# **Documentation: Project 5, Automation Script**.

**Project Overview:**

We will create a Python automation script that automates repetitive tasks. You can choose a specific task that you'd like to automate, but here are a few common automation ideas:

* **File Management Automation**: Automatically organize files in specific directories by file type.
* **Data Entry Automation**: Automate repetitive data entry tasks from one source to another.
* **Email Automation**: Automatically download and organize your emails into categorized folders.

Let's go with the **File Management Automation** project for simplicity and practicality.

**Project 5: File Management Automation Script**

**Step 1: Define the Project Goals**

We will write a Python script that organizes files into subfolders based on their file type (e.g., .pdf, .jpg, .txt, etc.) within a specified directory.

**Step 2: Requirements**

1. **Python 3.x** installed on your system.
2. Libraries to install (if not already installed):
   * os
   * shutil
   * argparse (optional for more advanced users, if you want to add command-line arguments)

bash

Copy code

pip install argparse # Optional, if you want to use CLI arguments

**Step 3: Create the Automation Script**

Here’s a Python script that will organize files by type:

python

Copy code

import os

import shutil

# Dictionary to map file types to directories

FILE\_TYPES = {

'Documents': ['.pdf', '.docx', '.txt', '.xlsx'],

'Images': ['.jpg', '.jpeg', '.png', '.gif'],

'Videos': ['.mp4', '.mov', '.avi'],

'Music': ['.mp3', '.wav', '.aac'],

'Archives': ['.zip', '.rar', '.7z', '.tar'],

}

def organize\_files(directory):

"""

Organize files into folders based on their file type.

:param directory: The path of the directory to organize

"""

if not os.path.exists(directory):

print(f"The directory {directory} does not exist.")

return

for filename in os.listdir(directory):

file\_path = os.path.join(directory, filename)

if os.path.isfile(file\_path):

file\_ext = os.path.splitext(filename)[1].lower()

# Find the appropriate folder for the file type

for folder\_name, extensions in FILE\_TYPES.items():

if file\_ext in extensions:

folder\_path = os.path.join(directory, folder\_name)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path) # Create folder if it doesn't exist

# Move the file into the appropriate folder

shutil.move(file\_path, os.path.join(folder\_path, filename))

print(f"Moved: {filename} --> {folder\_name}")

break

else:

# If file type doesn't match any in the list, move to 'Others'

other\_folder = os.path.join(directory, 'Others')

if not os.path.exists(other\_folder):

os.makedirs(other\_folder)

shutil.move(file\_path, os.path.join(other\_folder, filename))

print(f"Moved: {filename} --> Others")

if \_\_name\_\_ == "\_\_main\_\_":

folder\_to\_organize = input("Enter the directory path to organize: ")

organize\_files(folder\_to\_organize)

**Step 4: How It Works**

1. **Organize by File Type**: The script checks the file extension and moves files to directories based on their type (e.g., PDFs into Documents, JPEGs into Images, etc.).
2. **Create Folders**: If a directory for a file type doesn’t exist (e.g., Documents), the script creates one automatically.
3. **Unknown File Types**: If a file doesn’t match any known extensions, it’s moved to an Others directory.
4. **Input Directory**: The user inputs the folder they want to organize.

**Step 5: Running the Script**

1. Save the script as file\_organizer.py.
2. Open a terminal and navigate to the folder where the script is saved.
3. Run the script:

bash

Copy code

python file\_organizer.py

1. Enter the path of the directory you want to organize when prompted.

**Step 6: (Optional) Advanced Features**

You can expand this script by adding more features:

1. **Command-Line Arguments**: Use argparse to allow users to pass in the directory via the command line.
2. **Log File**: Log the actions taken by the script (e.g., create a .log file that records which files were moved).
3. **Scheduler**: Set up the script to run periodically using a task scheduler like cron (on Unix-based systems) or Task Scheduler (on Windows).

