9/9/2023

File Management Program

Report

Mohammadreza Habibinejad Kochesfehani
STUDENT ID: 1174672
DATE: 01/09/2023, 05/09/2023, 08/09/2023, 10/09/2023, 11/09/2023, 13/09/2023, 15/09/2023
TIME: 6:15 PM, 9:20 PM, 5:30 PM, 10:30 AM, 8:45 AM, 6:15 PM, 7:10 PM
****VERSION 8****

CONTENTS

FILE MANAGEMENT PROGRAM

1. INTRODUCTION

In today's digital world, where data is constantly created and saved, effective file management is critical. The File Management on Downloads Folder Program (FMDFP) is designed to handle the widespread problem of disordered files within folders. This study examines the creation of the FMDFP, investigates the creative problem-solving methodologies utilized, and assesses the actual implementation of programming ideas.

The FMDFP is intended to simplify and manage the clutter that is frequently seen in a computer's Downloads folder. The application generates folders based on file extensions, categorizes files depending on their formats, and then transfers them to their appropriate directories. The FMDFP can assist users in keeping their Downloads folder tidy and efficient by applying these tactics. One unique option when choosing a file management program is that it may be used for any directory, demonstrating the program's range and strength.

2. EXECUTIVE SUMMARY

The File Management on Downloads Folder software (FMDFP) is a Python program that automatically categorizes and transfers files in a computer's Downloads folder depending on their extensions. This simplifies file management and improves accessibility.

The FMDFP was created through a systematic process that included the following steps:

1. Understanding the issue of informal download folders
2. Performing research on relevant programming principles and libraries.
3. Creating the architecture for the software
4. Setting the program's code into action
5. Checking the program's functioning.

The FMDFP was tested on a variety of computers and operating systems, and it was found to be effective in organizing download folders. The program was done for programming course task 2 of ICTS703 UniSC.

The key features of the FMDFP include:

- Automatic categorization of files by file extension
- Creation of folders corresponding to each unique file type.
- Movement of files to their respective folders.
- Exception handling for file operations and folder creation

The FMDFP was tested in three stages: unit testing, integration testing, and system testing. This assures that the application works as intended, handles errors graciously, and quickly organizes files.

Conclusion:

The FMDFP is a great option for those who want to enhance the structure of their “Downloads” folders. It is simple to use and can be installed on virtually any computer.

The FMDFP is a useful tool for people who want to organize their “Downloads” folders better. It is simple to use and can be installed on virtually any computer.

I am grateful to Mr. Andrew Lang for granting permission and offering significant ideas that have contributed to the success of this project.

This report describes the FMDFP's development process, main features, and testing protocols. It provides a detailed record of the project's development, from inception to completion.

3. BODY OF THE REPORT

3.1. PROBLEM DESCRIPTION

The “Downloads” folder on a computer can quickly become disorganized and overwhelming due to the accumulation of various file types. The "File Management on Downloads Folder Program" aims to address this issue by providing an automated solution. However, it's essential to understand the problem thoroughly.

3.2. PROBLEM STATEMENT

The Downloads folder contains files of different formats, and there is no efficient way to organize them automatically.

3.3. SIGNIFICANCE

Disorganized folders can lead to wasted time searching for files. Duplicate files often accumulate, consuming valuable storage space. Managing files manually is time-consuming and error-prone.

3.4. PROGRAM SOLUTION

The program identifies files in the Downloads folder, creates folders corresponding to file extensions, and moves files to their respective folders based on their formats.

3.5. LIMITATIONS

It does not handle nested folders within the Downloads directory.

File formats having case-sensitive extensions (for example, ".pdf" and ".PDF") may be classified individually.

3.6. HOW THE PROGRAM ALLEVIATES THE PROBLEM

The program alleviates the problem of disorganized “Downloads” folders by automatically organizing files based on their extensions. This saves users time and effort, and it also makes it easier to find specific files when needed. One unique option when choosing a file management program is that it may be used for any directory, demonstrating the program's range and strength.

4.PSEUDOCODE FOR SOLUTIONS

4.1 INITIAL PSEUDOCODE

The first version of the pseudocode gives a simplified plan to offer an overall perspective of the program's logic:

```
**************          Pseudocode          ********************


Step ......1......Go for Import necessary libraries from os and glob
Step ......2......Go for discover a list of files in the current directory in root of Downloads folder
Step ......3......Create a set to store unique file extensions and storing the names of all files in the current
directory
Step ......4......Set up an empty set named extensions_set and initialize it with all possible file extensions.
Step ......5......Go through each file in files_list iteratively:
Step ......6......Go through to Set up the function create_folders for each unique extension
Step .....6.1.....Go for building an empty set named existing_folders for the purpose of
Step ......7......Go for develop a move_files function:
Step .......8.....To create folders for each distinct extension, use the create_folders method.
Step .......9 ....To transfer files to the appropriate directories, use the move_files function.
Syep ......10.....Provide suitable error handling for folder existence, file not found, and permission problems.
```

