Department of computer science and engineering

Course Code: CSE -3632

Course Title: operating System sessional

Project report

" Automating File Organization with Python "

SUBMITTED BY

Name: Simoon Nahar

Id: C-201212

Semester: 6th

Section:6AF

SUBMITTED TO

Zainal Abedin

Associate professor

Department of CSE, IIUC

Date of submission: 28 June,2024

Automating File Organization with Python

OBJECTIVE:

- To develop a user-friendly tool for automating file organization based on user preferences in an OS.
- To organize by grouping files based on type or size, the script promotes better organization within a directory in OS.
- To Separate large project files from smaller working documents for easier access.
- To gain Streamline workflows by having files readily accessible within categorized subdirectories.

INTRODUCTION:

This project aims to develop a user-friendly Python script for automating file organization within a directory based on user-specified criteria. It provides a practical tool to streamline file management and improve accessibility.

The main features of this projects:

1. **User-defined Organization:** Users can choose between two main modes for organizing files:
   a. **Type-based:** Files are sorted and moved to subdirectories based on their extension (e.g., ".pdf", ".docx").
   b. **Size-based:** Files are categorized into "Small Files" and "Large Files" subdirectories based on a user-defined size threshold (default: 10 MB).
2. **Error Handling:** The script includes error handling mechanisms to catch potential exceptions during file operations (e.g., moving files, creating directories). It provides informative error messages for troubleshooting.
3. **User Interaction:** The script prompts users for the directory path and organization mode, ensuring a user-friendly experience.

The benefit of this project:

1. **Reduced Manual Effort:** Automates the sorting process, saving users time and effort in organizing their files.
2. **Improved File Management:** Groups files based on type or size, promoting a more organized directory structure.
3. **Enhanced Accessibility:** Makes files easier to locate by categorizing them within relevant subdirectories.

Uses of this project:

- **OS Concepts:** Provides practical experience with operating system functionalities related to file systems and file management.
- **Python Programming:** Demonstrates the use of Python libraries for file manipulation tasks.

- **Scripting Principles:** Showcases modularity, user interaction, and error handling within a scripting context.

This project offers a foundation for building a more robust file organization tool and serves as a valuable learning resource for understanding essential concepts in OS, Python programming, and scripting principles.

METHODOLOGY:

The methodology of this projects are:

