nome : miguel sousa equipa : the bugs data 10/10/2025

Report: ICC-03 – Your First Python Script

Introduction

The goal of this task was to understand how Python works with files and learn to use the different file modes: r, w, a, and r+.
In cybersecurity, it is common to read logs, update configuration files, or store reports automatically.
Knowing how to use these modes allows us to automate these tasks safely and efficiently.
During the exercise, I created a simple Python menu that lets the user choose between reading a file, writing new content, appending text at the end, or reading and writing at the same time. This helped me see in practice how each mode works and when to use it.

Objectives

1. Understand how to open, read, and write files in Python.
2. Learn to create a functional menu using loops and conditional statements.
3. Implement file operations (r, w, a, r+) safely and effectively.
4. Apply exception handling (try / except) to avoid runtime errors.
5. Connect user input with automated file management tasks.

Development

1. File Handling in Python

Mode	Description	Example Use
`'r'`	Open a file for reading only.	Viewing system logs or configuration data.
`'w'`	Opens a file for writing and overwrites existing content.	Replacing outdated configuration files.
`'a'`	Opens a file for appending new data at the end.	Adding new alerts or log entries.
`'r+'`	Opens a file for both reading and writing.	Reviewing and updating files simultaneously.

The command **open("filename", "mode")** was used together with the with statement to ensure that files are automatically closed after use.

Example:

```
with open(FILENAME, "r") as file:

with open(FILENAME, "w") as file:

with open(FILENAME, "a") as file:

with open(FILENAME, "r+") as file:
```

2. Menu System

The script uses a while True loop to continuously display a menu with five options:

1. Read file (r)
2. Write new file (w)
3. Append content (a)
4. Read & Write (r+)
5. Exit

```
while True:
    print("\n=== FILE HANDLING MENU ===")
    print("1) Read file ('r')")
    print("2) Write new file ('w')")
    print("3) Append to file ('a')")
    print("4) Read & Write ('r+')")
    print("5) Exit")
```

https://jobbinge.in/menu-driven-programs-in-python/

---

3. Error Handling with `try` / `except`

To make the program robust, I implemented try and except blocks to handle predictable errors, especially when the file does not exist.

Example:

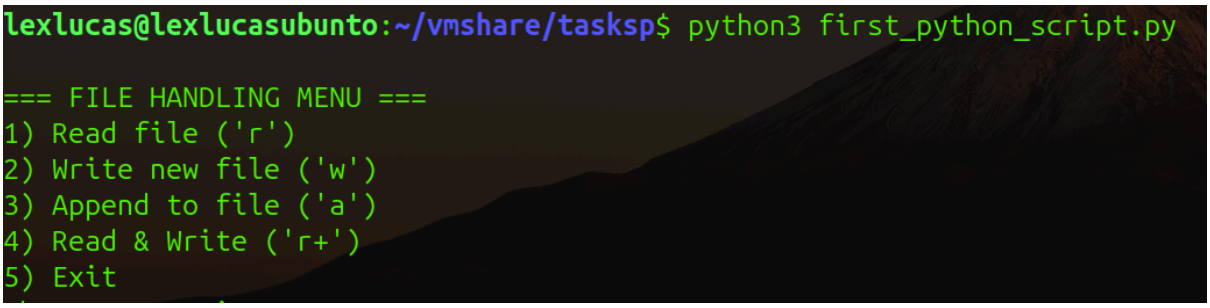
```
    if choice == "1":
        try:
            with open(FILENAME, "r") as file:
                content = file.read().strip()
                if content:
                    print("\n--- File Content ---")
                    print(content)
                    print("----------------------")
                else:
                    print("\nThe file is empty.")
        except FileNotFoundError:
            print("\n File not found. Create or write one first.")
```

https://stackoverflow.com/questions/16138232/is-it-a-good-practice-to-use-try-except-else-in-python