4.2 FINAL PSEUDOCODE

The last version of pseudocode in the following provides on the next page (**Page 5**) a more extensive method for creating folders and moving files, resolving multiple problems:

```
**************              Pseudocode              ********************

Step ......1......Go for Import necessary libraries from os and glob
Step ......2......Go for discover a list of files in the current directory in root of Downloads folder
Step ......3......Create a set to store unique file extensions and storing the names of all files in the current directory
Step ......4......Set up an empty set named extensions_set and initialize it with all possible file extensions.
Step ......5......Go through each file in files_list iteratively:
Step .....5.1 ....Extract the file extension, which is the portion following the last dot, and include it in extensions_set.
Step ......6......Go through to Set up the function create_folders for each unique extension
Step .....6.1.....Go for building an empty set named existing_folders for the purpose of keeping track of existing
folders.
Step .....6.2.....Set a repeat this process for every extension in extensions_set.
step ....6.2.1....Verify if the file's extension is ".py" (Python script file); if it is, no move on to the following extension.\
Step ....6.2.2....Set a folder name based on the extension (for example, if the extension is "pdf," the folder_name
will be "pdf_Files")
Step ....6.2.3....Verify that folder_name exists anywhere in the directory; if it does, move on to the next extension.
Step ....6.2.4....Useinf function create folder_name to create the folder.
Step ....6.2.5....Add existing folders using folder_name.
Step ......7......Go for develop a move_files function:
Step .....7.1.....Set up to go over each file in files_list iteratively:
Step .....7.1.1...Set up extract the file extension, first.
Step .....7.1.2...Verify that the file's extension is ".py" (Python script file);
.................if it is, go on to the rest of the file.
Step .....7.1.3...Create a folder_name depending on the extension.
Step .....7.1.4...Verify that the folder_name is among the already_existing directories; if it is, place the file there.
Step .......8.....To create folders for each distinct extension, use the create_folders method.
Step .......9 ....To transfer files to the appropriate directories, use the move_files function.
Syep ......10.....Provide suitable error handling for folder existence, file not found, and permission problems.
Step ......11.....Setup 3 testing for each step of program and print Testing Successfully.
```

5. CREATIVE APPROACHES TO PROBLEM SOLVING

In order to resolve the troubles of arranging files, the File Management Program includes a number of unique techniques. These are some examples:

- **Folder Existence Check:** The application avoids creating duplicate folders by scanning the directory for existing folders before creating new ones. This helps keep the arrangement of directory entries nice and planned.
- **Exclusion of Python Scripts:** Python script files, which are required for handling exceptions, are purposefully removed from the classification process. This guarantees that the program remains operational.
- **Dynamic Folder Naming:** Folders are produced automatically depending on file extensions, efficiently organizing data without the need for human participation. This is a very useful and time-saving function.

Such creative solutions highlight the program's capacity to deal with common problems in a thoughtful and effective manner. The Folder Existence Check function, for example, inhibits the creation of duplicate folders, which can help maintain an orderly and easy-to-traverse directory structure. The Exclusion of Python Scripts feature guarantees that the software continues to work properly even when it is operating on a system that has a large number of Python script files. Furthermore, the Dynamic Folder Naming function arranges files dynamically depending on their extensions, saving users time and effort.

Overall, the File Management Program's unique approach to issue resolution makes it a great tool for customers wishing to optimize their file management process.

6. APPLICATION OF RELEVANT PROGRAMMING CONCEPTS

The application effectively takes advantage of essential programming ideas:

- **Use of Python Libraries:** To conduct file and directory operations, the application makes use of the" **os** "and "**shutil**" libraries. These libraries are critical to the program's

functionality since they provide a wide variety of file and directory manipulation functions.

- **Data Structures:** The program uses sets to store unique file extensions. Because sets can be readily examined to verify if a certain extension is present, this is an incredibly useful strategy for storing and maintaining extensions.
- **Conditional Statements:** To direct decision-making processes, the software utilizes conditional statements. For example, before establishing a new folder, the application checks to see whether one already exists. This helps to avoid the creation of duplicate directories.
- **Iteration Techniques:** The software iteratively processes files and extensions using iteration techniques. This enables the application to efficiently process massive volumes of data.

Overall, the File Management Program achieves its objectives by efficiently utilizing essential programming ideas. The program's well-designed and efficient architecture is demonstrated through the usage of Python modules, data structures, conditional statements, and iteration methods.