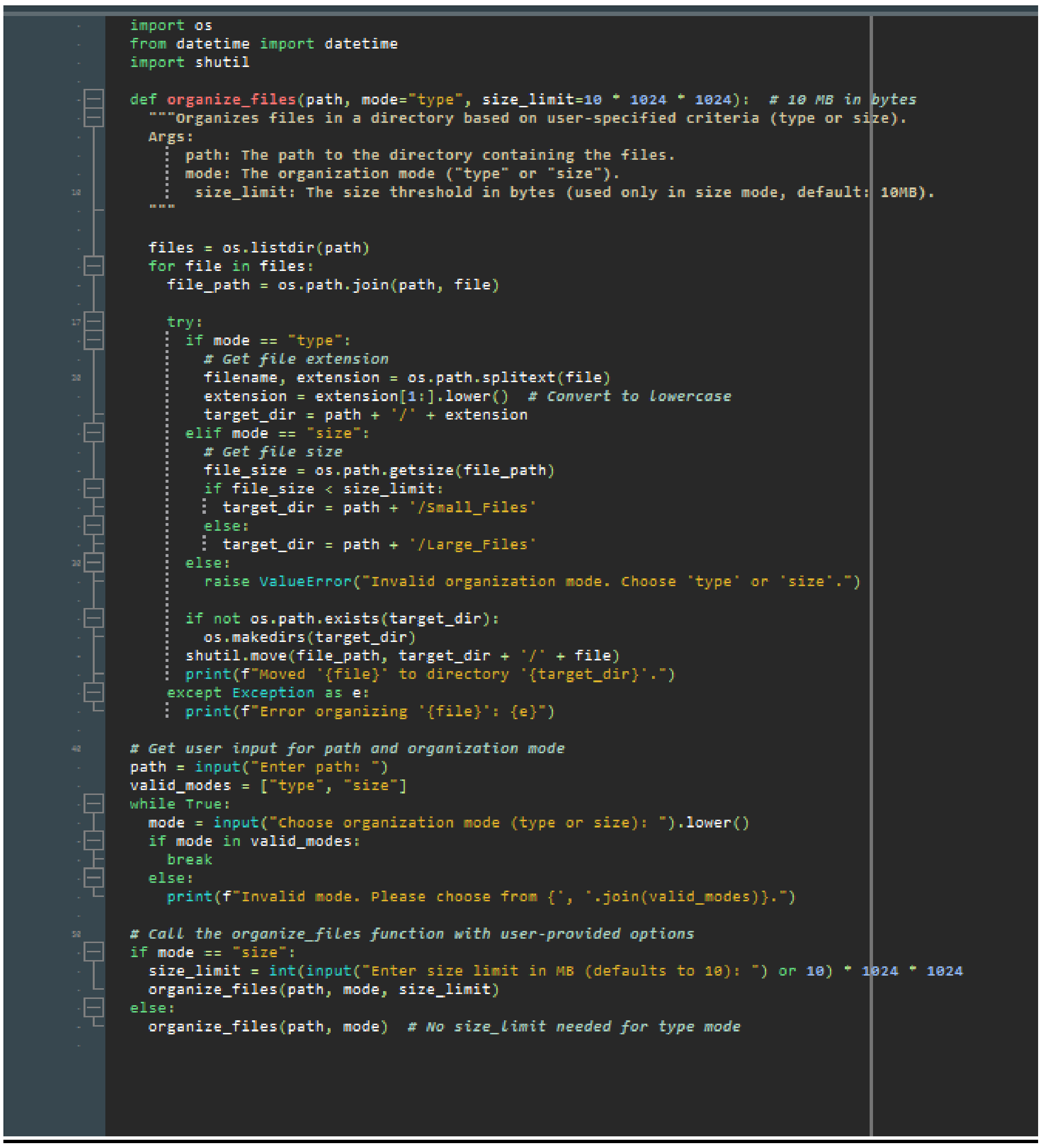
1. **Define Target Users:** Identify who will be using the script (beginners, experienced users).
2. **Choose Organization Modes:** Decide on the primary methods (type, size, potential date or custom).
3. **Function Breakdown:** Plan core functionalities as well-defined functions (e.g., get_user_input, organize_by_type).
4. **Data Structures:** Identify necessary data structures (e.g., lists for file paths, dictionaries for configuration options).
5. **Error Handling Integration:** Plan how to handle exceptions (try-except blocks, informative error messages).
6. **User Interaction Functions:** Develop functions for: - Getting user input (directory path and organization mode). - Providing clear instructions and prompts.
7. **Organization Mode Functions:** Implement functions for each mode:

    - **Type-Based:** - Extract file extensions (using os.path.splitext). - Create subdirectories for each unique extension (using os.makedirs). - Move files to corresponding subdirectories (using shutil.move).
    - **Size-Based (optional):** - Get file size (using os.path.getsize). - Create "Small_Files" and "Large_Files" subdirectories. - Move files based on user-defined size threshold (using conditional statements).

8. **Error Message Improvement:** Enhance user experience by refining error messages.

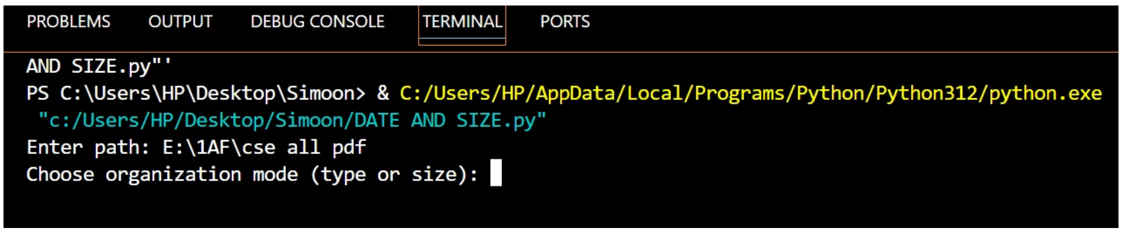
RESULT AND ANALYSIS:

Code snippet:



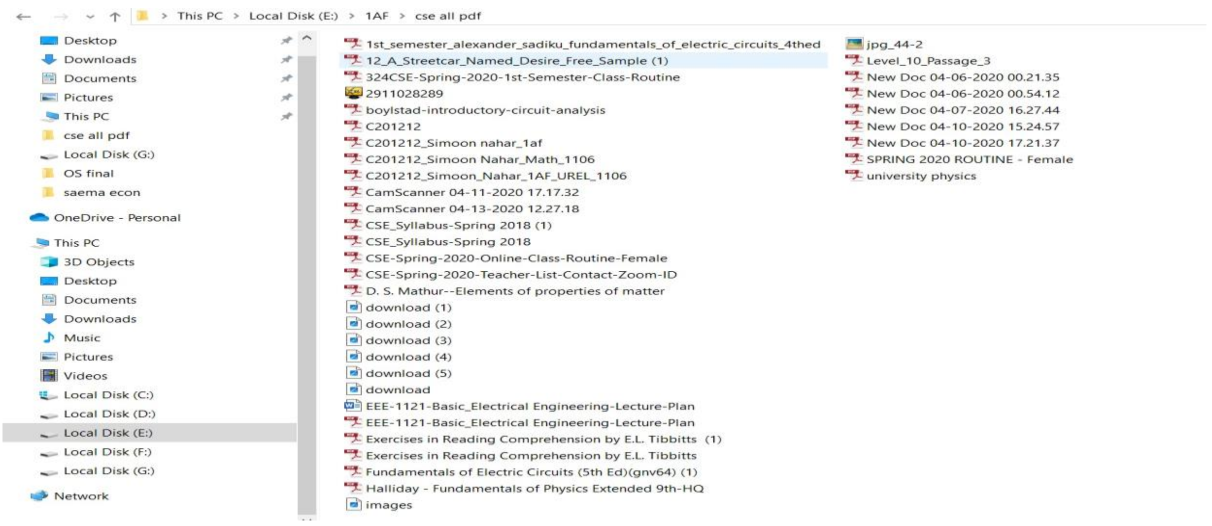
```python
import os
from datetime import datetime
import shutil

def organize_files(path, mode="type", size_limit=10 * 1024 * 1024):  # 10 MB in bytes
  """Organizes files in a directory based on user-specified criteria (type or size).
  Args:
    path: The path to the directory containing the files.
    mode: The organization mode ("type" or "size").
     size_limit: The size threshold in bytes (used only in size mode, default: 10MB).
  """

  files = os.listdir(path)
  for file in files:
    file_path = os.path.join(path, file)

    try:
      if mode == "type":
        # Get file extension
        filename, extension = os.path.splitext(file)
        extension = extension[1:].lower()  # Convert to lowercase
        target_dir = path + '/' + extension
      elif mode == "size":
        # Get file size
        file_size = os.path.getsize(file_path)
        if file_size < size_limit:
          target_dir = path + '/Small_Files'
        else:
          target_dir = path + '/Large_Files'
      else:
        raise ValueError("Invalid organization mode. Choose 'type' or 'size'.")

      if not os.path.exists(target_dir):
        os.makedirs(target_dir)
      shutil.move(file_path, target_dir + '/' + file)
      print(f"Moved '{file}' to directory '{target_dir}'.")
    except Exception as e:
      print(f"Error organizing '{file}': {e}")

# Get user input for path and organization mode
path = input("Enter path: ")
valid_modes = ["type", "size"]
while True:
  mode = input("Choose organization mode (type or size): ").lower()
  if mode in valid_modes:
    break
  else:
    print(f"Invalid mode. Please choose from {', '.join(valid_modes)}.")

# Call the organize_files function with user-provided options
if mode == "size":
  size_limit = int(input("Enter size limit in MB (defaults to 10): ") or 10) * 1024 * 1024
  organize_files(path, mode, size_limit)
else:
  organize_files(path, mode)  # No size_limit needed for type mode
```

