Python Project 10: File Organizer

1. Project Overview

Build a simple desktop **File Organizer** that automatically sorts files from a selected folder into separate folders based on their file extensions.

For example:

```
Downloads/
□□□ photo.jpg
□□□ report.pdf
□□□ song.mp3
□□□ notes.txt
□□□ video.mp4
```

After organizing:

```
Downloads/
□□□ Images/
□   □□□ photo.jpg
□□□ Documents/
□   □□□ report.pdf
□   □□□ notes.txt
□□□ Audio/
□   □□□ song.mp3
□□□ Videos/
    □□□ video.mp4
```

Python's `pathlib` provides cross-platform path and filesystem operations, while `Path.iterdir()` can be used to inspect directory contents.

2. Features

- Select a folder
- Organize files automatically
- Categorize files by extension
- Images folder
- Documents folder
- Videos folder
- Audio folder
- Archives folder
- Code folder
- Other Files folder
- Avoid overwriting existing files
- Display moved file count
- Simple Tkinter interface
- No external packages

3. Technologies

- Python 3
- Tkinter
- pathlib
- shutil
- Object Oriented Programming

4. Folder Structure

```
file_organizer/
□□□ main.py
□□□ ui.py
□□□ organizer.py
□□□ README.md
```

5. How the Project Works

The project has two main parts:

Frontend

`ui.py` provides the graphical interface where the user selects a folder and starts the organization process.

Backend

`organizer.py` checks each file extension, determines its category, creates the required folder, and moves the file.

The project uses `pathlib` for filesystem paths and `shutil.move()` for moving files.

6. Complete Backend Code

organizer.py

```
from pathlib import Path
import shutil


class FileOrganizer:
    FILE_CATEGORIES = {
        "Images": {
            ".jpg", ".jpeg", ".png", ".gif",
            ".bmp", ".webp", ".svg"
        },
        "Documents": {
            ".pdf", ".doc", ".docx", ".txt",
            ".xlsx", ".xls", ".ppt", ".pptx",
            ".csv"
        },
        "Videos": {
            ".mp4", ".mkv", ".avi",
```

```
            ".mov", ".wmv", ".webm"
        },
        "Audio": {
            ".mp3", ".wav", ".aac",
            ".flac", ".ogg", ".m4a"
        },
        "Archives": {
            ".zip", ".rar", ".7z",
            ".tar", ".gz"
        },
        "Code": {
            ".py", ".js", ".jsx", ".ts",
            ".tsx", ".html", ".css",
            ".java", ".cpp", ".c",
            ".cs", ".php"
        }
    }

    def __init__(self, folder_path):
        self.folder = Path(folder_path)

    def get_category(self, file_path):
        extension = file_path.suffix.lower()

        for category, extensions in self.FILE_CATEGORIES.items():
            if extension in extensions:
                return category

        return "Other Files"

    def get_unique_path(self, destination):
        if not destination.exists():
            return destination
```

```
        counter = 1

        while True:
            new_name = (
                f"{destination.stem}_{counter}"
                f"{destination.suffix}"
            )

            new_path = destination.with_name(new_name)

            if not new_path.exists():
                return new_path

            counter += 1

    def organize(self):
        if not self.folder.exists():
            raise FileNotFoundError("Folder does not exist.")

        if not self.folder.is_dir():
            raise NotADirectoryError("Selected path is not a folder.")

        moved_files = 0

        for file_path in self.folder.iterdir():

            if not file_path.is_file():
                continue

            category = self.get_category(file_path)
            category_folder = self.folder / category

            category_folder.mkdir(
                parents=True,
```

```
            exist_ok=True
        )

        destination = category_folder / file_path.name
        destination = self.get_unique_path(destination)

        shutil.move(
            str(file_path),
            str(destination)
        )

        moved_files += 1

    return moved_files
```

