Degree Project:

File Handling Script in Python



Björn Lagerblad

NBI/Handelsakademin

File Handling Script in Python

2025/02

# 

# Abstract

I chose to write this essay in English so that any future employers or curious visitors to my GitHub can easily understand its purpose.

This project focuses on the development of a Python script to automate file organization using **pathlib, shutil** and **tkinter**.

The script identifies files in a given directory, categorizes them based on file type, and moves them into designated folders for better organization. This automation reduces manual effort and improves overall efficiency in managing files. The script was tested across multiple environments, including Windows, Linux, and Raspberry Pi, ensuring cross-platform compatibility. Future improvements include the addition of a graphical user interface (**GUI**) to further enhance user experience.

To ensure clarity and correctness, I used ChatGPT to proofread and refine the English wording of this report, as I’m eager to explore new tools and I see it as an excellent way to test the LLM/Human integration of language barriers and terminology.

# Abbreviations and Terminology

* **POC (Proof of Concept)** – A small-scale implementation to verify if an idea works before full development.
* **GUI (Graphical User Interface)** – A visual interface that allows users to interact with software using buttons, menus, and other graphical elements instead of the command line.
* **OS (Operating System)** – The software that manages hardware and software resources on a computer (e.g. Windows, Linux, macOS).
* **Git** – A version control system used to track changes in code and collaborate with others.
* **Pseudocode** – A rough, human-readable outline of how a program will work before actual coding begins.
* **venv (Virtual Environment)** – An isolated Python environment used to manage dependencies and prevent conflicts between projects.
* **Script** – A file containing a sequence of commands written in a programming language (in this case, Python).
* **PyInstaller** – A Python tool used to convert scripts into standalone executable files (e.g., .exe for Windows).
* **Python** – A high-level programming language known for its readability and versatility, used for this project.
* **pathlib –** A **Python** library that provides an object-oriented approach to handling file system paths, making it easier to work with file locations across different operating systems.
* **shutil –** A high-level utility module in **Python** that enables file operations such as copying, moving, and deleting files and directories.
* **tkinter –** Python’s built-in library for creating graphical user interfaces (GUIs).
* **Executable (.exe)** – A standalone program file that runs on Windows without requiring additional dependencies.
* **file\_sorter.py** – The main Python script developed in this project, responsible for sorting and organizing files.
* **Caffeine** – A critical development tool that powers long coding sessions.
* **Bug** – An unexpected error or issue in the program that needs to be fixed.

Table of Contents

[1 Abstract 2](#_Toc191568737)

[2 Abbreviations and Terminology 3](#_Toc191568738)

[3 Introduction 1](#_Toc191568739)

[4 Theory 2](#_Toc191568740)

[4.1 Cross Platform File Systems 2](#_Toc191568741)

[4.2 Python’s File Management Approach 2](#_Toc191568742)

[4.3 GUI Theory and Principles 3](#_Toc191568743)

[4.4 Error Handling & Robustness 3](#_Toc191568744)

[4.5 Automation & Productivity 3](#_Toc191568745)

[4.6 Summary 4](#_Toc191568746)

[5 Method 5](#_Toc191568747)

[5.1 Initial Setup 5](#_Toc191568748)

[5.2 Research and Warm-Up Exercises 5](#_Toc191568749)

[5.3 Designing the Workflow 6](#_Toc191568750)

[5.4 Iterative Development and Refinement 6](#_Toc191568751)

[5.5 Testing and Feedback 7](#_Toc191568752)

[5.6 Future Enhancements after the First Stage 7](#_Toc191568753)

[5.7 Summary of the First Stage 8](#_Toc191568754)

[5.8 Proof of Concept First Stage 8](#_Toc191568755)

[5.9 Second Stage Implementing a GUI and creating an .exe 11](#_Toc191568756)

[6 Results and Discussion 13](#_Toc191568757)

[6.1 Evaluating Functionality 13](#_Toc191568758)

[6.2 Challenges and How They Were Addressed 14](#_Toc191568759)

[6.3 User Feedback and Testing Results 15](#_Toc191568760)

[7 Conclusions 16](#_Toc191568761)

[7.1 GUI Implementation Overview 17](#_Toc191568762)

[8 References 19](#_Toc191568763)

# Introduction

I found it prudent to create something simpler yet with a new challenge: a file organizer that works across different operating systems and includes a functional GUI with an executable file.

