NEAR EAST UNIVERSITY

FILE ORGANIZER PROJECT

PRESENTED BY:
Jedidiah M. Katabana: 20233370
Denis K. Anderson: 20234528
Joe C. Mahundi: 20235140

Introduction

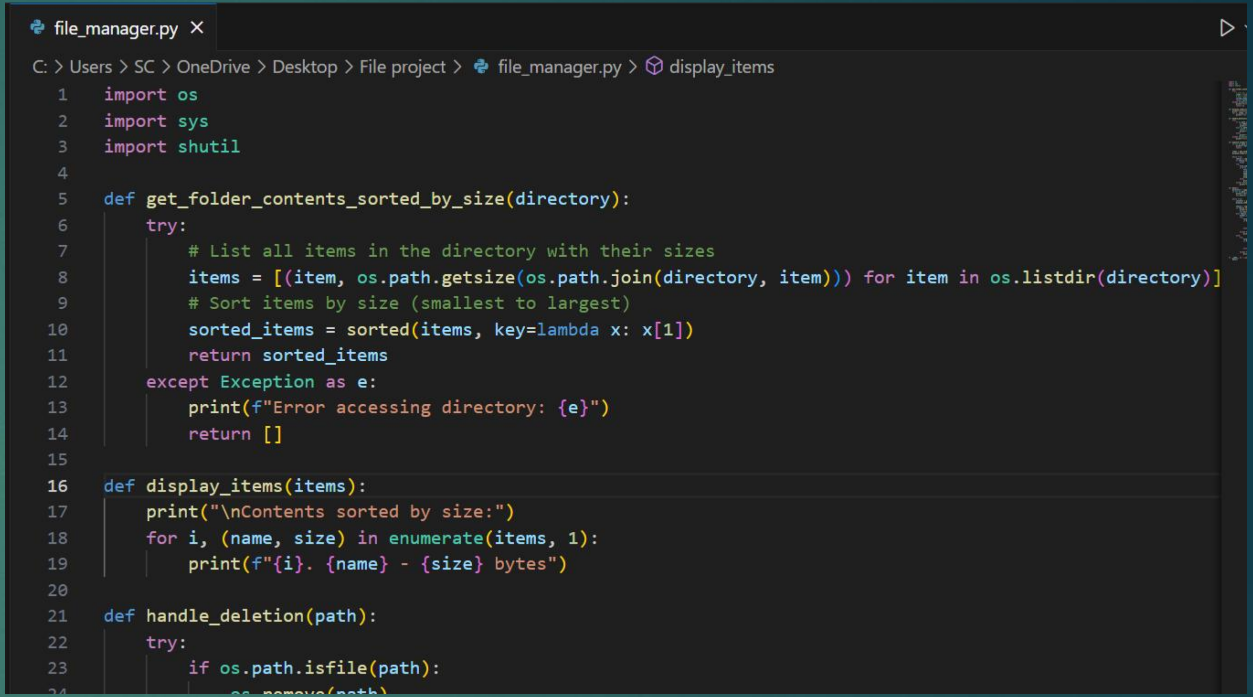
File organizers in Python refer to scripts or applications designed to streamline the storage, retrieval, and management of digital files. These tools leverage Python’s simplicity and powerful libraries to automate file organization processes. This report explores the type benefits, and best practices associated with Python-based file organizers.

Types of file organizer

1. **Simple File Sorting Scripts**

. Python scripts that categorize files based on their extensions, such as grouping images, documents, and videos into separate folders.

Example: Using “os” and “shutil” libraries to move files based on predefined rules.

