CAPSTONE P-5

O(1)lympians

Section B

Table of Contents

ABOUT US

Team ID: O(1)lympians

Group Members:

1) Manya Shah (202301424)
2) Manit Shah (202301425)
3) Sharnam Shah (202301247)
4) Jainiesh Patel (202301446)

Capstone Idea: P5 {Folder Cleaner}

In this question we were given a CSV file, in which the file path, the date of creation of the files and the frequency i.e the number of times the file was opened. Using this data we were required to do the following task:

1. Delete Redundant files (i.e. a newer copy is present)
2. Delete files older than N number of months
3. Delete Empty files
4. Delete files that have not been accessed for at least M number of times.

Github Repository Link:

https://github.com/202301247/O-1-lympians__P5.git

Explanation of Our code for the four tasks:

The Question consists of doing 4 tasks:
1. Redundant files (i.e., a newer copy is present),
2. Files old for more than N number of months,
3. Empty files (i.e., no content),
4. Files that have not been accessed for at least M number of times.

Do Please Read How to Open and Run our files.

Pseudocodes:

1. Redundant files:

Here, we try to see whether any two files have the exact same content in them. If yes then one of the files will get deleted.

Approach 1: The code could have been written such that we can take for example the first file and then check it with the rest of the files. Thus a total of $nC2$ operations, but with a large number of rifles, this would take up a lot of time.

Approach 2: We create a hash map, and store the string with the filenames it, and then delete the files having the same key (hash value for the string content) . Also a new csv file is created in which the remaining files are put in and is then replaced with the original files.

Time Complexity of the code is: O(n+m*n+m*k)
Space Complexity of the code is: O(n)
Here n is the number of lines in the input file
Here m is the average number of files with the same content
Here k is the average number of files within each group

Code:

```
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>
#include <unordered_map>
#include <vector>
#include <cstring>
#include <string>
#include <bits/stdc++.h>

using namespace std;

// Function to calculate the hash value of a string
size_t hashString(const string& str) {
   hash<string> hasher;
   return hasher(str);
}

// Function to compute the hash value of a file
size_t computeFileHash(const string& filename) {
   ifstream file(filename);
   stringstream buffer;
   buffer << file.rdbuf();
   string content = buffer.str();
   return hashString(content);
}

// Function to group files based on their content
unordered_map<size_t, vector<string>> groupFilesByContent(const vector<string>&
filenames) {
   unordered_map<size_t, vector<string>> groups;
   for (const auto& filename : filenames) {
       size_t hash = computeFileHash(filename);
       groups[hash].push_back(filename);
   }
   return groups;
}


void redundant_delete()
{
   // Open FIle pointers
   fstream fin, fout;

   // Open the existing file
   fin.open("file_data_csv.csv", ios::in);

   // Create a new file to store the non-deleted data
   fout.open("new.csv", ios::out);


   int row_size;
   string line, word, word2, temp;
   vector<string> filenames;
   map<string, string> mp;

   while (getline(fin, line)) {

       stringstream s(line);

       getline(s, word, ',');
       getline(s, word2);

       mp[word]=word2;

       filenames.push_back(word);
   }
   unordered_map<size_t, vector<string>> fileGroups =
groupFilesByContent(filenames);

   for (const auto& pair : fileGroups) {
       const vector<string>& files = pair.second;
       cout << "Group with same content:\n";
       cout << files[0] << endl;

       temp=files[0];
       fout << temp << ',' << mp[temp] <<endl;


       for (int i = 1; i < files.size(); i++){
           temp=files[i];
           cout << files[i] << endl;
           const int length = temp.length();
           char* char_array = new char[length + 1];
           strcpy(char_array, temp.c_str());
           remove(char_array);
       }
   }

   fin.close();
   fout.close();

   // removing the existing file
   remove("file_data_csv.csv");

   // renaming the new file with the existing file name
   rename("new.csv", "file_data_csv.csv");
}


int main(){
   redundant_delete();
}
```

2.File Old for more than N number of months

Here, we try to see if a file is older than N number of months then we delete the files.