Let's break down the provided Python code, line by line, to explain what it does.

python

Copy code

import os

import shutil

* **import os**: This imports the os module, which provides functions to interact with the operating system, such as navigating file systems, checking paths, creating directories, etc.

### 1. Import Statements

python

Copy code

import os

import shutil

* **import os**: The os module provides functions for interacting with the operating system. It is used to navigate directories, check file types, and manipulate paths.
* **import shutil**: The shutil module provides functions for high-level file operations, such as copying and moving files or directories.

### 2. Dictionary to Map File Types to Directories

python

Copy code

FILE\_TYPES = {

'Documents': ['.pdf', '.docx', '.txt', '.xlsx'],

'Images': ['.jpg', '.jpeg', '.png', '.gif'],

'Videos': ['.mp4', '.mov', '.avi'],

'Music': ['.mp3', '.wav', '.aac'],

'Archives': ['.zip', '.rar', '.7z', '.tar'],

}

* **FILE\_TYPES**: This is a dictionary that maps file categories (e.g., "Documents", "Images") to a list of file extensions. Each key represents a folder where files of certain types (represented by extensions) will be moved.
* Example: Files with extensions .pdf, .docx, etc., will be moved to a "Documents" folder.

### 3. The Function organize\_files()

python

Copy code

def organize\_files(directory):

* **def organize\_files(directory):**: This defines a function called organize\_files that takes one argument, directory, which is the path to the folder that needs organizing.

### 4. Checking if the Directory Exists

python

Copy code

if not os.path.exists(directory):

print(f"The directory {directory} does not exist.")

return

* **if not os.path.exists(directory):**: This checks whether the directory path passed to the function actually exists. The os.path.exists() function returns True if the directory exists, and False otherwise.
* **print(f"The directory {directory} does not exist.")**: If the directory doesn’t exist, the function prints an error message to the console.
* **return**: This exits the function early if the directory does not exist.

### 5. Looping Through the Directory’s Contents

python

Copy code

for filename in os.listdir(directory):

file\_path = os.path.join(directory, filename)

* **for filename in os.listdir(directory):**: This line loops through all files and folders in the specified directory using os.listdir(), which returns a list of all filenames in the directory.
* **file\_path = os.path.join(directory, filename)**: This constructs the full path of each file or folder by joining the directory path with the filename.

### 6. Checking if the Item is a File

python

Copy code

if os.path.isfile(file\_path):

file\_ext = os.path.splitext(filename)[1].lower()

* **if os.path.isfile(file\_path):**: This checks if the current item (given by file\_path) is a file, and not a folder, using os.path.isfile().
* **file\_ext = os.path.splitext(filename)[1].lower()**: This extracts the file extension from the filename using os.path.splitext(). The splitext() function splits a file into its name and extension, and [1] accesses the extension part (e.g., .pdf). The lower() function ensures the extension is lowercase to handle case variations (e.g., .JPG and .jpg).

### 7. Looping Through the FILE\_TYPES Dictionary

python

Copy code

for folder\_name, extensions in FILE\_TYPES.items():

if file\_ext in extensions:

* **for folder\_name, extensions in FILE\_TYPES.items():**: This loops through each key-value pair in the FILE\_TYPES dictionary. folder\_name is the folder category (e.g., "Documents"), and extensions is the list of file extensions associated with that folder.
* **if file\_ext in extensions:**: This checks if the file’s extension matches any of the extensions for the current folder category. If it does, the file will be moved into that folder.

### 8. Creating the Folder (if Necessary)

python

Copy code

folder\_path = os.path.join(directory, folder\_name)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path)

* **folder\_path = os.path.join(directory, folder\_name)**: This creates the path for the target folder where the file will be moved. It combines the directory path with the folder name (e.g., Documents, Images).
* **if not os.path.exists(folder\_path):**: This checks if the folder already exists.
* **os.makedirs(folder\_path)**: If the folder doesn’t exist, this creates it using os.makedirs().