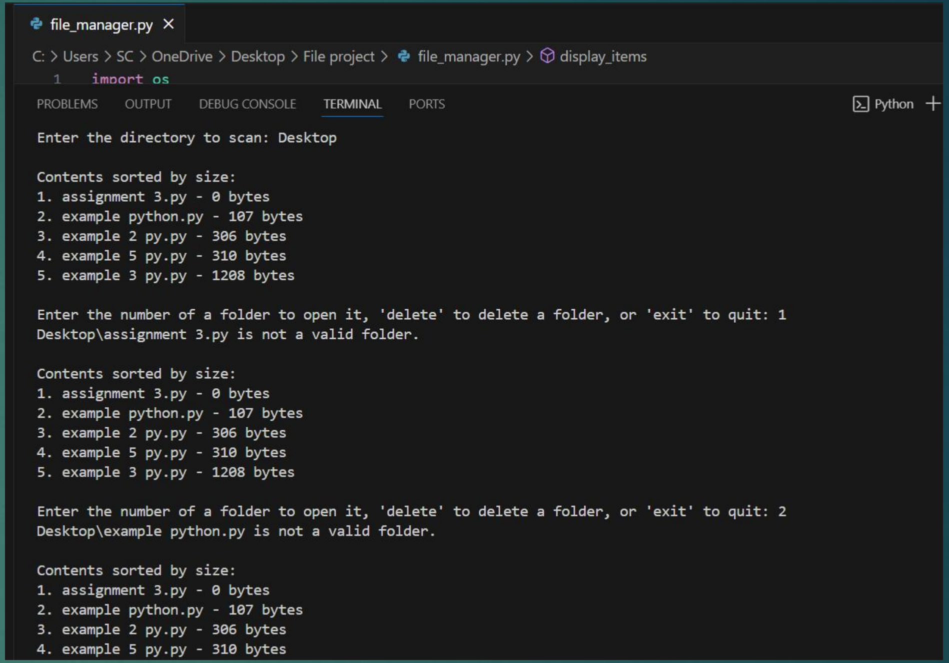

file_manager.py
C: > Users > SC > OneDrive > Desktop > File project > file_manager.py > display_items

```python
import os
import sys
import shutil

def get_folder_contents_sorted_by_size(directory):
    try:
        # List all items in the directory with their sizes
        items = [(item, os.path.getsize(os.path.join(directory, item))) for item in os.listdir(directory)]
        # Sort items by size (smallest to largest)
        sorted_items = sorted(items, key=lambda x: x[1])
        return sorted_items
    except Exception as e:
        print(f"Error accessing directory: {e}")
        return []

def display_items(items):
    print("\nContents sorted by size:")
    for i, (name, size) in enumerate(items, 1):
        print(f"{i}. {name} - {size} bytes")

def handle_deletion(path):
    try:
        if os.path.isfile(path):
            os.remove(path)
```

2. Advanced File Management Tools

. Programs with graphical user interfaces (GUIs) built using libraries like “tkinter”, “PyQt”, or “Kivy” for intuitive file organization.

Example: A tool that allows users to sort, search, and rename files interactively.