During larger projects, I noticed that many of my classmates(including myself) have massive download folders with scattered files, resulting in an unorganized mess that makes efficiency worse.

Automating this process can (hopefully) help future students keep their files in order, and it will also serve as a good example of thorough code documentation.

This project focuses on developing a Python script to automate file organization. The main goals of the script are to:

* Identify files in a folder
* Sort them based on file type (e.g., images, documents, videos)
* Move them to specific directories for better organization
* Implement a working GUI with robust error handling
* Make a working executable
* Ensure compatibility across multiple operating systems

The script is built using **Python 3.11** and its standard libraries, particularly **pathlib** for file handling. If time permits, a **Graphical User Interface (GUI)** using **tkinter** or another **Python** framework will be added to improve usability.

Ultimately, the aim is to challenge myself with something new (**GUI** handling in this case) in **Python** while documenting every step of the process, from planning and coding to testing and analysis.

# Theory

In this section, I’ll explain the main ideas behind how this project will work. First, we’ll look at how different **operating systems** handle files and why that matters between the different **Operating Systems** today. Then, I’ll discuss Python’s built-in tools—like **pathlib** and **shutil**—that works such as moving, copying, and deleting files without having to worry about path separators or manual string manipulation. Finally, I’ll give a quick overview of how a **GUI** deals with user interactions, making the program more accessible for people who don’t know terminal commands.

## Cross Platform File Systems

Operating systems like **Windows**, **Linux**, and **macOS** (to name the most popular ones) differ in how they manage file paths, directory structures, and permissions some examples are.

* **Windows:** Commonly uses backslashes(\) in file paths (e.g. C:\Users\Documents), and file names are often case-insensitive.
* **Linux/macOS:** Use forward slashes (/) in file paths (e.g., /home/user/Documents), and typically have case-sensitive file systems.
* **Permissions:** Each system has its own approach to user permissions, read/write access, and hidden files.

For a file organizer to be truly cross-platform, it must handle these variables perfect, or we will get more errors than a new computer in a senior home. By researching libraries and patterns that works away these differences, we can find that some libraries in **python** will function with that goal in mind.

## Python’s File Management Approach

Python provides multiple ways to interact with files and directories. Historically, the **OS** module (and **OS** path) was the standard tool, meaning you had to manually handle paths in your code. However, modern **Python** has introduced more user-friendly options that simplify these tasks.:

* **pathlib:**  
  Offers an object-oriented approach to file paths, simplifying code and improving readability. It automatically manages path separators, so the same code works on Windows, Linux, and macOS without manual string manipulation. A great tool with our cross platform goal in mind.
* **shutil:**  
  Provides high-level file operations such as copying, moving, and deleting directories. It complements **pathlib** by handling tasks that go beyond simple path manipulation, aka we have the tool that works in tandem with **pathlib**.

## GUI Theory and Principles

A **GUI** allows users to interact with an application through visual elements like buttons, menus, and dialog boxes, rather than the command line. In **Python**, **tkinter** is a common library for building simple GUIs:

* **The User Interface:**
* **GUI**s rely on events (e.g buttons, key presses and so on from user input) something I have not worked on before and I am approaching this challenge with an open mind.
* **User-Friendliness:**  
  A **GUI** needs to be as straightforward as possible, so I plan to test it with friends who have minimal technical experience some who admit they struggle even with microwaves. If they can navigate the interface easily, then it’s truly user-friendly.