### 9. Moving the File

python

Copy code

shutil.move(file\_path, os.path.join(folder\_path, filename))

print(f"Moved: {filename} --> {folder\_name}")

* **shutil.move(file\_path, os.path.join(folder\_path, filename))**: This moves the file to the appropriate folder using shutil.move(). It constructs the destination path by joining the folder\_path and filename.
* **print(f"Moved: {filename} --> {folder\_name}")**: After moving the file, the function prints a message indicating the file has been moved and specifies its new folder.

### 10. Handling Unknown File Types

python

Copy code

else:

other\_folder = os.path.join(directory, 'Others')

if not os.path.exists(other\_folder):

os.makedirs(other\_folder)

shutil.move(file\_path, os.path.join(other\_folder, filename))

print(f"Moved: {filename} --> Others")

* **else:**: If the file’s extension doesn’t match any of the extensions in FILE\_TYPES, this block is executed.
* **other\_folder = os.path.join(directory, 'Others')**: This defines a folder called "Others" where unknown file types will be moved.
* **if not os.path.exists(other\_folder):**: If the "Others" folder doesn’t exist, it is created using os.makedirs().
* **shutil.move(file\_path, os.path.join(other\_folder, filename))**: This moves the file to the "Others" folder.
* **print(f"Moved: {filename} --> Others")**: A message is printed to indicate that the file was moved to the "Others" folder.

### 11. Main Execution Block

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

folder\_to\_organize = input("Enter the directory path to organize: ")

organize\_files(folder\_to\_organize)

* **if \_\_name\_\_ == "\_\_main\_\_":**: This ensures that the script only runs when executed directly, not when imported as a module in another script.
* **folder\_to\_organize = input("Enter the directory path to organize: ")**: This prompts the user to input the path of the directory they want to organize. The input() function gets the user's input.
* **organize\_files(folder\_to\_organize)**: The organize\_files() function is called, and the user-provided directory path is passed as the argument.

### 1. Adding Command-Line Arguments with argparse

Instead of asking the user to input the directory path via input(), you can pass it as a command-line argument. This makes the script more versatile, especially when running it from different environments or as part of a batch process.

#### Steps:

1. **Install argparse** if it’s not already installed (though it’s part of Python’s standard library):

bash

Copy code

pip install argparse

1. **Modify the Script**: Here's how to modify your script to use argparse:

python

Copy code

import os

import shutil

import argparse

FILE\_TYPES = {

'Documents': ['.pdf', '.docx', '.txt', '.xlsx'],

'Images': ['.jpg', '.jpeg', '.png', '.gif'],

'Videos': ['.mp4', '.mov', '.avi'],

'Music': ['.mp3', '.wav', '.aac'],

'Archives': ['.zip', '.rar', '.7z', '.tar'],

}

def organize\_files(directory):

if not os.path.exists(directory):

print(f"The directory {directory} does not exist.")

return

for filename in os.listdir(directory):

file\_path = os.path.join(directory, filename)

if os.path.isfile(file\_path):

file\_ext = os.path.splitext(filename)[1].lower()

for folder\_name, extensions in FILE\_TYPES.items():

if file\_ext in extensions:

folder\_path = os.path.join(directory, folder\_name)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path)

shutil.move(file\_path, os.path.join(folder\_path, filename))

print(f"Moved: {filename} --> {folder\_name}")

break

else:

other\_folder = os.path.join(directory, 'Others')

if not os.path.exists(other\_folder):

os.makedirs(other\_folder)

shutil.move(file\_path, os.path.join(other\_folder, filename))

print(f"Moved: {filename} --> Others")

if \_\_name\_\_ == "\_\_main\_\_":

# Set up argparse for command-line arguments

parser = argparse.ArgumentParser(description="Organize files in a directory by their file type.")

parser.add\_argument('directory', type=str, help="The path to the directory to organize")

args = parser.parse\_args()

# Call the function to organize the files

organize\_files(args.directory)

1. **Usage**: After modifying the script, you can run it from the command line like this:

bash

Copy code

python file\_organizer.py C:\path\to\your\directory

### 2. Adding Logging

Logging the operations that the script performs is useful for tracking what files were moved, when it happened, and where they were moved. You can store this information in a .log file for future reference.