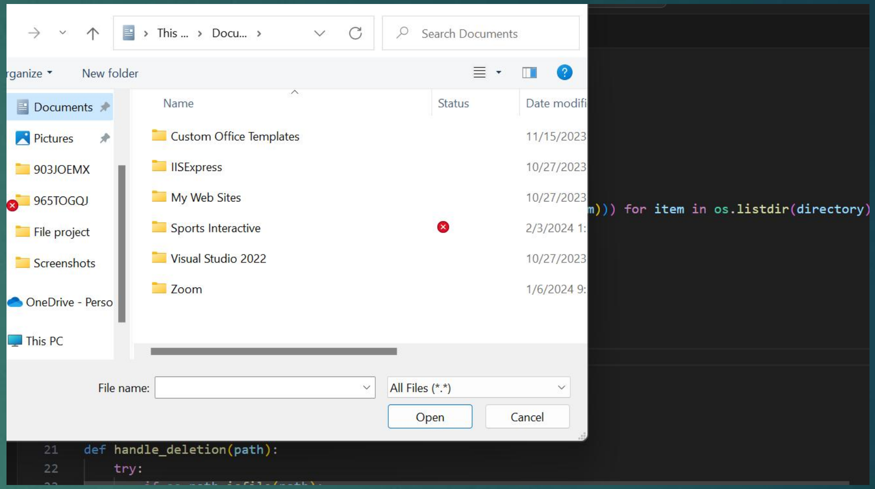

file_manager.py
C: > Users > SC > OneDrive > Desktop > File project > file_manager.py > display_items
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python

```
import os
```

```
Enter the directory to scan: Desktop

Contents sorted by size:
1. assignment 3.py - 0 bytes
2. example python.py - 107 bytes
3. example 2 py.py - 306 bytes
4. example 5 py.py - 310 bytes
5. example 3 py.py - 1208 bytes

Enter the number of a folder to open it, 'delete' to delete a folder, or 'exit' to quit: 1
Desktop\assignment 3.py is not a valid folder.

Contents sorted by size:
1. assignment 3.py - 0 bytes
2. example python.py - 107 bytes
3. example 2 py.py - 306 bytes
4. example 5 py.py - 310 bytes
5. example 3 py.py - 1208 bytes

Enter the number of a folder to open it, 'delete' to delete a folder, or 'exit' to quit: 2
Desktop\example python.py is not a valid folder.

Contents sorted by size:
1. assignment 3.py - 0 bytes
2. example python.py - 107 bytes
3. example 2 py.py - 306 bytes
4. example 5 py.py - 310 bytes
```

3. Cloud-Based File Organizers

. Python scripts that integrate with cloud storage APIs like Google Drive(using “pydrive” or “google-api-python-client”) or Dropbox(using “dropbox” library) for organizing files in the cloud.



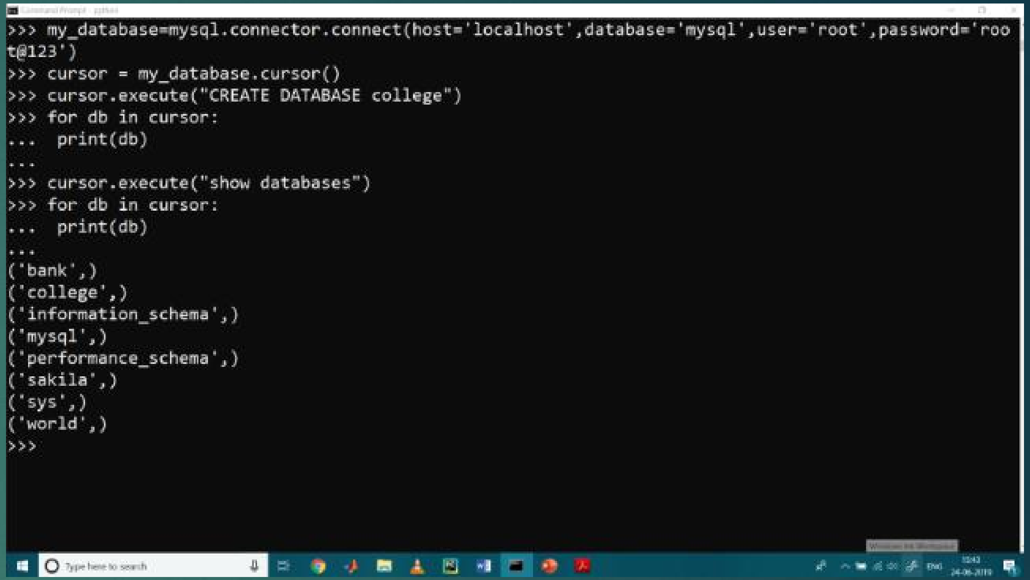
```
m))) for item in os.listdir(directory)
```

```
def handle_deletion(path):
    try:
```

4. Database-Integrated File Systems

. Applications that store metadata about files in databases like SQLite, enabling advanced search and categorization capabilities.