## Error Handling & Robustness

When working with file operations, robust error handling is essential to prevent data loss or application crashes. Potential issues include:

* **Permission Errors:**  
  Attempts to move or delete files without the necessary read/write permissions.
* **Locked Files:**  
  Some files may be open in other programs, preventing modifications.
* **Invalid Paths:**  
  Missing or broken directory references, especially on systems where a path may not exist or has changed.
* **Different OS Handling:**  
  Different Operating Systems have different sets of rules for file permissions, hidden files and so on, I think this will be the biggest hurdle to get over.

By using exception handling and meaningful error messages, I can help users from clearly know what the issue is so they know what the problem is.

## Automation & Productivity

Manual file sorting can be tedious and prone to human error, especially in environments with high file turnover (e.g. large download folders for teaching a neural network). Automating these processes will in turn:

* **Save Time:**  
  Routinely using this program will save a lot of headaches in not needing to manual sort the files.
* **Reduces Errors:**  
  Consistently using the same script will decrease human error and minimize misplaced files.
* **Improves Organization:**  
  Folders and subfolders stay orderly, making it easier to locate and manage files later.

## Summary

By learning about cross-platform file systems, exploring Python’s powerful libraries, and planning a user-friendly GUI with a rudimentary error handling, I’ve laid out a clear foundation for this file organizer. The concepts I covered ranging from path abstractions with **pathlib** and **shutil** to the benefits of automation and with what I need to have in mind starting the project.

# Method

I approached this project in several iterative steps, allowing me to gradually build up from basic concepts to a fully functioning file organizer, I’ve documented my journey in bulletin points here.

## Initial Setup

I started by setting up a **Git** repository to ensure that every change was tracked or if I needed to make a rollback. I created a **venv** using **Python** to isolate my project dependencies from the rest of my system, a good rule when especially when it comes to working in Python. Once the environment was ready, I installed the necessary libraries and froze my requirements in a **requirements.txt** file, a crucial step if I need to reinstall something later down the line or share it with others.

## Research and Warm-Up Exercises

Before diving into the full implementation, I needed to deepen my understanding and experience with our main libraries **pathlib** and **shutil**. Although the official documentation for these libraries is robust, I felt it better to ask GPT for some warm-up exercises. These exercises was simple, such as moving a file or creating a directory, which helped me gain confidence in using these modules.

For instance, I practiced by writing small snippets to:

* Read and list all files in a directory.
* Create a new folder using Path.mkdir().
* Move files from one directory to another using shutil.move().

With these warmup exercises I felt more comfortable to start the project in full.

## Designing the Workflow

Once I felt comfortable with the basics, I moved on to designing the overall workflow for the file organizer. I began by drafting **pseudocode** to outline my ideas. My initial **pseudocode** included steps for:

* Identifying files in a target directory.
* Determining each file’s type by checking its extension.
* Sorting the files into corresponding folders (e.g., Images, Documents, Videos).
* Handling files with unknown extensions by placing them into a “Misc” folder.
* Eventually, incorporating a **GUI** that allows users to select folders more easily.

Since I am less experienced with **GUI** development, that part of the project was only roughly outlined at this stage, with the main focus being on the file sorting logic.

## Iterative Development and Refinement

Armed with my **pseudocode** and more **caffeine** in my system that would probably kill a smaller elephant, I implemented a basic version of the file organizer. In the first iteration, the script successfully moved files into pre-created folders. This version was purely based on command lines in the terminal but helped me verify that the core worked as expected.

After confirming that the script could sort files correctly, I modified the sorting by modifying the folder setup process. Instead of relying on pre-existing folders, I updated the script to automatically create the necessary folders if they didn’t exist and give a print if they were already created.

This change made the program more user friendly and autonomous, reducing the manual setup required before running the script.