Dialog box: After running the code:



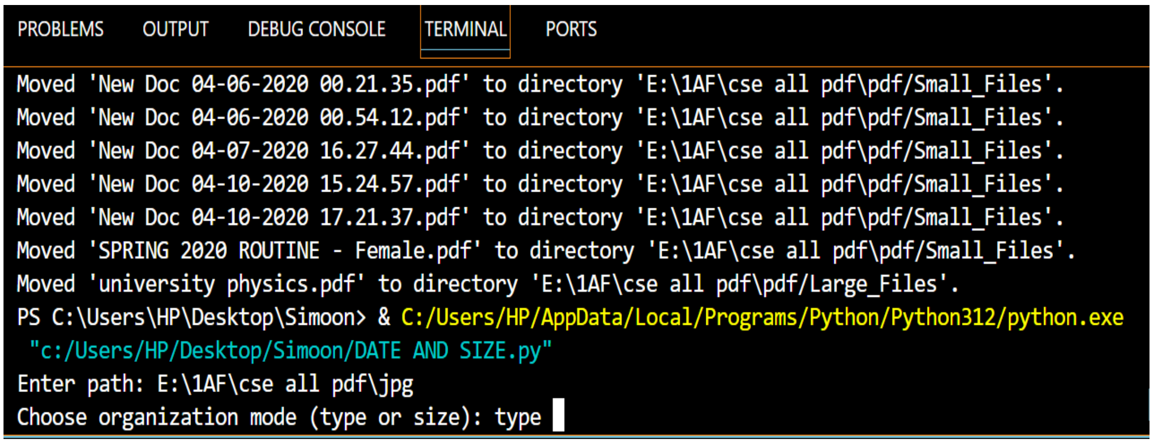
```
AND SIZE.py"'
PS C:\Users\HP\Desktop\Simoon> & C:/Users/HP/AppData/Local/Programs/Python/Python312/python.exe
 "c:/Users/HP/Desktop/Simoon/DATE AND SIZE.py"
Enter path: E:\1AF\cse all pdf
Choose organization mode (type or size):
```

Folder Before any mode:



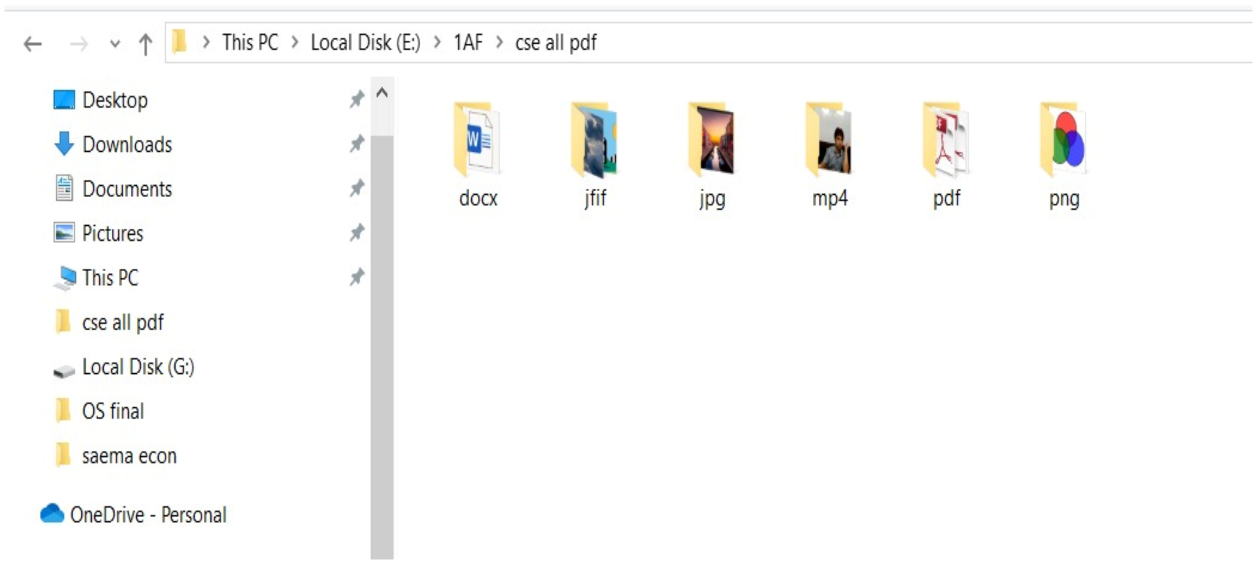
```
1st_semester_alexander_sadiku_fundamentals_of_electric_circuits_4thed
12_A_Streetcar_Named_Desire_Free_Sample (1)
324CSE-Spring-2020-1st-Semester-Class-Routine
2911028289
boylstad-introductory-circuit-analysis
C201212
C201212_Simoon nahar_1af
C201212_Simoon Nahar_Math_1106
C201212_Simoon_Nahar_1AF_UREL_1106
CamScanner 04-11-2020 17.17.32
CamScanner 04-13-2020 12.27.18
CSE_Syllabus-Spring 2018 (1)
CSE_Syllabus-Spring 2018
CSE-Spring-2020-Online-Class-Routine-Female
CSE-Spring-2020-Teacher-List-Contact-Zoom-ID
D. S. Mathur--Elements of properties of matter
download (1)
download (2)
download (3)
download (4)
download (5)
download
EEE-1121-Basic_Electrical Engineering-Lecture-Plan
EEE-1121-Basic_Electrical Engineering-Lecture-Plan
Exercises in Reading Comprehension by E.L. Tibbitts (1)
Exercises in Reading Comprehension by E.L. Tibbitts
Fundamentals of Electric Circuits (5th Ed)(gnv64) (1)
Halliday - Fundamentals of Physics Extended 9th-HQ
images
jpg_44-2
Level_10_Passage_3
New Doc 04-06-2020 00.21.35
New Doc 04-06-2020 00.54.12
New Doc 04-07-2020 16.27.44
New Doc 04-10-2020 15.24.57
New Doc 04-10-2020 17.21.37
SPRING 2020 ROUTINE - Female
university physics
```

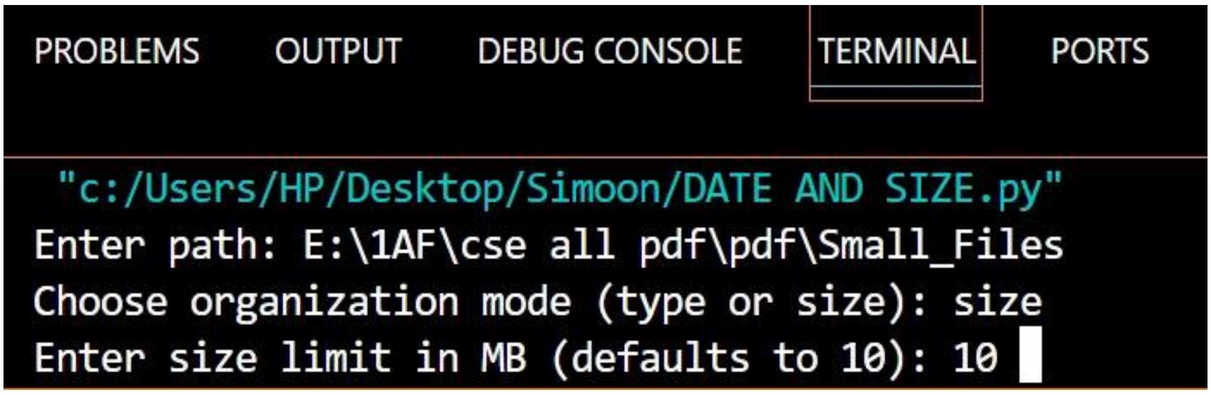
After selecting type mode:



```
Moved 'New Doc 04-06-2020 00.21.35.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'New Doc 04-06-2020 00.54.12.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'New Doc 04-07-2020 16.27.44.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'New Doc 04-10-2020 15.24.57.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'New Doc 04-10-2020 17.21.37.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'SPRING 2020 ROUTINE - Female.pdf' to directory 'E:\1AF\cse all pdf\pdf/Small_Files'.
Moved 'university physics.pdf' to directory 'E:\1AF\cse all pdf\pdf/Large_Files'.
PS C:\Users\HP\Desktop\Simoon> & C:/Users/HP/AppData/Local/Programs/Python/Python312/python.exe
 "c:/Users/HP/Desktop/Simoon/DATE AND SIZE.py"
Enter path: E:\1AF\cse all pdf\jpg
Choose organization mode (type or size): type
```