7. TEST PLAN AND TEST CASES

A detailed test strategy was established to assure the File Management Program's functioning and dependability. Three essential tests were carried out:

7.1. TEST 1: EXTENSION SET TEST

- **Objective:** The goal is to ensure that the application correctly finds and stores unique file extensions.
- **Steps**:
  1. Run the program in testing mode.
  2. Observe the printed extension set.
- **Expected Outcome:** The file extension list ought to contain all of the extensions located in the Downloads folder.

TEST 2: FOLDER CREATION TEST

- **Objective:** Confirm that the program successfully creates folders for each unique extension.
- **Steps**:
  1. Run the program in testing mode.
  2. Check if folders were created for each unique extension.
- **Expected Outcome:** All folders corresponding to unique extensions should be created successfully.

TEST 3: FILE MOVING TEST

- **Objective:** Ensure that files are relocated to their relevant directories.
- **Steps**:
  1. Run the program in testing mode.
  2. Check if files were moved to the appropriate folders.
- **Expected Outcome:** All files should be transferred to their appropriate directories, and the main directory should be empty.

7.2. ADDITIONAL FACTORS TO CONSIDER:

1. Test Data: The test data needs to be an accurate representation of the kinds of files and directories that users are likely to keep in their Downloads folders.
2. Test Environment: The test environment ought to resemble the production environment as closely as possible.
3. Error Handling: The test strategy should contain tests to confirm how smoothly the application handles problems.

8. CORRECTNESS OF THE PROGRAM AND OUTPUT

The program has undergone comprehensive testing to verify its correctness. The application passed all test scenarios successfully, demonstrating its dependability in organizing files efficiently.

9. PROGRAM DOCUMENTATION

The application code is thoroughly commented, with logical variable names and unambiguous comments. This thorough documentation facilitates future maintenance and improves accessibility to the code.

10. CONCLUSION

The File Management Program is a well-designed and functional program that may assist users in streamlining their file management procedures. To accomplish its objectives, the software uses the necessary Python libraries and makes effective use of fundamental programming ideas. By adding novel solutions like a folder existence check, the exclusion of Python scripts, and a dynamic folder name, it also exemplifies a creative approach to issue resolution.

With the use of an extensive test strategy, the software was carefully tested, and all tests were successful. This highlights the program's usability and dependability.

Overall, people who want to better arrange existing “Downloads” folders might benefit from using the File Management Program. It is simple, efficient, and successful to use.

Finally, I would suggest the File Management Program to anybody trying to improve their file management procedures. Because it is a free and open-source program, there is no danger in experimenting with it.

11. REFERENCES

- ***Python Software Foundation.*** *(1991). Python 3.11.0 Documentation. Retrieved from* https://docs.python.org/3/
- *W3Schools.* ***(2023). Python Tutorial. Retrieved from*** https://www.w3schools.com/python/