I also incorporated exception handling to deal with common issues:

* **Permission Errors:** If a file could not be moved due to insufficient permissions, the script now catches this exception and prints an informative error message.
* **Duplicate Files:** The script checks if a file with the same name already exists in the target folder and skips the move operation to prevent overwriting.
* **Skipping Critical Files:** To ensure that the script itself and essential files (like a requirements file) are not moved, I added specific conditions to skip them.

Throughout the development process, I made frequent commits to **Git**, committing each significant change and pushing them to my main branch.

## Testing and Feedback

After implementing the core functionality, I conducted extensive testing:

* I created a test directory filled with a variety of file types, including images, documents, videos, and miscellaneous files.
* I introduced dummy cases, such as files without extensions, duplicate files, and files that were open in other applications, to see how the script handled these situations.
* As I am still a junior and not always the best at commenting my own code, I’ve asked GPT to help me with comments on my code as to not confuse future visitors on my program.

Feedback from these tests were a learning experience. I ran the script in a controlled environment and checked my console output to ensure that each action (e.g. folder creation, file movement, error handling) was logged clearly and that no unseen errors would come to fruition. These steps confirmed that the script behaved as expected and helped me identify areas for further improvement.

## Future Enhancements after the First Stage

Although the current version of the file organizer is functional and meets the primary goal of sorting files, I have identified several areas for future development:

* **GUI Integration:** I plan to develop a user-friendly interface using **tkinter** to allow users to select directories and view progress in real time.
* **Advanced Sorting Rules:** In addition to sorting by file extension, I would like to implement more sophisticated categorization (e.g. grouping files by date or size).
* **Secure a new caffeine supplier:** My **caffeine** reserves have reached dangerously low levels, and immediate resupply is critical.

## Summary of the First Stage

In summary, my method in the first stage of development involved a structured, iterative approach:

* Setting up the project environment with **Git**, a **virtual environment**, and dependency management.
* Researching and practicing basic file operations with **pathlib** and **shutil** through warmup exercises.
* Drafting and refining **pseudocode** to outline the file organizer’s workflow.
* Implementing the script in iterative stages, beginning with basic file movement and progressing to automated folder creation and a more robust handling.

This approach ensures that the final product is not only functional but also well-documented, maintainable, and prepared for future upgrades such as GUI integration.

## Proof of Concept First Stage

I will visually document how the script works so far, I will include a series of screenshots demonstrating its key features and execution. These images will illustrate:

* The initial state of an unorganized folder.
* Running the script and its output in the terminal.
* The resulting organized folder structure after execution.
* Any key error handling messages or user interactions.

This section serves as so called **POC**, confirming that the script performs as intended and successfully automates file organization.

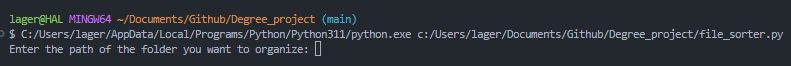
**Initial stage:**

**A screenshot of a computer

AI-generated content may be incorrect.**

In the picture above we can see that we have some dummy files with different extensions, do note the copy of **kittenjpg.jpg** one in the Images folder and outside of it. This will help demonstrate how the script handles duplicate files during the sorting process.

**Running the script:**

****

When running the **script**, it prompts the user to enter a path for the directory to organize. If the user simply presses Enter, the **script** will automatically organize files within the same folder where the **script** is located. Alternatively, the user can specify a different path, such as “Documents”, to organize files in another directory.

**A screenshot of a computer

AI-generated content may be incorrect.** **A screenshot of a computer

AI-generated content may be incorrect.**

For the purpose of this exercise I have deleted the “Documents” folder to show what happens when a a folder gets created or already exits.

**A screenshot of a computer screen

AI-generated content may be incorrect.**

And as we can see, existing folders are not recreated, but since the “Documents” folder was deleted earlier, the **script** automatically generates it again. The files are then sorted into their appropriate folders as intended, additionally as previously noted I have added some exclusions e.g. **file\_sorter.py** and the extra **kittenjpg.jpg** file was not moved because a file with the same name already exists in the ” Images” folder. This prevents accidental overwriting and ensures that duplicate files remain in their original location.