#### Steps:

1. **Import logging** and set up basic logging functionality at the start of your script.

python

Copy code

import logging

1. **Set Up Logging Configuration**: Configure the logging system so that it writes to a file (e.g., file\_organizer.log):

python

Copy code

logging.basicConfig(filename='file\_organizer.log', level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s')

1. **Modify the File Movement Code to Log Actions**: Update the code to log each file movement:

python

Copy code

if os.path.isfile(file\_path):

file\_ext = os.path.splitext(filename)[1].lower()

for folder\_name, extensions in FILE\_TYPES.items():

if file\_ext in extensions:

folder\_path = os.path.join(directory, folder\_name)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path)

shutil.move(file\_path, os.path.join(folder\_path, filename))

print(f"Moved: {filename} --> {folder\_name}")

logging.info(f"Moved: {filename} --> {folder\_name}")

break

else:

other\_folder = os.path.join(directory, 'Others')

if not os.path.exists(other\_folder):

os.makedirs(other\_folder)

shutil.move(file\_path, os.path.join(other\_folder, filename))

print(f"Moved: {filename} --> Others")

logging.info(f"Moved: {filename} --> Others")

1. **Usage**: After you run the script, it will create (or update) a file named file\_organizer.log that records each file's movement, including a timestamp.

### 3. Scheduling the Script to Run Periodically

You can set up the script to run automatically at scheduled intervals using task schedulers like **Task Scheduler (Windows)** or **cron (Linux/macOS)**.

#### 3.1 Scheduling with Windows Task Scheduler

1. Open **Task Scheduler** from the Windows start menu.
2. Click **Create Basic Task** and follow the wizard:
   * Name: "File Organizer"
   * Trigger: Set the schedule (e.g., daily, weekly).
   * Action: **Start a Program**
   * Program: Browse for your Python executable (python.exe).
   * Add Arguments: Specify the path to the script and the directory to organize.
   * Example:

bash

Copy code

"C:\path\to\your\script\file\_organizer.py" C:\path\to\organize

1. Complete the setup, and Task Scheduler will run the script automatically at the specified time.

#### 3.2 Scheduling with cron (Linux/macOS)

1. Open a terminal and type:

bash

Copy code

crontab -e

1. Add a new cron job:

bash

Copy code

0 2 \* \* \* /usr/bin/python3 /path/to/your/script/file\_organizer.py /path/to/directory

This will run the script every day at 2 AM. You can adjust the timing as per your needs.

1. Save and exit. Your script will now run automatically based on the schedule you defined.

### Summary of Optional Features:

* **Command-Line Arguments**: Use argparse to make the script accept command-line arguments for the directory path.
* **Logging**: Track the script’s actions (file movements) in a log file for future reference.
* **Scheduled Execution**: Automate the script to run periodically using Windows Task Scheduler or cron.

These features will make the script much more powerful and user-friendly. Let me know if you need help with any of the implementations!

To commit the changes for **Step 1: Adding Command-Line Arguments with argparse** to GitHub, follow these steps:

### Step 1: Modify Your Script

Ensure your script has been modified with the argparse functionality as described earlier. Save your changes to the script.

### Step 2: Check the Changes

To see the changes you've made, run:

bash

Copy code

git status

This will show you any modified files.

### Step 3: Stage the Changes

Stage the modified script (for example, if the script is named file\_organizer.py):

bash

Copy code

git add file\_organizer.py

### Step 4: Commit the Changes

Commit the changes with a descriptive message:

bash

Copy code

git commit -m "Added command-line arguments using argparse for directory input"

### Step 5: Push the Changes to GitHub

Finally, push the changes to your GitHub repository:

bash

Copy code

git push origin main

This will upload the updated script to the main branch of your repository.

Let me know if you encounter any issues, or if you need help with further steps!

