

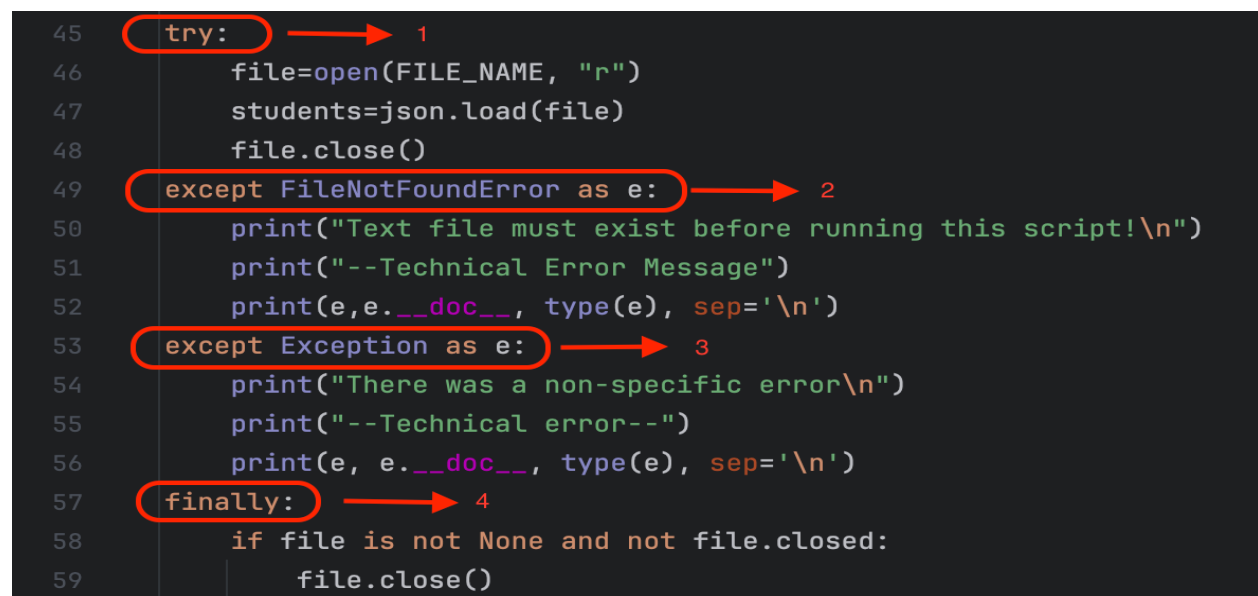
The Try and Except Approach in Python Programming

Introduction

Module 05 went over quite a variety of interesting topics, but the one that caught my attention was Structured Error Handling and why the Try-Except construct is the recommended approach. I personally had no idea that there was such a concept, what started off as frustrating to comprehend turned out to be a valuable tool that could potentially enhance my programming skills in the future.

What is Try and Except in Python Programming?

In Python, **Try** and **Except** are used to handle exceptions, which are error that can occur during program execution. “try” let’s you test a block of code of errors while “except” let’s you handle the error. It’s neat in a sense that you may customize the error handling or the messages to your liking.



```
45 try: → 1
46     file=open(FILE_NAME, "r")
47     students=json.load(file)
48     file.close()
49 except FileNotFoundError as e: → 2
50     print("Text file must exist before running this script!\n")
51     print("--Technical Error Message")
52     print(e,e.__doc__, type(e), sep='\n')
53 except Exception as e: → 3
54     print("There was a non-specific error\n")
55     print("--Technical error--")
56     print(e, e.__doc__, type(e), sep='\n')
57 finally: → 4
58     if file is not None and not file.closed:
59         file.close()
```

Figure 1.1 try, except, finally used in blocks of code

- 1.) try – try here tests the first block of code for any errors that may occur with opening the file
- 2.) except – note that “e” here is used a variable to capture reference to the automatically generated Exception object. FileNotFoundError – as the

name suggests, this error will occur when the file or data does not exist. You can customize the print statements.

- 3.) `except` – here Exception is automatically built by Python when an error occurs, now we put it as our variable “e”. Also notice that aside from the custom error message that is printed, we are also commanding it to `print(e.__doc__)` which prints the documentation string of the exception type and `print(type(e))` which prints the type of the exception object.
- 4.) `finally` – this block let’s you execute the code regardless of the result of the try-except block. We are commanding the script to close the file if file is not `file.closed`

```
/Users/terence/PycharmProjects/pythonProject/.venv/bin/python /Users/terence/PycharmProjects/pythonProject/Mod05-Lab03-WorkingWithExceptions.py
Text file must exist before running this script!
--Technical Error Message
[Errno 2] No such file or directory: 'MyLabData.jsonZZZZZZ'
File not found.
<class 'FileNotFoundError'
```

Figure 1.2 Visual representation of outcome of figure 1.1 try-except error handling

Personally, I like how the error is represented using the try-except handling that was implemented. In a way it looks less intimidating and it easier for the user to understand what went wrong and how to handle it.

Why do we use Try and Except?

There are numerous advantages of using try-except.

- This can improve your script by preventing crashing when unexpected errors occur.
- You can catch or anticipate specific exception to handle different errors in different ways.
- You can customize your error messages depending on the user making it more user-friendly
- It allows the program to recover from errors and continue executing, which is especially useful in applications that rely on user input or external resources.
- It can save you, the user, other developers so much time instead of attempting to debug the script.

```
elif menu_choice == "2":
    try:
        student_first_name=(input("What is the student's first name? "))
        if not student_first_name.isalpha():
            raise ValueError("The first name should not contain numbers.")
```

Figure 2.1 using try with an if not statement. The if not here uses `.isalpha()` – which is used to check if all value/s entered are alphabetic. If the any value or character is non-alphabetic then it will raise a `ValueError` with a custom message.

```
except ValueError as e:
    print(e)
    print("--Technical Error found--")
    print(e.__doc__)
    print(e.__str__())
```

Figure 2.2 except `ValueError`, this is what to expect when the error from figure 2.1 occurs

```
What is the student's first name? B9bby
The first name should not contain numbers.
--Technical Error found--
Inappropriate argument value (of correct type).
The first name should not contain numbers.
```

Figure 2.3 This is the output when an exception is met in figure 2.1/2.2 . As a user this was a very simple way to let me know the cause and how to resolve it.

Summary:

Try and Except has many significant advantages when utilized properly. I believe that this approach provides value to both the programmer and the user and when used optimally it can save you a great deal of time. This is definitely a practice that I can see myself using as we get further into the lessons.