After selecting size mode:

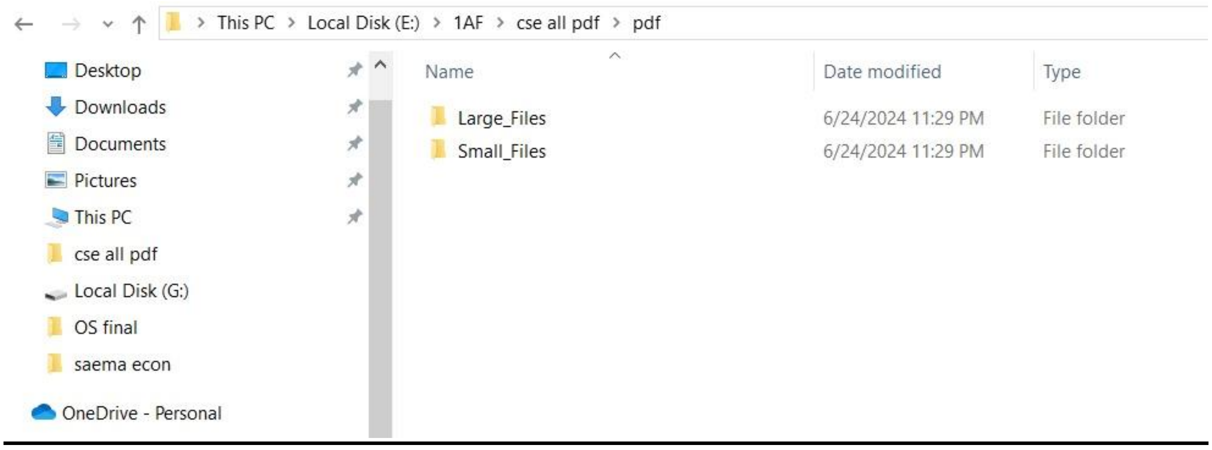




```
 "c:/Users/HP/Desktop/Simoon/DATE AND SIZE.py"
Enter path: E:\1AF\cse all pdf\pdf\Small_Files
Choose organization mode (type or size): size
Enter size limit in MB (defaults to 10): 10
```

Size limit =10 MB

Here bellow 10 mb stored in small files, and others are in large file



Name	Date modified	Type
Large_Files	6/24/2024 11:29 PM	File folder
Small_Files	6/24/2024 11:29 PM	File folder

This concludes the code for the file organization script. It retrieves user input, performs organization based on the chosen mode, and handles potential errors during file operations.

CONCLUSION:

This Python script successfully automates file organization within a directory. Users choose between sorting files by type (extension) or size (large vs. small). It offers basic error handling and guides users through the process. This project serves as a foundation for building a robust file organization tool while offering valuable learning opportunities in computer science concepts and Python programming. By understanding and potentially modifying this script, you can gain practical skills in file management automation and enhance your understanding of core OS functionalities and scripting principles.

There are some Future Enhancements:

- **More Modes:** The script can be extended to include additional organization modes based on date, keywords, or custom criteria.
- **Configuration Options:** Users could have options to define custom size thresholds or preferred subdirectory structures.
- **Graphical User Interface (GUI):** A GUI could be developed to provide a more user-friendly experience for those less comfortable with command-line interfaces.
- **Integration with Cloud Storage:** The script could be adapted to work with cloud storage services for organizing files stored online.

This Python project within the OS domain automates file organization by type or size, offering basic error handling and user interaction. It provides a foundation for building a more powerful organizer and learning OS concepts through Python.