4o

##### You said:

modify the code for this step:2. Adding Logging

###### ChatGPT said:

ChatGPT

To modify the code for **Step 2: Adding Logging**, follow the instructions below. The logging module will be used to log file movements in a log file (file\_organizer.log).

### Step 1: Modify the Script to Add Logging

Here’s how you can modify your script to include logging:

python

Copy code

import os

import shutil

import argparse

import logging

# Configure logging to write to file\_organizer.log

logging.basicConfig(

filename='file\_organizer.log',

level=logging.INFO,

format='%(asctime)s - %(levelname)s - %(message)s'

)

FILE\_TYPES = {

'Documents': ['.pdf', '.docx', '.txt', '.xlsx'],

'Images': ['.jpg', '.jpeg', '.png', '.gif'],

'Videos': ['.mp4', '.mov', '.avi'],

'Music': ['.mp3', '.wav', '.aac'],

'Archives': ['.zip', '.rar', '.7z', '.tar'],

}

def organize\_files(directory):

"""

Organize files into folders based on their file type.

:param directory: The path of the directory to organize

"""

if not os.path.exists(directory):

print(f"The directory {directory} does not exist.")

logging.error(f"The directory {directory} does not exist.")

return

for filename in os.listdir(directory):

file\_path = os.path.join(directory, filename)

if os.path.isfile(file\_path):

file\_ext = os.path.splitext(filename)[1].lower()

# Find the appropriate folder for the file type

for folder\_name, extensions in FILE\_TYPES.items():

if file\_ext in extensions:

folder\_path = os.path.join(directory, folder\_name)

if not os.path.exists(folder\_path):

os.makedirs(folder\_path) # Create folder if it doesn't exist

# Move the file into the appropriate folder

shutil.move(file\_path, os.path.join(folder\_path, filename))

print(f"Moved: {filename} --> {folder\_name}")

logging.info(f"Moved: {filename} --> {folder\_name}")

break

else:

# If file type doesn't match any in the list, move to 'Others'

other\_folder = os.path.join(directory, 'Others')

if not os.path.exists(other\_folder):

os.makedirs(other\_folder)

shutil.move(file\_path, os.path.join(other\_folder, filename))

print(f"Moved: {filename} --> Others")

logging.info(f"Moved: {filename} --> Others")

if \_\_name\_\_ == "\_\_main\_\_":

# Set up argparse for command-line arguments

parser = argparse.ArgumentParser(description="Organize files in a directory by their file type.")

parser.add\_argument('directory', type=str, help="The path to the directory to organize")

args = parser.parse\_args()

# Call the function to organize the files

organize\_files(args.directory)

### Changes Made:

1. **Logging Setup:**
   * The logging.basicConfig() is used to set up logging to a file (file\_organizer.log) with the following parameters:
     + filename='file\_organizer.log': Specifies the name of the log file.
     + level=logging.INFO: Logs all events with severity level INFO or higher.
     + format='%(asctime)s - %(levelname)s - %(message)s': Specifies the format of each log entry (timestamp, severity, message).
2. **Logging in the Function:**
   * **Error logging**: If the directory doesn’t exist, it logs an error using logging.error().
   * **File movements**: Each time a file is moved, an INFO-level log entry is added using logging.info().

### Step 2: Test the Logging

After running the script, a file\_organizer.log file will be generated, containing entries like this:

yaml

Copy code

2023-09-05 12:34:56,123 - INFO - Moved: example.pdf --> Documents

2023-09-05 12:34:57,456 - INFO - Moved: image.jpg --> Images

2023-09-05 12:34:58,789 - ERROR - The directory C:\nonexistent\path does not exist.

### Step 3: Commit the Changes to GitHub

1. **Check the modified files**:

bash

Copy code

git status

1. **Stage the changes**:

bash

Copy code

git add file\_organizer.py

1. **Commit the changes**:

bash

Copy code

git commit -m "Added logging functionality to log file movements"

1. **Push the changes to GitHub**:

bash

Copy code

git push origin main

This will add the logging feature to your script and push the updated version to your GitHub repository.

