**Code Overview**

This Python script is designed to find duplicate PDF files in a specified directory by comparing their content. It uses hashing and text extraction techniques to identify duplicates and logs the results to a text file.

**Code**

import os

import hashlib

import PyPDF2

import difflib

def file\_hash(filepath):

    with open(filepath, 'rb') as f:

        return hashlib.md5(f.read()).hexdigest()

def extract\_text\_from\_pdf(pdf\_path):

    text = ""

    with open(pdf\_path, 'rb') as file:

        reader = PyPDF2.PdfFileReader(file)

        for page\_num in range(reader.numPages):

            page = reader.getPage(page\_num)

            text += page.extract\_text()

    return text

def are\_texts\_similar(text1, text2, threshold=0.8):

    return difflib.SequenceMatcher(None, text1, text2).ratio() > threshold

def file\_hash\_or\_text(filepath):

    try:

        # Try to hash the file

        return file\_hash(filepath)

    except:

        # If hashing fails, extract text from PDF and hash the text

        if filepath.lower().endswith('.pdf'):

            text = extract\_text\_from\_pdf(filepath)

            return hashlib.md5(text.encode()).hexdigest()

        else:

            raise ValueError("Unsupported file format")

def find\_duplicates(directory):

    hashes = {}

    duplicates = []

    for root, dirs, files in os.walk(directory):

        for file in files:

            filepath = os.path.join(root, file)

            filehash = file\_hash\_or\_text(filepath)

            if filehash in hashes:

                duplicates.append((filepath, hashes[filehash]))

            else:

                hashes[filehash] = filepath

    return duplicates

def log\_duplicates\_to\_file(duplicates, log\_file\_path):

    with open(log\_file\_path, 'w') as log\_file:

        for dup in duplicates:

            log\_file.write(f"Duplicate file: {dup[0]} is a duplicate of {dup[1]}\n")

# Example usage

directory\_path = r'C:\Users\SYANTHA2\OneDrive - Province of Nova Scotia\Desktop\Python\Approval Required'

log\_file\_path = r'C:\Users|SYANTHA2\OneDrive - Province of Nova Scotia\Desktop\Python\Approval Required\duplicates\_log.txt'

duplicates = find\_duplicates(directory\_path)

log\_duplicates\_to\_file(duplicates, log\_file\_path)

print(f"Duplicate files have been logged to {log\_file\_path}")

**Detailed Explanation**

1. **Imports**

import os

import hashlib

import PyPDF2

import difflib

* + os: Provides a way to interact with the operating system, particularly for file and directory operations.
  + hashlib: Used for hashing files to generate unique hash values.
  + PyPDF2: A library for reading and extracting text from PDF files.
  + difflib: Provides tools for comparing sequences, used here for text similarity checks.

1. **Function: file\_hash**

def file\_hash(filepath):

with open(filepath, 'rb') as f:

return hashlib.md5(f.read()).hexdigest()

* + **Purpose**: Computes the MD5 hash of a file.
  + **Parameters**: filepath - the path to the file.
  + **Process**:
    - Opens the file in binary read mode ('rb').
    - Reads the entire file content.
    - Computes the MD5 hash of the file content.
    - Returns the hash as a hexadecimal string.
  + **Technique**: MD5 hashing is used to generate a unique identifier for the file content. MD5 is fast but not the most secure; however, it’s sufficient for detecting duplicates.

1. **Function: extract\_text\_from\_pdf**

def extract\_text\_from\_pdf(pdf\_path):

text = ""

with open(pdf\_path, 'rb') as file:

reader = PyPDF2.PdfFileReader(file)

for page\_num in range(reader.numPages):

page = reader.getPage(page\_num)

text += page.extract\_text()

return text

* + **Purpose**: Extracts text from a PDF file.
  + **Parameters**: pdf\_path - the path to the PDF file.
  + **Process**:
    - Opens the PDF file in binary read mode.
    - Uses PyPDF2 to read the PDF.
    - Iterates through each page and extracts text.
    - Concatenates the text from all pages and returns it.

1. **Function: are\_texts\_similar**

def are\_texts\_similar(text1, text2, threshold=0.8):

return difflib.SequenceMatcher(None, text1, text2).ratio() > threshold

* + **Purpose**: Compares two texts and determines if they are similar based on a threshold.
  + **Parameters**: text1, text2 - the texts to compare; threshold - the similarity threshold.
  + **Process**:
    - Uses difflib.SequenceMatcher to compute the similarity ratio between the two texts.
    - Returns True if the similarity ratio exceeds the threshold, otherwise False.