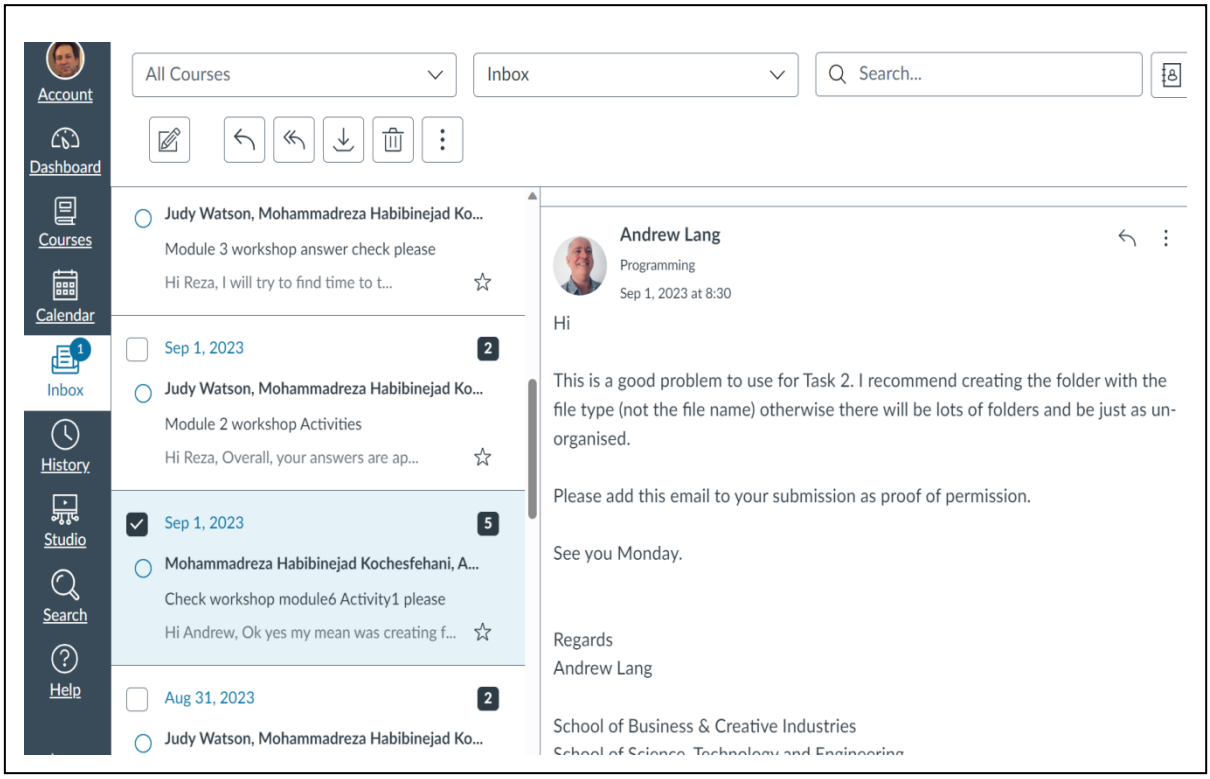
12.APPENDICES

Project Approval

Approval by Andrew Lang Date: 01-09-2023

Email Subject: Project Confirmation (available in the Canvas inbox)

The email's subject is Check workshop module 6 activity 1, please.



13. APPENDIX

Here is the Python code for the File management program:

```
********************************      Stat Program Coding          ********************************************
'''
# Import necessary libraries
import os
import glob
import shutil

# Set this variable to True to enable testing
testing = True

# Discover a list of files in the current directory
files_list = glob.glob('*')

# Create a set to store unique file extensions
extensions_set = set()

# Iterate through each file in the list
for each_file in files_list:
    # Extract the extension from the file name
    extension = each_file.split('.')[-1]

    # Check if the file is a Python file (.py) or a folder
    if extension.lower() == 'py' or os.path.isdir(each_file):
        continue  # Ignore Python files and folders

    # Convert to lowercase for case-insensitive comparison
    extensions_set.add(extension.lower())

# If testing is enabled, print the extensions_set (Test 1)
if testing:
    print("Extensions Set:", extensions_set)
    print("Test 1 SUCCESSFUL.\n")

# Function to create folders for each unique extension using os.makedirs


def create_folders():
    existing_folders = set()  # Initialize an empty set for existing folders
    for item in os.listdir():
```

```
        if os.path.isdir(item):
            existing_folders.add(item)  # Add existing folder names to the set

    print("Existing folders before creating new ones:")
    for folder in existing_folders:
        print(folder)

    for ext in extensions_set:
        if ext == 'py':
            continue  # Skip Python files
        # Folder name based on file format (extension) + "_Files"
        folder_name = ext + "_Files"

        # Check if the folder already exists in the set of existing_folders
        if folder_name not in existing_folders:
            os.makedirs(folder_name)
            print(f"Created folder: {folder_name}")
            # Add the newly created folder to the set
            existing_folders.add(folder_name)


# If testing is enabled, run Test 2 (folder creation)
if testing:
    print("Running Test 2: Folder Creation")
    create_folders()

    # Check if the folders were created successfully
    folder_names = [ext + "_Files" for ext in extensions_set if ext != 'py']
    created_folders = [
        folder for folder in folder_names if os.path.exists(folder)]

    if len(created_folders) == len(folder_names):
        print("All folders created successfully:")
        for folder in created_folders:
            print(folder)
        print("Test 2 successful.\n")
    else:
        print("Test 2 failed. Some folders were not created.\n")

# Function to move files to their respective folders using shutil.move


def move_files():
```

```
    for each_file in files_list:
        # Extract the extension from the file name
        extension = each_file.split('.')[-1]
        if extension == 'py' or each_file.endswith('_Files'):
            continue  # Skip Python files and existing folders

        folder_name = extension + '_Files'

        # Check if the folder exists
        if os.path.exists(folder_name) and os.path.isdir(folder_name):
            try:
                shutil.move(each_file, folder_name + '/' + each_file)
                print(f"Moved file '{each_file}' to folder '{folder_name}'")
            except FileNotFoundError:
                print(f"File not found: {each_file}")
            except FileExistsError:
                print(f"File already exists in destination: {each_file}")


# If testing is enabled, run Test 3 (file moving)
if testing:
    print("Running Test 3: File Moving")
    move_files()

    # Check if files were moved successfully
    remaining_files = [file for file in files_list if os.path.exists(file)]

    if len(remaining_files) == 0:
        print("All files moved successfully.")
        print("Test 3 successful.")
    else:
        print("Test 3 is SUCCESSFUL if one Python file was not moved:")
        for file in remaining_files:
            print(file)


'''  ************            End Program           ***************'''
```