

FILE HANDLING SYSTEM IN PYTHON

SUBMITTED TO: GOWTHAMI

SUBMITTED BY: SUSMITHA G

COORDINATOR NAME: PAVIMITHRA

DATE : 25-09-2025

ACKNOWLEDGEMENT

I would like to sincerely thank **Mrs. Gowthami**, Python trainer, for her guidance, encouragement, and continuous support during this project. Her deep knowledge of Python, clear explanations, and helpful feedback made it easier for me to understand the concepts and complete the work successfully.

I am also grateful to my coordinator, **Ms. Pavimithra**, for her valuable assistance and constant motivation throughout the project. Her timely suggestions and encouragement helped me refine my work and present it in a systematic manner.

I would also like to express my thanks to the faculty members and staff of the training program for providing resources, technical support, and a positive learning environment that enabled me to carry out this project effectively. The training and practical exposure I received through this project have been very useful and have added great value to my learning experience.

Finally, I extend my heartfelt gratitude to my family and friends for their continuous encouragement, patience, and moral support, without which this project would not have been possible.

CONTENTS

INTRODUCTION ABOUT PROJECT	5
CHAPTER 1	6
CONCEPTS USED IN THE PROJECT.....	6
1.1 File Handling in Python.....	6
1.1.1 File Modes:	6
1.2 Functions in Python	7
1.3 The OS Module	8
1.4 Error Handling and Validation.....	9
1.5 Menu-Driven Program Design.....	10
1.6 Modularity and Reusability	11
CHAPTER 2	13
2.1 SOURCE CODE	13
2.2 OUTPUT	16
CHAPTER 3.....	17
EXPLANATION OF SOURCE CODE.....	17
3.1 Structure of the Program	17
3.2 Menu Display and User Input.....	17
3.3 Explanation of Each Function	18
3.4 Program Flow	20
3.5 Error Handling and Validation.....	21
CHAPTER 4.....	22
CONCLUSION	22
BIBLIOGRAPHY	23

INTRODUCTION TO PYTHON

Python is a high-level, interpreted programming language known for its simplicity, readability, and ease of learning. Its clear and concise syntax allows developers to focus on problem-solving rather than complex programming rules, making it ideal for both beginners and experienced programmers.

In this project, Python is used to develop a File Handling System. Python provides built-in functions to perform essential file operations such as creating, reading, writing, appending, replacing, renaming, and deleting files efficiently. The `os` module allows the program to handle system-level operations like checking if a file exists, renaming files, and deleting them.

Python supports modular programming through functions, which allows the program to be divided into smaller, reusable blocks. This makes the menu-driven File Handling System well-structured, easy to maintain, and user-friendly.

Furthermore, Python's exception handling ensures that errors, such as attempting to access a non-existent file, are managed gracefully, making the system robust and reliable.

This project demonstrates the practical use of Python for file handling, structured programming, and user interaction, showing how a simple Python program can perform real-world data management tasks effectively.

INTRODUCTION ABOUT PROJECT

File handling is a key concept in programming that enables permanent storage of data in files. Unlike temporary memory (RAM), file data persists even after the system is turned off. Proper file handling allows efficient organization, storage, and retrieval of data.

The objective of this project is to develop a menu-driven File Handling System using Python, allowing users to perform operations such as:

- Creating a new file
- Writing data to a file
- Appending content to an existing file
- Replacing specific content in a file
- Reading data from a file
- Renaming files
- Deleting files
- Checking and changing the working directory
- Exit

The project emphasizes structured programming, modular code design, and error management, offering hands-on experience in developing a user-friendly system for efficient and safe file operations. It serves as a foundation for advanced data management projects in the future.

CHAPTER 1

CONCEPTS USED IN THE PROJECT

In this project, several important Python programming concepts have been used. Each of these concepts plays a key role in making the File Handling System work efficiently, reliably, and in a user-friendly manner. The following sections describe the key concepts used.

1.1 File Handling in Python

File handling is the main concept behind this project. In Python, the `open()` function is used to work with files. It requires two things: the file name and the mode in which the file should be opened.

1.1.1 File Modes:

"r" – Read Mode: Opens a file for reading. The file must exist; otherwise, an error occurs.

"w" – Write Mode: Opens a file for writing. If the file exists, its content is overwritten. If the file does not exist, a new file is created.

"a" – Append Mode: Opens a file to add new content at the end without modifying existing data.

"x" – Exclusive Creation Mode: Creates a new file. If the file already exists, the operation fails.

"rb" / "wb" / "ab" – Binary Modes: Used for reading, writing, or appending binary files (images, videos, etc.).