**Error Handling:**

**A computer screen shot of a computer program

AI-generated content may be incorrect.** ****

Since the script's primary function is to move files into their designated folders, error handling has been relatively straightforward. If a file's extension is not listed in the predefined folder structure, it is placed in the "Misc" folder. Additionally, if the specified target folder does not exist, the script exits with an error message.

## Second Stage Implementing a GUI and creating an .exe

With the core functionality of the script working as intended, the next step is to improve user experience by integrating **a GUI** and making the program executable as a standalone **.exe** file.

While the command-line version functions well, it requires users to manually enter folder paths, which might not work well for a regular user. **A** **GUI** will provide a more accessible and user-friendly way to interact with the program.

**Objectives of the GUI Implementation and .exe file:**

* Allowing users to **select a directory** via a file explorer instead of manually typing a path.
* Displaying a **progress log** so users can see what the script is doing in real time.
* Adding **error messages and warnings** in a clear and structured way.
* **Use pyinstaller** to package the script into an **executable.**

For this project, I decided to use **tkinter**, Python’s built-in **GUI** library. **tkinter** is lightweight, easy to implement, and does not require additional dependencies, making it an ideal choice for this task.

**Development Approach:**

1. **Designing the Interface:**

* A simple window with a **"Browse"** button for selecting a folder.
* A **"Start Sorting"** button to trigger file organization.
* A text box or list that logs file movements and status updates.

1. **Integrating the GUI with the Sorting Logic:**

* The script should take the selected folder as input instead of requiring manual path entry.
* The sorting function should be called when the user clicks the **"Start Sorting"** button.

1. **Error Handling and User Feedback:**

* Display warning messages for missing or invalid directories.
* Show a success message once sorting is complete.
* Create an undo button so its easier to test.

1. **Testing and Adjustments:**

* Testing the GUI on different resolutions and operating systems.
* Gathering feedback from non-technical users to refine usability.
* Adding an undo button to make it easier for me to **bug** test.

1. **Creating an .exe File:**

* Use **pyinstaller** to bundle the script into a Windows **executable** file, ensuring it runs without requiring **Python**.

This second stage shifts the project from being purely functional to being more **user-friendly**, making it accessible to a wider and less technical audience. With a working GUI, the program will require minimal technical knowledge to use, making file organization more efficient and intuitive.

I will note my progress in this in chapter **6 Results and Discussion.**

# Results and Discussion

With the second stage of the project implemented, this section will evaluate the results of the file organizer, analyse its effectiveness, and discuss improvements based on testing. The discussion will also include reflections on the challenges faced, how they were resolved, and what can be improved in future iterations.

## Evaluating Functionality

The script was tested under different conditions to ensure it correctly sorts and organizes files while maintaining stability. The key functionalities that were evaluated include:

* **Sorting Accuracy:** Files were successfully identified based on their extensions and moved to the appropriate folders. The “Misc” folder correctly handled unknown file types.
* **Folder Creation:** If a required folder did not exist, it was automatically created without user intervention.
* **Duplicate Handling:** If a file already existed in the target folder, the script skipped it to prevent overwriting.
* **Error Handling:** The script displayed meaningful error messages for permission issues and missing directories.
* **GUI Integration:** The **tkinter** interface allowed for easy selection of directories and provided real-time feedback on sorting progress.
* **Executable Packaging:** The script was successfully converted into a standalone .exe file using **pyinstaller**, making it accessible without requiring Python installation.

**Limitations:**  
Due to time constraints, I was unable to test the script on **Linux and Raspberry Pi** as originally planned. While the use of **pathlib** should theoretically allow for cross-platform functionality, **pyinstaller** only works for the OS it gets created in.