1. **Function: file\_hash\_or\_text**

def file\_hash\_or\_text(filepath):

try:

# Try to hash the file

return file\_hash(filepath)

except:

# If hashing fails, extract text from PDF and hash the text

if filepath.lower().endswith('.pdf'):

text = extract\_text\_from\_pdf(filepath)

return hashlib.md5(text.encode()).hexdigest()

else:

raise ValueError("Unsupported file format")

* + **Purpose**: Computes a hash for the file content or extracted text from a PDF.
  + **Parameters**: filepath - the path to the file.
  + **Process**:
    - Tries to hash the file directly.
    - If hashing fails (e.g., due to unsupported format), checks if the file is a PDF.
    - Extracts text from the PDF and hashes the text.
    - Returns the hash as a hexadecimal string.

1. **Function: find\_duplicates**

def find\_duplicates(directory):

hashes = {}

duplicates = []

for root, dirs, files in os.walk(directory):

for file in files:

filepath = os.path.join(root, file)

filehash = file\_hash\_or\_text(filepath)

if filehash in hashes:

duplicates.append((filepath, hashes[filehash]))

else:

hashes[filehash] = filepath

return duplicates

* + **Purpose**: Finds duplicate files in a directory.
  + **Parameters**: directory - the path to the directory to search.
  + **Process**:
    - Initializes two collections: hashes (a dictionary) and duplicates (a list).
    - Uses os.walk to traverse the directory tree.
    - For each file, computes its hash using file\_hash\_or\_text.
    - Checks if the hash is already in the hashes dictionary:
      * If yes, it means the file is a duplicate, and it adds the file paths (current and original) to the duplicates list.
      * If no, it adds the hash and file path to the hashes dictionary.
    - Returns the list of duplicates.

1. **Function: log\_duplicates\_to\_file**

def log\_duplicates\_to\_file(duplicates, log\_file\_path):

with open(log\_file\_path, 'w') as log\_file:

for dup in duplicates:

log\_file.write(f"Duplicate file: {dup[0]} is a duplicate of {dup[1]}\n")

* + **Purpose**: Logs duplicate file information to a text file.
  + **Parameters**: duplicates - the list of duplicate file pairs; log\_file\_path - the path to the log file.
  + **Process**:
    - Opens the log file in write mode.
    - Writes each pair of duplicate files to the log file.

1. **Example Usage**

directory\_path = r'I:\ICM Payments\APPROVAL REQUIRED'

log\_file\_path = r'I:\ICM Payments\APPROVAL REQUIRED\duplicate\_files\_log.txt'

duplicates = find\_duplicates(directory\_path)

log\_duplicates\_to\_file(duplicates, log\_file\_path)

1. print(f"Duplicate files have been logged to {log\_file\_path}")
   * **Purpose**: Demonstrates how to use the functions to find and log duplicate files.
   * **Process**:
     + Sets the directory path to search for duplicates.
     + Sets the log file path to save the duplicate information.
     + Calls find\_duplicates to find duplicates in the specified directory.
     + Calls log\_duplicates\_to\_file to log the duplicates to the specified log file.
     + Prints a message indicating where the duplicates have been logged.

**Techniques Involved**

* **Hashing**: Uses MD5 hashing to generate unique identifiers for file content. MD5 is chosen for its speed, though it’s not the most secure.
* **Text Extraction**: Uses PyPDF2 to extract text from PDF files, allowing for content-based comparison.
* **Text Similarity**: Uses difflib.SequenceMatcher to compare the similarity of text content, which can be useful for identifying near-duplicates.

## **Instructions to Run the Script**

1. **Install Python**:

- Ensure Python is installed on your computer. You can download it from python.org.

2. **Set Up a Virtual Environment**:

- Open a terminal or command prompt.

- Navigate to the directory where you saved the script and `requirements.txt`.

- Create a virtual environment:

*```bash*

python -m venv myenv

*```*

- Activate the virtual environment:

- On Windows:

*```bash*

myenv\Scripts\activate

*```*

- On macOS/Linux:

*```bash*

source myenv/bin/activate

*```*

3. **Install Dependencies**:

- Install the required packages using `requirements.txt`:

*```bash*

pip install -r requirements.txt

*```*

4. **Run the Script**:

- Run the Python script:

*```bash*

python your\_script.py

*```*

Replace `your\_script.py` with the name of your Python file (*FindDuplicates.py*).