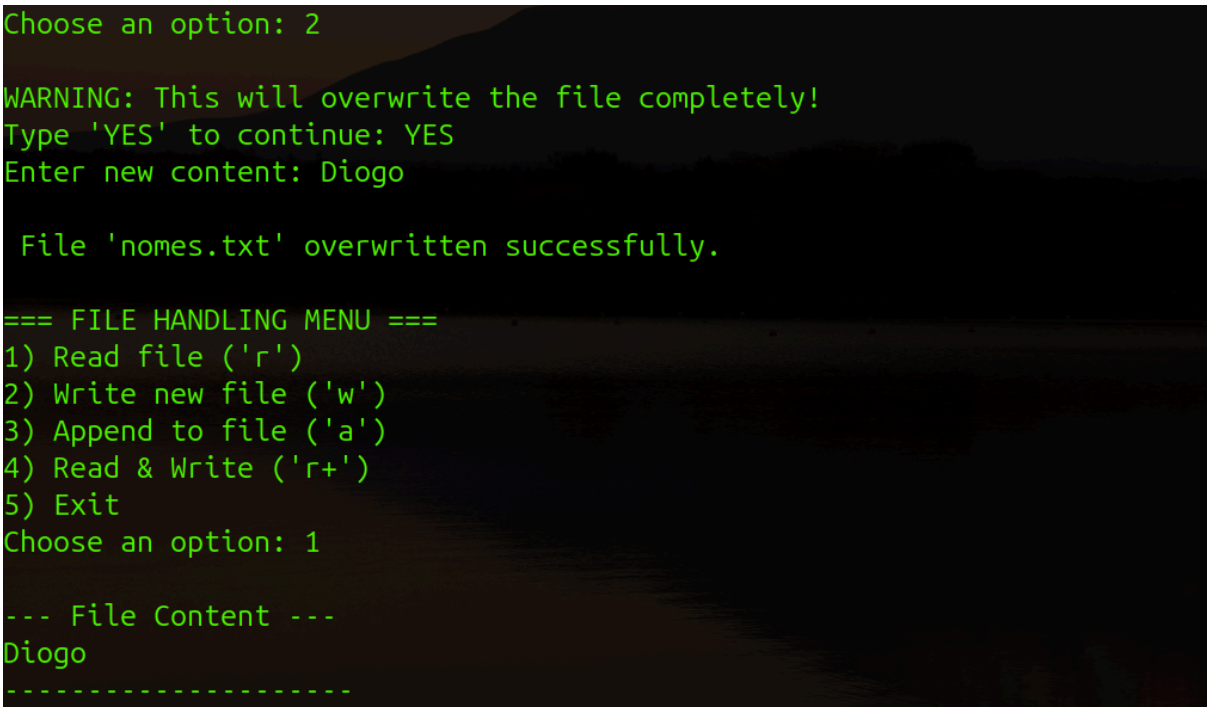
This ensures the program does not crash and provides a clear message to the user. In cybersecurity scripting, this technique is essential for preventing uncontrolled failures during automation.

