifying logic errors is to use a debugger.
A debugger will let you insert breakpoints into your code.
Breakpoints allow you to segment your code into sections and run just one portion at a time.
Just like with the print statements, running these sections independently can help isolate problems in the code.
Let's move on to our last type of error: an exception.
Exceptions happen when the program doesn't know how to execute code even though there are no problems with the syntax.
Exceptions occur for a variety of reasons.
For example, they can happen when something is mathematically impossible, like asking the code to divide something by 0.
Exceptions might also happen when you ask Python to access index values that don't exist or when Python doesn't recognize variable or function names.
Exceptions also may occur when you use an incorrect data type.
Let's demonstrate an exception.
Let's say you have a variable called my_string that contains the word "security".
Since this string has 8 characters, we can successfully print any index less than 8.
Index 0 contains "s." Index 1 contains "e." And index 2 contains "c." But, if you try to access the character at index 100, you'll get an error.
Let's run this and explore what happens.
After it successfully prints the first three statements, we get an error message: "string index out of range." For exception errors, you can also make use of debuggers and print statements to figure out the potential source of error.
Errors and exceptions can be expected when working in Python.
The important thing is to know how to deal with them.
Hopefully, this video provided some valuable insight about debugging code.
This will help ensure that the code that you write is functional.

Revision #1
Created 2024-01-02 22:15:22 UTC by naruzkurai
Updated 2024-01-02 22:16:08 UTC by naruzkurai