. Example: Using “sqlite3” to maintain a database of file locations, types, and tags.



```
>>> my_database=mysql.connector.connect(host='localhost',database='mysql',user='root',password='roo
t@123')
>>> cursor = my_database.cursor()
>>> cursor.execute("CREATE DATABASE college")
>>> for db in cursor:
...  print(db)
...
>>> cursor.execute("show databases")
>>> for db in cursor:
...  print(db)
...
('bank',)
('college',)
('information_schema',)
('mysql',)
('performance_schema',)
('sakila',)
('sys',)
('world',)
>>>
```

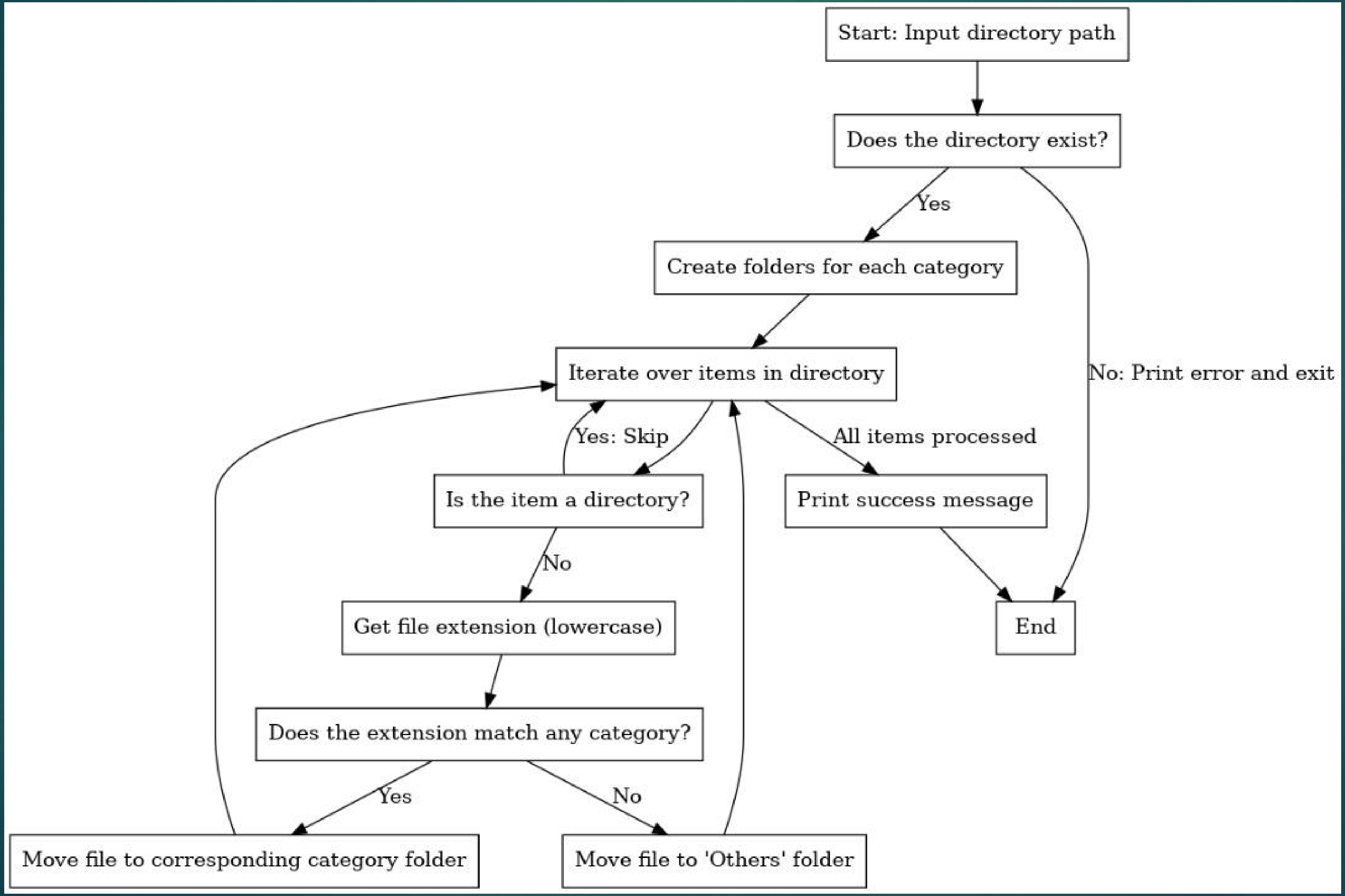
Benefits of Python-Based File Organizers