Let me know if you need any help with this!

4o

##### You said:

and lets finish the project with this step:Scheduling the Script to Run Periodically

2/2

###### ChatGPT said:

ChatGPT

To finish the project by scheduling the script to run periodically, you can set up the script to run at scheduled intervals using **Windows Task Scheduler** (for Windows) or **cron** (for Linux/macOS). I’ll guide you through both approaches depending on your system.

### 1. Scheduling the Script with **Windows Task Scheduler**

Here’s how you can schedule the script to run periodically on Windows:

#### Step 1: Create a Batch File (Optional but Helpful)

Since Task Scheduler often runs scripts through the command line, it’s convenient to create a .bat file that can run the Python script with your desired arguments.

1. Open a text editor and create a new file.
2. Add the following content to the file:

bat

Copy code

@echo off

cd C:\path\to\your\project # Navigate to the directory where the script is located

venv\Scripts\activate # Activate the virtual environment (if you're using one)

python file\_organizer.py C:\path\to\directory # Run the Python script with the target directory

1. Save this file as run\_file\_organizer.bat.

#### Step 2: Set Up Task Scheduler

1. Open **Task Scheduler** from the Windows start menu.
2. Click **Create Basic Task** in the right-hand sidebar.
3. Give your task a name and description, such as "File Organizer".
4. Set the **Trigger** (e.g., "Daily", "Weekly", or "At system startup").
5. Choose the **Action** as **Start a Program**.
6. In the **Program/Script** box, browse for the .bat file you just created (run\_file\_organizer.bat).
7. Complete the setup, and Task Scheduler will now automatically run the script based on your schedule.

#### Example Task Scheduler Settings:

* **Trigger**: Daily at 2 AM
* **Action**: Start the batch file (run\_file\_organizer.bat)
* **Script**: This will run the Python script to organize your files every day at the specified time.

### 2. Scheduling the Script with **cron** (Linux/macOS)

For Linux/macOS systems, you can use **cron** to schedule the script.

#### Step 1: Open crontab

Open a terminal and run:

bash

Copy code

crontab -e

This opens the cron jobs list in the default text editor.

#### Step 2: Add a Cron Job

Add the following line to schedule the script:

bash

Copy code

0 2 \* \* \* /usr/bin/python3 /path/to/your/project/file\_organizer.py /path/to/directory

Explanation:

* **0 2 \* \* \***: This means the script will run every day at 2:00 AM. You can adjust this timing as needed.
* **/usr/bin/python3**: Path to the Python executable (you can find this with which python3).
* **/path/to/your/project/file\_organizer.py**: Full path to your Python script.
* **/path/to/directory**: The directory that you want to organize.

Save and close the editor. Your cron job will now run the Python script automatically at the specified time.

#### Example Cron Job:

To run the script every day at 2 AM:

bash

Copy code

0 2 \* \* \* /usr/bin/python3 /home/user/projects/file\_organizer.py /home/user/Downloads

### Step 3: Verify the Scheduled Task

For both Windows Task Scheduler and cron, after setting up the task, make sure to check that the script runs as expected on schedule. You can also verify that the logging system is capturing the file movements as the script executes.

### Summary of the Scheduling Steps:

1. **Windows**: Use Task Scheduler and optionally create a batch file to run the script.
2. **Linux/macOS**: Use cron to schedule the Python script.

### Step 4: Commit the Final Project to GitHub

If you created any additional files (like the .bat file), you can add them to your GitHub project:

1. **Check the status**:

bash

Copy code

git status

1. **Stage any new or modified files**:

bash

Copy code

git add .

1. **Commit the final changes**:

bash

Copy code

git commit -m "Added script scheduling instructions for Task Scheduler and cron"

1. **Push to GitHub**:

bash

Copy code

git push origin main

This will finalize the project, including the scheduling feature, and push it to your repository.