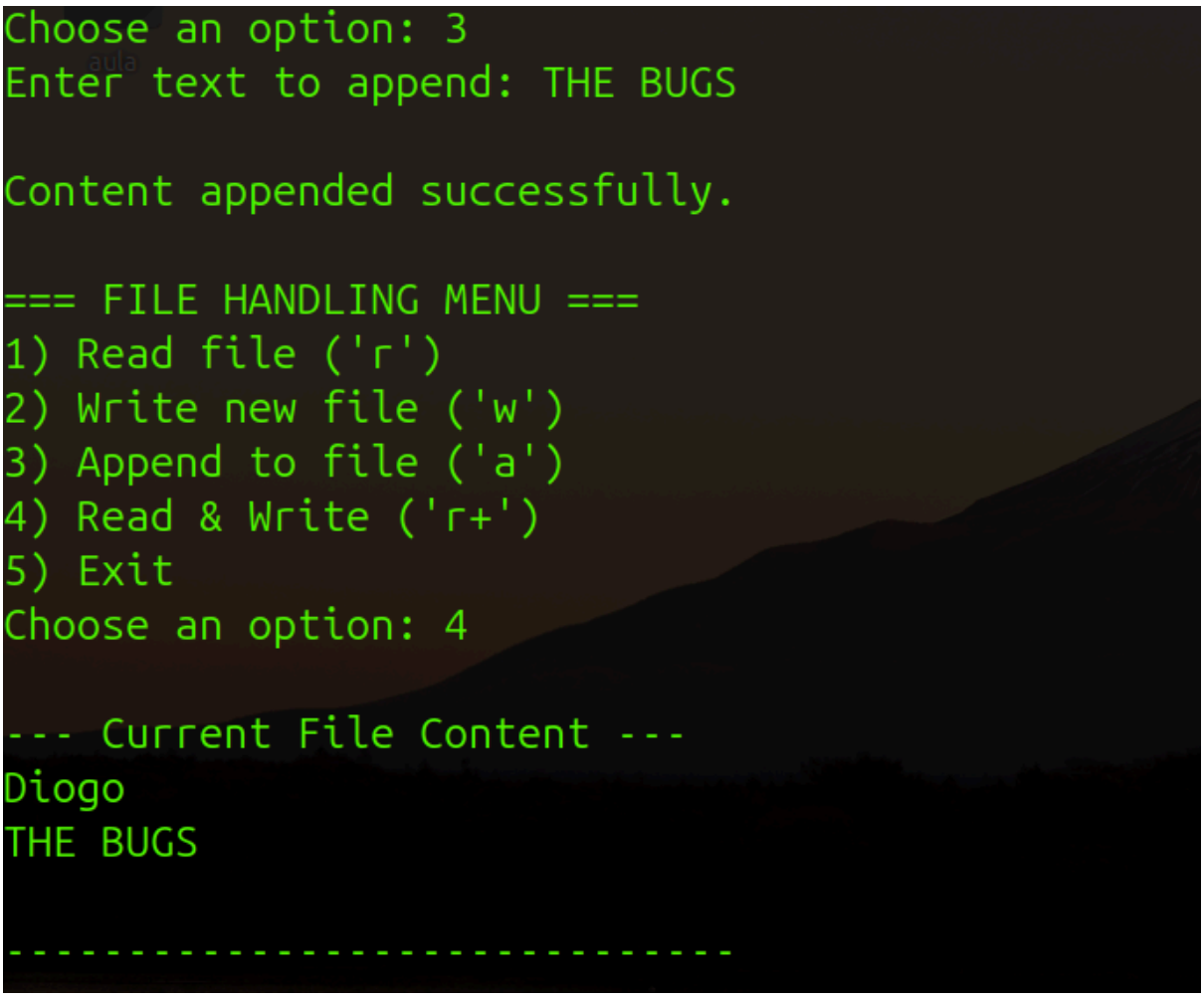
Results

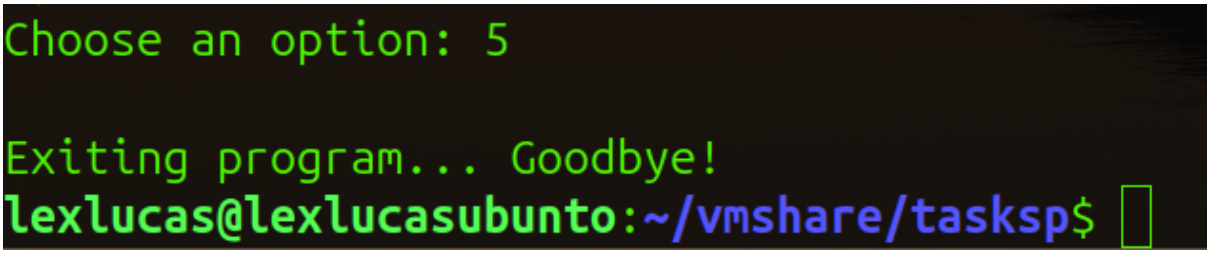
Displays a clear and functional menu.