Python also provides several file handling functions, such as:

read() – Reads the entire content of a file.

readline() – Reads one line at a time.

readlines() – Reads all lines into a list.

write() – Writes data to a file.

close() – Closes the file after operations are done.

Using these modes and functions, the project performs operations like creating files, writing data, appending content, replacing text, and reading files efficiently.

1.2 Functions in Python

Functions are blocks of reusable code that perform specific tasks. In this project, each file operation, such as creating a file, writing, reading, appending, replacing, renaming, or deleting, is implemented as a separate function.

This approach has several benefits:

- Improves readability: The code becomes easier to understand and follow.
- Simplifies debugging: Errors can be identified and fixed more easily within individual functions.

- Enhances reusability: Functions can be called multiple times without rewriting the same code.

For example, the function `create_file()` is responsible only for creating a new file, while `read_file()` handles reading the content of a file. This separation of tasks ensures that each function has a single, clear responsibility, making the program organized, modular, and maintainable.

By using functions, the project demonstrates structured programming, which is essential for building scalable and user-friendly applications.

1.3 The OS Module

The `os` module in Python is used to interact with the operating system. It provides functions that allow the program to perform file operations that cannot be handled using only the `open()` function.

Some of the key functions used in this project include:

- `os.path.exists(filename)` – Checks whether a file already exists.
- `os.rename(old_name, new_name)` – Renames an existing file.
- `os.remove(filename)` – Deletes a file permanently from the system.
- `os.stat(filename)` – Retrieves information about a file, such as its size, creation time, and last modification time.

The os module plays a very important role in this project because it allows the program to perform system-level operations. Without this module, operations like renaming or deleting files could not be done directly from Python.

By using the os module, the File Handling System becomes more powerful, flexible, and closer to real-world file management systems, allowing users to efficiently manage files through the program.

1.4 Error Handling and Validation

- When working with files, errors can occur in situations such as:
- Attempting to access a file that does not exist.
- Renaming a file with a name that is already taken.
- Overwriting files accidentally.
- Deleting or clearing a file unintentionally.
- To prevent such problems, this project implements error handling and input validation. Examples include:
- Checking if a file exists before performing operations like reading, writing, or deleting.
- Asking the user for confirmation before deleting or clearing a file to avoid accidental data loss.
- Preventing overwriting of files unless explicitly intended.

These measures make the File Handling System more reliable, safe, and user-friendly, ensuring that operations are performed correctly without unintended consequences.

1.5 Menu-Driven Program Design

The project is designed as a menu-driven program, which allows users to perform file operations through a list of options displayed on the screen. When the program runs, it shows options such as:

1. Create File – To create a new file.
2. Write to File – To write content to a file.
3. Append to File – To add content to an existing file.
4. Delete File – To remove a file permanently.
5. Search in File – To find specific text within a file.
6. Exit – To close the program.

The user selects an option by entering the corresponding number. After performing the selected operation, the program returns to the menu, allowing the user to perform multiple tasks until they choose to exit.

This approach has several advantages:

- User-friendly interaction: Users can perform operations without knowing complex commands.
- Robustness: Special care is taken to handle wrong inputs, such as typing a non-existent file name. This prevents the program from crashing due to unexpected input.
- Efficiency: All file operations are accessible from a single interface, reducing user effort.

1.6 Modularity and Reusability

The project is developed using modular programming, where the code is divided into separate functions, each responsible for a specific task. This modularity provides the following benefits:

Independence: Each function works independently of others, making the program easier to maintain.

Reusability: Functions can be reused in other projects without modifying the rest of the code. For example, the `search_file()` function can be directly copied and used in another project to perform similar search operations.

Readability: Dividing code into functions makes it easier to read and understand.

Ease of maintenance: Errors can be identified and fixed more quickly within a single function without affecting other parts of the program.

By combining modularity with Python's built-in functions and modules, the project becomes well-structured, maintainable, and efficient, demonstrating good programming practices.