APENDIX:

CODE	DESCRIPTION
import os	This library provides functions for interacting with the operating system's file system (listing files, getting file size, creating directories, moving files).
import shutil	This library offers functions for high-level file operations.
def organize_files(path, mode="type", size_limit=10 * 1024 * 1024):	This defines a function called organize_files. It takes three arguments: path: The path to the directory containing the files. mode: The organization mode ("type" or "size"). size_limit: The size threshold in bytes (used only in size mode, default: 10MB).
files = os.listdir(path) for file in files:	These lines retrieve a list of all filenames within the specified directory (path) using os.listdir(path). The for loop iterates through each filename (file) in the retrieved list.
file_path = os.path.join(path, file)	This line constructs the complete file path by joining the directory path (path) with the current filename (file) using os.path.join. This creates the absolute path for each file.
try:	It checks the organization mode (mode)

if mode == "type":
    # Get file extension
    filename, extension =os.path.splitext(file)
    extension = extension[1:].lower()
#Convert to lowercase
    target_dir = path + '/' + extension
	If "type":
	• Uses os.path.splitext(file) to separate the filename and extension.
	• Converts the extension to lowercase using string slicing ([1:]) and .lower().
	• Constructs the target directory path

	by combining the original path (path) with the lowercase extension.

`elif mode == "size":`
`# Get file size`
`file_size = os.path.getsize(file_path)`
`if file_size < size_limit:`
`target_dir = path + '/Small_Files'`
`else:`
`target_dir = path + '/Large_Files'`
	If "size":
	• Gets the file size using os.path.getsize(file_path).
	• Checks if the file size is less than the size_limit.
	o If yes, set the target directory to "Small_Files" subdirectory within the path.
	o If no, sets the target directory to "Large_Files" subdirectory within the path.

`else:`
`raise ValueError("Invalid organization mode. Choose 'type' or 'size'.")`
	If the mode is invalid (not "type" or "size"), it raises a ValueError with an informative message.

`if not os.path.exists(target_dir):`
`os.makedirs(target_dir)`
`shutil.move(file_path, target_dir + '/' + file)`
`print(f"Moved '{file}' to directory '{target_dir}'.")`
	It checks if the target directory (target_dir) doesn't already exist using not os.path.exists(target_dir).
	If it doesn't exist, it creates the subdirectory using os.makedirs(target_dir). This ensures the subdirectories ("Small_Files" and "Large_Files" for size-based organization, or extension-based subdirectories for type-based organization) are created before attempting to move files.
	It attempts to move the file to the target directory using shutil.move(file_path, target_dir + '/' + file). This efficiently moves the file from its current location to the designated subdirectory.
	If the move is successful, it prints a message indicating the file name and the target directory it was moved to.

`except Exception as e:`
`print(f"Error organizing '{file}': {e}")`
	The except block catches any exceptions (Exception is a general exception class) that might occur during file operations

`path = input ("Enter path: ")`	It prompts the user to enter the directory path (path) using input.
`valid_modes = ["type", "size"]`	It defines a list of valid organization modes (valid_modes).

`while True:`
`mode = input("Choose organization mode (type or size): ").lower()`
	It uses a while loop to ensure the user enters a valid mode ("type" or "size").
	It prompts the user to choose the organization mode (mode) and converts it to lowercase.

`if mode in valid_modes:`
`break`
	If valid, the loop breaks and execution continues.

else:
    print (f"Invalid mode. Please choose from {', ‘. join(valid_modes)}.")
	If invalid, it displays an error message prompting the user to choose from the valid options.

if mode == "size":
  size_limit = int(input("Enter size limit in MB (defaults to 10): ") or 10) * 1024 * 1024
  organize_files(path, mode, size_limit)
	If "size", it prompts the user for the size limit in MB (defaults to 10) and converts it to bytes before calling the function with path, mode, and size_limit.

else: organize_files(path, mode) # No size_limit needed for type mode	If "type", it directly calls the function with path and mode (no size limit needed).

Reference:

1. https://www.youtube.com/watch?v=KBjBPQExJLw
2. File Systems in Operating System, 24 Jun, 2024, geeksforgeeks, https://www.geeksforgeeks.org/file-systems-in-operating-system/