1. **Automation:** Reduces manual effort by automating repetitive tasks like file sorting and renaming.
2. **Customization:** Python's flexibility allows users to tailor solutions to their specific needs.
3. **Integration:** Seamlessly integrates with other systems, APIs, and platforms.
4. **Scalability:** Python scripts can handle both small-scale and large-scale file management tasks.
5. **Open Source:** Many Python libraries and tools are free to use, reducing development costs.

Best Practices for Python File Organization Projects

1. **Use Standard Libraries:** Leverage “os”, “shutil” and “pathlib” for file and directory operations.
2. **Implement Error Handling:** Anticipate issues like missing files or permission errors using “try-except” blocks.
3. **Adopt Modular Design:** Organize code into functions or classes for better readability and reusability.
4. **Incorporate Logging:** Use the “logging” module to track operations and debug issues effectively.
5. **Test Thoroughly:** Ensure scripts handle edge cases, such as duplicate filenames or unsupported file types.

Flow chart diagram


Start: Input directory path
Does the directory exist?
Yes
No: Print error and exit
Create folders for each category
Iterate over items in directory
Yes: Skip
All items processed
Is the item a directory?
Print success message
No
End
Get file extension (lowercase)
Does the extension match any category?
Yes
No
Move file to corresponding category folder
Move file to 'Others' folder

Overview

This Python script is a directory management tool that allows users to:

1. View the contents of a directory sorted by size.
2. Navigate directories interactively.
3. Delete files or folders.

It uses the os, shutil, and sys modules for file system operations, directory traversal, and program control.

Function analysis

1. get_folder_contents_sorted_by_size(directory)

- Retrieves all items in the given directory and their sizes using os.listdir() and os.path.getsize().
- Sorts the items by size using sorted() with a lambda function.
- Handles errors like invalid paths or access issues with a try-except block.

2. display_items(items)

- Loops through the list of items and prints their names and sizes in a user-friendly, numbered format.

3. handle_deletion(path)

- Deletes files with os.remove() or folders with shutil.rmtree().
- Ensures proper error handling for missing paths or permission issues.

4. explore_folder(folder_path)

- Validates the folder path.
- Displays contents sorted by size using get_folder_contents_sorted_by_size() and display_items().
- Allows users to delete files interactively or return to the parent directory.

5. main()

- Serves as the program’s entry point, validating the directory provided by the user.
- Provides an interactive loop to explore directories, delete folders, or exit the program.

Key features

- Sorting by Size: Makes it easy to identify large files or folders.
- Interactive Navigation: Allows users to navigate through subdirectories dynamically.
- Deletion Functionality: Simplifies directory cleanup by enabling file and folder deletions.
- Error Handling: Ensures the program doesn’t crash when encountering invalid inputs or paths

Limitations

1. No Confirmation for Deletion: Files or folders are deleted immediately without prompting for confirmation.

2. No Undo Feature: Deletions are permanent, with no way to recover removed items.

3. Performance: Sorting file sizes can be slow for directories with a large number of files.

Conclusion

- This script is a robust and user-friendly directory management tool. It is suitable for users looking to clean up and organize their file systems. However, adding features like deletion confirmation, undo functionality, and filtering options would enhance its usability and safety.