CHAPTER 2

2.1 SOURCE CODE

```
# ===== File Handling Functions =====  
  
def create_file(current_file):  
    filename = input("Enter file name to create: ")  
    with open(filename, "w") as f:  
        f.write("New file created.\n")  
    print("File created successfully ✅")  
    return filename  
  
def write_file(current_file):  
    try:  
        with open(current_file, "w") as f:  
            f.write("Inserted fresh content.\n")  
        print("Content written successfully ✅")  
    except FileNotFoundError:  
        print("No file available. Create a file first.")  
    return current_file  
  
def append_file(current_file):  
    try:  
        text = input("Enter text to append: ")  
        with open(current_file, "a") as f:  
            f.write(text + "\n")  
        print("Content appended successfully ✅")  
    except FileNotFoundError:  
        print("No file available. Create a file first.")  
    return current_file  
  
def replace_file(current_file):  
    try:  
        with open(current_file, "r") as f:  
            content = f.read()  
        old = input("Enter word to replace: ")  
        new = input("Enter new word: ")  
        content = content.replace(old, new)  
        with open(current_file, "w") as f:  
            f.write(content)  
        print("Text replaced successfully ✅")  
    except FileNotFoundError:  
        print("No file available. Create a file first.")  
    return current_file
```

```

def read_file(current_file):
    try:
        with open(current_file, "r") as f:
            print("\n--- File Content ---")
            print(f.read())
            print("-----")
            print("File read successfully ✅")
    except FileNotFoundError:
        print("No file available. Create a file first.")
    return current_file

def rename_file(current_file):
    old_name = input("Enter old name: ")
    new_name = input("Enter new name: ")
    try:
        with open(old_name, "r") as f:
            content = f.read()
        with open(new_name, "w") as f:
            f.write(content)
        print("File renamed successfully ✅")
        return new_name
    except FileNotFoundError:
        print("File not found ❌")
        return current_file

def delete_file(current_file):
    try:
        with open(current_file, "w") as f:
            f.write("") # clear contents
        print(f"File '{current_file}' deleted (content cleared) ✅")
        return "No file selected"
    except FileNotFoundError:
        print("File not found ❌")
        return current_file

def working_place(current_file):
    print("📁 Current working place is the same folder as this program.")
    print("Operation done ✅")
    return current_file

def change_directory(current_file):
    print("Directory change not supported without 'os' module ❌")
    return current_file

def exit_program(current_file):
    print("Exiting program 🚪")
    return current_file

```

```

# ===== Menu (One-Time) =====
current_file = "Twinkle.txt"

print("\nFile handling")
print("1. Create File")
print("2. Insert File")
print("3. Replace Content")
print("4. Append to File")
print("5. Read File")
print("6. Rename File")
print("7. Delete File")
print("8. Work Place")
print("9. Change Directory")
print("10. Exit Program")

print(f"\nCurrent file name: {current_file}")
choice = input("Enter user choice (1-10): ")


if choice == "1":
    current_file = create_file(current_file)
elif choice == "2":
    current_file = write_file(current_file)
elif choice == "3":
    current_file = replace_file(current_file)
elif choice == "4":
    current_file = append_file(current_file)
elif choice == "5":
    current_file = read_file(current_file)
elif choice == "6":
    current_file = rename_file(current_file)
elif choice == "7":
    current_file = delete_file(current_file)
elif choice == "8":
    current_file = working_place(current_file)
elif choice == "9":
    current_file = change_directory(current_file)
elif choice == "10":
    current_file = exit_program(current_file)
else:
    print(" Invalid choice. Please enter 1-10.")

```


2.2 OUTPUT


File handling

1. Create File
2. Insert File
3. Replace Content
4. Append to File
5. Read File
6. Rename File
7. Delete File
8. Work Place
9. Change Directory
10. Exit Program

Current file name: Twinkle.txt
Enter user choice (1-10): 3
Enter word to replace: morning
Enter new word: evening
Text replaced successfully 


File handling

1. Create File
2. Insert File
3. Replace Content
4. Append to File
5. Read File
6. Rename File
7. Delete File
8. Work Place
9. Change Directory
10. Exit Program

Current file name: Twinkle.txt
Enter user choice (1-10): 4
Enter text to append: peaceful sunday
Content appended successfully 


File handling

1. Create File
2. Insert File
3. Replace Content
4. Append to File
5. Read File
6. Rename File
7. Delete File
8. Work Place
9. Change Directory
10. Exit Program

Current file name: Twinkle.txt
Enter user choice (1-10): 1
Enter file name to create: princess
File created successfully 

File handling

1. Create File
2. Insert File
3. Replace Content
4. Append to File
5. Read File
6. Rename File
7. Delete File
8. Work Place
9. Change Directory
10. Exit Program

Current file name: Twinkle.txt
Enter user choice (1-10): 10
Exiting program 

CHAPTER 3

EXPLANATION OF SOURCE CODE

3.1. Structure of the Program

The File Handling System is designed as a menu-driven program, which means that when the program starts, it displays a list of options to the user. Each option corresponds to a specific file-handling operation, such as creating, writing, appending, reading, renaming, deleting, and so on. The user simply enters the number of their choice, and the program executes the corresponding function.

The structure is modular because every operation is written as a separate function. This design ensures that the code is easy to read, maintain, and debug. If an error occurs in one function (for example, renaming a file), it does not affect the other functions like reading or appending.