## Challenges and How They Were Addressed

**During development, several challenges arose that needed to be solved:**

| **Challenge:** | **Solution:** |
| --- | --- |
| Handling different **OS** path structures | Used **pathlib** for cross-platform compatibility |
| Preventing accidental overwrites | Script checks for existing files before moving |
| **GUI** responsiveness | Kept it lightweight with **tkinter** to ensure smooth performance |
| Packaging into an **.exe** | Used **pyinstaller** for standalone execution without dependencies |
| Limited testing on Linux/Raspberry Pi | Hard to implement from my home without destroying my structure on those systems. |
| Sort from size and date. | Due to time constraints I left this to future implementation. |

These solutions ensured that the script performed reliably under Windows testing conditions, though additional testing on other operating systems remains a potential area for improvement.

## User Feedback and Testing Results

To evaluate usability, I conducted tests with friends and coworkers, some of whom have a strict “do not approach” policy when it comes to important systems, as technology tends to blue screen in their mere presence.

The key feedback points included:

* **Positive Feedback:**
  + The **GUI** made it significantly easier to use than the command-line version.
  + The automatic **folder creation** was helpful and prevented setup confusion.
  + The **undo button** made testing easier and reduced the risk of misplaced files.
* **Improvement Areas:**
  + Some users requested a **progress bar** instead of just a log to visually track sorting.
  + A few users suggested **file previews** before sorting, so they could confirm the organization before committing to the changes.
  + Adding **custom sorting rules** (e.g. sorting by date or size) would make the tool more flexible.

# Conclusions

**In summary, the file organizer successfully met most of its original objectives:**

* Automated file sorting worked as intended.
* GUI integration improved accessibility for non-technical users.
* Error handling enhanced stability and user experience.
* The .exe file provided a standalone Windows solution.
* Testing on Linux and Raspberry Pi was not completed due to time constraints.
* Sorting by size and date was not implemented due to time constraints.
* **Future Improvements:**
* Adding a progress bar for better visual feedback.
* Implementing advanced sorting rules beyond file extensions (e.g., size, date).
* Completing Linux and Raspberry Pi testing to ensure full cross-platform compatibility.
* Exploring multi-threading to speed up sorting for large file sets.

**Overall, the project successfully demonstrated the benefits of automation, improved usability through a GUI, and executable packaging for broader accessibility.**

## GUI Implementation Overview

**Below are some screenshots showcasing the final GUI and its functionality:**

**Start:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Successful sorting:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Undo:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Undo Failed(If theres nothing to Undo):**

**A yellow triangle with black text

AI-generated content may be incorrect.**

# References

**Online Documentation & Libraries**

* Python Documentation. (n.d.). *pathlib — Object-oriented filesystem paths*. Python.org. Retrieved from <https://docs.python.org/3/library/pathlib.html>
* Python Documentation. (n.d.). *shutil — High-level file operations*. Python.org. Retrieved from <https://docs.python.org/3/library/shutil.html>
* Python Documentation. (n.d.). *tkinter — Python's standard GUI library*. Python.org. Retrieved from <https://docs.python.org/3/library/tkinter.html>
* PyInstaller Documentation. (n.d.). *PyInstaller Manual*. Retrieved from <https://pyinstaller.org/en/stable/>

**Tools & Resources**

* GitHub. (n.d.). *Version Control & Collaboration Platform*. Retrieved from <https://github.com>
* Stack Overflow. (n.d.). *Community-driven Q&A for programming & debugging help*. Retrieved from <https://stackoverflow.com>
* PyCharm/VS Code. (n.d.). *Integrated Development Environments (IDEs) used for coding*. Retrieved from <https://www.jetbrains.com/pycharm/> and <https://code.visualstudio.com/>

**AI Assistance**

* OpenAI. (2024). *ChatGPT AI Language Model*. Retrieved from <https://openai.com/chatgpt>

**Own Contributions**

* Lagerblad, B. (2025). *Degree Project*. GitHub Repository. <https://github.com/Markofbear/Degree_project>