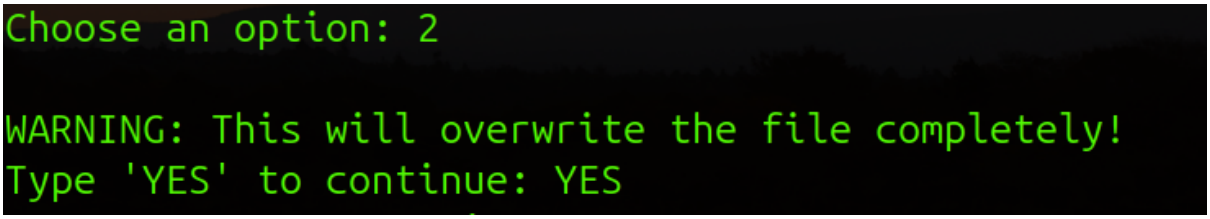
Executes file operations in all four modes.



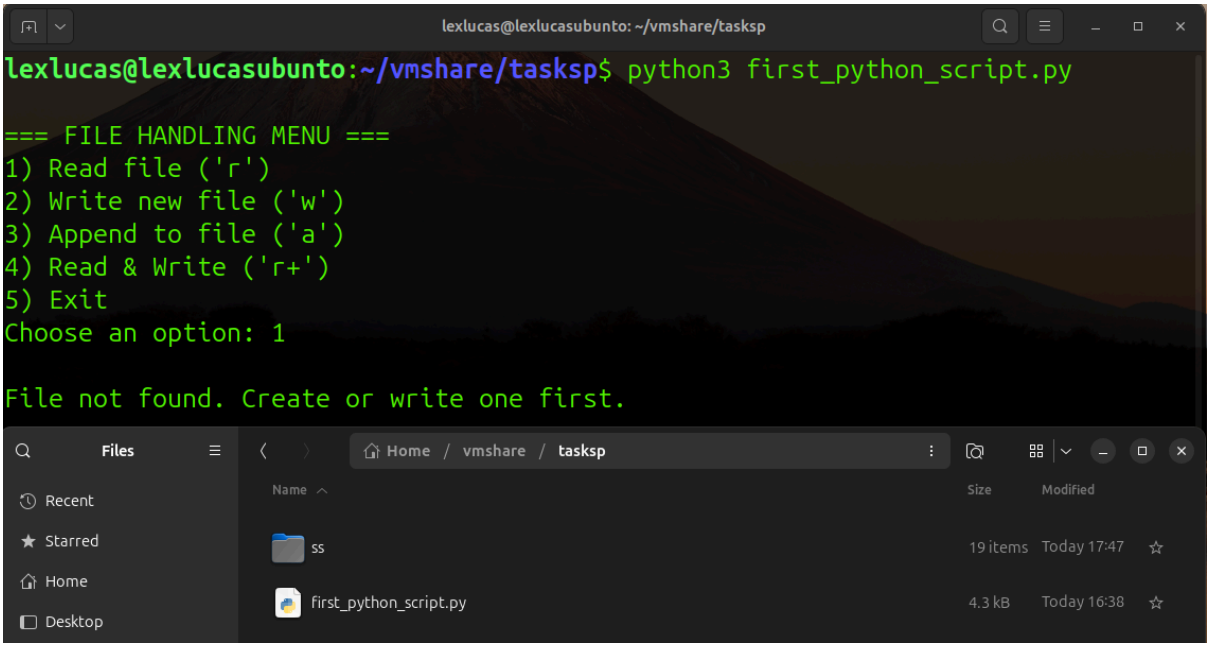




Handles user input safely.



Prevents crashes with proper error management.



Conclusion

This activity helped me gain a deeper understanding of Python and how it can be applied to solve problems in a logical, step-by-step way. Like the previous tasks, it combined practice with problem-solving, where I tested my skills and searched for reliable sources to support my code ideas.

I learned the practical differences between the main Python file modes:

r is used to read existing content.
w creates a new file or overwrites an existing one.

a adds new lines at the end of the file.
r+ allows reading and writing in the same file without deleting it.

I also learned how to use the **try** and **except** commands to handle possible errors when working with files, making the program more stable and preventing it from crashing.