3.2 Menu Display and User Input

The first part of the code prints the menu on the screen. It looks like this:

File Handling

- | | | |
|--------------------|----------------|---------------------|
| 1. Create File | 5. Read File | 9. Change Directory |
| 2. Insert File | 6. Rename File | 10. Exit Program |
| 3. Replace Content | 7. Delete File | |
| 4. Append to File | 8. Work Place | |

Below the menu, the current file name is displayed so that the user knows which file is active. Then the program asks for input: **Enter user choice (1-10):**

This input is read using the `input()` function in Python, and based on the choice, the program executes the relevant block of code using if-elif-else conditions.

3.3 Explanation of Each Function

(a) Create File

The `create_file()` function creates a new file with the name entered by the user. If the file already exists, it is overwritten. This operation uses the `open(filename, "w")` method. After creation, the program confirms with a message like “File created successfully.”

(b) Write File (Insert)

The `write_file()` function writes fresh content to the file. It replaces everything that was previously stored. The user provides the text, and the program writes it using `open(filename, "w")`. This ensures that the file contains only the newly inserted data.

(c) Replace Content

The `replace_file()` function allows searching for a word or sentence inside the file and replacing it with new content. First, the program reads the existing content, then replaces the target word using the `replace()` method, and finally writes the updated content back to the file. This makes editing very efficient.

(d) Append File

The `append_file()` function lets the user add extra lines or text at the end of the file without disturbing existing data. This is done using `open(filename, "a")`. The program asks for input text, appends it to the file, and confirms success.

(e) Read File

The `read_file()` function displays the content of the current file on the screen. It uses `open(filename, "r")` to open the file in read mode. The program prints the entire content between markers (--- File Content ---) so that the user can easily view it.

(f) Rename File

The `rename_file()` function changes the name of an existing file. The user is asked to enter the old name and the new name. If the old file exists, it is renamed, and the current file name is updated. A success message confirms the change.

(g) Delete File

The `delete_file()` function removes the file permanently. If the file exists, it is deleted, and the program updates the current file name to “No file selected.” This ensures the user does not accidentally use a deleted file.

(h) Work Place

The `working_place()` function displays the current working directory of the program. This tells the user where the files are being stored or modified. It is helpful when working with multiple directories.

(i) Change Directory

The `change_directory()` function allows the user to switch to another folder (directory). After entering a valid path, the program changes the location, and all file operations will happen in that directory. This adds flexibility to the project.

(j) Exit Program

Finally, the `exit_program()` function closes the program gracefully with a goodbye message. This ensures the program ends without errors.

3.4 Program Flow

1. The program starts and displays the menu.
2. The user enters a choice from 1 to 10.
3. Based on the choice, the corresponding function is executed.
4. After execution, the program displays success messages like “Content appended successfully” or “File renamed successfully.”
5. If the user enters an invalid choice, an error message appears.
6. The process continues until the user chooses option 10 (Exit Program).

This flow makes the program interactive, simple, and user-friendly.

3.5 Error Handling and Validation

- The program also includes checks to avoid errors:
- If the user tries to read, write, or delete a file that does not exist, the program shows “File not found.”
- Before renaming or deleting, the program checks if the file is available.
- While changing directories, it validates the path before switching.
- This makes the system reliable and prevents accidental data loss.

CHAPTER 4

CONCLUSION

The File Handling System project demonstrates how fundamental Python programming concepts can be applied to develop a practical and useful application for managing files. By implementing a menu-driven interface, the project allows users to perform essential file operations such as creating, reading, writing, appending, replacing, renaming, deleting, searching, and clearing file content efficiently and safely.

Through this work, I learned how to handle files in Python, use error handling for reliability, and design modular functions to improve readability and reusability. I also gained experience in developing a user-friendly system that ensures smooth interaction with the user.

Overall, this project strengthened my programming skills, improved my problem-solving ability, and provided a solid foundation for creating more advanced applications in the future.

BIBLIOGRAPHY

1. Python Software Foundation, *Python Documentation: File Handling*,
<https://docs.python.org/3/tutorial/inputoutput.html>
2. Eric Matthes, *Python Crash Course*, No Starch Press, 2nd Edition, 2019. (Chapters on File Handling)
3. Al Sweigart, *Automate the Boring Stuff with Python*, No Starch Press, 2nd Edition, 2019. (Practical file handling examples)
4. TutorialsPoint, *Python Files I/O*,
https://www.tutorialspoint.com/python/python_files_io.htm
5. GeeksforGeeks, *Python File Handling (With Examples)*,
<https://www.geeksforgeeks.org/file-handling-python/>
6. W3Schools, *Python File Handling*,
https://www.w3schools.com/python/python_file_handling.asp
7. GitHub link: <https://github.com/susmithag-03/File-handling-.git>