7. Complete Frontend Code

ui.py

```
import tkinter as tk
from tkinter import filedialog, messagebox

from organizer import FileOrganizer


class FileOrganizerUI:

    def __init__(self, root):
        self.root = root
        self.root.title("File Organizer")
        self.root.geometry("650x400")
        self.root.configure(bg="#0f172a")

        self.selected_folder = tk.StringVar()
```

```
        self.create_ui()

    def create_ui(self):
        title = tk.Label(
            self.root,
            text="File Organizer",
            font=("Arial", 24, "bold"),
            bg="#0f172a",
            fg="white"
        )
        title.pack(pady=(35, 10))

        subtitle = tk.Label(
            self.root,
            text="Automatically organize files by type",
            font=("Arial", 11),
            bg="#0f172a",
            fg="#94a3b8"
        )
        subtitle.pack()

        folder_frame = tk.Frame(
            self.root,
            bg="#1e293b",
            padx=20,
            pady=20
        )
        folder_frame.pack(
            fill="x",
            padx=50,
            pady=30
        )

        folder_label = tk.Label(
```

```
            folder_frame,
            text="Selected Folder",
            font=("Arial", 11, "bold"),
            bg="#1e293b",
            fg="white"
        )
        folder_label.pack(anchor="w")

        entry = tk.Entry(
            folder_frame,
            textvariable=self.selected_folder,
            font=("Arial", 11),
            bg="#334155",
            fg="white",
            insertbackground="white",
            relief="flat"
        )
        entry.pack(
            fill="x",
            pady=10,
            ipady=8
        )

        browse_button = tk.Button(
            folder_frame,
            text="Browse Folder",
            command=self.browse_folder,
            bg="#334155",
            fg="white",
            relief="flat",
            padx=15,
            pady=8
        )
        browse_button.pack(anchor="w")
```

```
        organize_button = tk.Button(
            self.root,
            text="Organize Files",
            command=self.organize_files,
            bg="#2563eb",
            fg="white",
            font=("Arial", 12, "bold"),
            relief="flat",
            padx=30,
            pady=12
        )
        organize_button.pack()

        self.status_label = tk.Label(
            self.root,
            text="Select a folder to begin.",
            font=("Arial", 10),
            bg="#0f172a",
            fg="#94a3b8"
        )
        self.status_label.pack(pady=20)

    def browse_folder(self):
        folder = filedialog.askdirectory(
            title="Select Folder"
        )

        if folder:
            self.selected_folder.set(folder)
            self.status_label.config(
                text="Folder selected. Ready to organize."
            )
```

```
    def organize_files(self):
        folder = self.selected_folder.get().strip()

        if not folder:
            messagebox.showwarning(
                "No Folder",
                "Please select a folder first."
            )
            return

        try:
            organizer = FileOrganizer(folder)
            moved_count = organizer.organize()

            self.status_label.config(
                text=f"{moved_count} file(s) organized successfully."
            )

            messagebox.showinfo(
                "Completed",
                f"{moved_count} file(s) organized successfully."
            )

        except Exception as error:
            messagebox.showerror(
                "Error",
                str(error)
            )
```

8. Main Code

main.py

```
import tkinter as tk
```

```
from ui import FileOrganizerUI


def main():
    root = tk.Tk()
    FileOrganizerUI(root)
    root.mainloop()


if __name__ == "__main__":
    main()
```

9. README

README.md

```
# File Organizer

A simple Python desktop application that organizes files
into folders based on their extensions.

## Features

- Select a folder
- Organize files automatically
- Categorize files
- Prevent filename conflicts
- Simple Tkinter interface

## Requirements

Python 3

No external packages are required.
```

```
## Run

python main.py
```

10. How Frontend + Backend Connect

The flow is:

```
User selects folder
        ↓
ui.py
        ↓
FileOrganizer
        ↓
Check file extension
        ↓
Find category
        ↓
Create category folder
        ↓
Move file
        ↓
Show result
```

`ui.py` creates a `FileOrganizer` object and calls:

```
organizer.organize()
```

The backend handles all file organization logic.

11. How to Run

Open the project folder in a terminal:

```
cd file_organizer
```

Run:

```
python main.py
```

Select a folder containing files and click **Organize Files**.

12. Basic Testing

Test with a folder containing:

```
photo.jpg
document.pdf
song.mp3
movie.mp4
project.py
archive.zip
unknown.xyz
```

Expected result:

```
Images/photo.jpg
Documents/document.pdf
Audio/song.mp3
Videos/movie.mp4
Code/project.py
Archives/archive.zip
Other Files/unknown.xyz
```

Also test:

- Empty folder
- Invalid folder

- Multiple files with the same name
- Files without extensions
- Re-running the organizer

The application generates a new filename when a destination file already exists, so an existing file is not overwritten.

13. Small Improvements

- Add drag and drop
- Add custom categories
- Add undo functionality
- Add file preview
- Add organization history
- Add a dry-run mode
- Add support for subfolders
- Add a progress bar

Visit haas.dev for more resources and guides.