{In the code we have taken N = 6, for convince}

Approach: As the data for the time when the file was created is given in the excel/csv file, we take the data from their and then delete the files if it is old for more than N number of months

Time Complexity of the code is: O(m)
Space Complexity of the code is: O(m*n)
Here n is the average number of fields per line
Here m represents the number of lines in the file

Code:

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <ctime>
#include <fstream>
#include <cstring>
#include <string>
#include <bits/stdc++.h>

using namespace std;

// Function to parse date and time string into time_t
time_t parseDateTime(const string& datetime) {
   tm tm = {};
   istringstream ss(datetime);
   ss >> get_time(&tm, "%T %d %m %Y"); // HH:MM:SS DD MM YYYY format
   if (ss.fail()) {
       throw runtime_error("Failed to parse date and time.");
   }
   return mktime(&tm);
}

// Function to calculate the difference in days between two time_t values
int daysDifference(time_t start, time_t end) {
   return static_cast<int>((end - start) / (60 * 60 * 24)); // Convert seconds
to days
}

void n_months_old()
{
   time_t now = time(nullptr);
   // Open FIle pointers
   fstream fin, fout;

   // Open the existing file
```

```cpp
   fin.open("file_data_csv.csv", ios::in);

   // Create a new file to store the non-deleted data
   fout.open("new.csv", ios::out);


   int N_months=6, access_times, row_size, N_days=6*30;
   string line, word;

   vector<string> row;


   while (getline(fin, line)) {

       row.clear();

       stringstream s(line);

       while (getline(s, word, ',')) {
           row.push_back(word);
       }

       int row_size = row.size();
       string file_path(row[0]);
       access_times = stoi(row[2]);

       string file_datetime = row[1];
       time_t file_time = parseDateTime(file_datetime);
       int days_old = daysDifference(file_time, now);

       //string to char array
       const int length = file_path.length();
       char* char_array = new char[length + 1];
       strcpy(char_array, file_path.c_str());


       if (days_old>N_days) {
           cout << "File " << file_path << " is more than " << N_months << "
old.\n";
           remove(char_array);
       }
       else {
           for (int i = 0; i < row_size - 1; i++) {
               fout << row[i] << ",";
           }
           fout << row[row_size - 1] << "\n";
       }
   }
   fin.close();
   fout.close();

   // removing the existing file
   remove("file_data_csv.csv");

   // renaming the new file with the existing file name
   rename("new.csv", "file_data_csv.csv");
}

int main(){
   n_months_old();
}
```

3. Empty files

Here if the file is found empty (i.e no content is found in the file)

Approach : We take the file location from the excel/CSV file and then delete the file if it is empty and the non-empty files are then added to a new csv and renamed to replace the original csv file.

Time Complexity of the code is: O(n)
Space Complexity of the code is: O(m) where m is the longest line of the file

Code:

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <ctime>
#include <fstream>
#include <cstring>
#include <string>
#include <bits/stdc++.h>

using namespace std;

bool isFileEmpty(const string& filename) {
```

```cpp
   ifstream file(filename);
   return file.peek() == ifstream::traits_type::eof();
}

void empty_delete()
{
   // Open FIle pointers
   fstream fin, fout;

   // Open the existing file
   fin.open("file_data_csv.csv", ios::in);

   // Create a new file to store the non-deleted data
   fout.open("new.csv", ios::out);

   string line, file_path, word2;

   vector<string> row;


   while (getline(fin, line)) {

       row.clear();

       stringstream s(line);

       getline(s, file_path, ',');
       getline(s, word2);

       //string to char array
       const int length = file_path.length();
       char* char_array = new char[length + 1];
       strcpy(char_array, file_path.c_str());


       if (isFileEmpty(file_path)) {
           cout << "File " << file_path << " is empty\n";
           remove(char_array);
       }
       else {
           fout << file_path << ',' << word2 << endl;
       }
   }

   fin.close();
   fout.close();

   // removing the existing file
   remove("file_data_csv.csv");
   // renaming the new file with the existing file name
   rename("new.csv", "file_data_csv.csv");
}

int main(){
   empty_delete();
}
```

4. Files that have not been accessed for at least M number of times.

Here we try to delete a file which hasn’t been accessed for at least M number of times. {we have taken M as 6, i.e. if a file isn’t accessed for at least 6 times then it gets deleted.}

Approach:As the number of times the file has been opened would be given in the csv file, we take the data form the csv file itself and create a new csv file in which the remaining files would be stored and would replace the original csv file.

Time Complexity of the code is: O(n*L)
Space Complexity of the code is: O(L)
Here n is the number of lines in the code and L is the average length of each line

Code:

```
#include <iostream>
#include <fstream>
#include <cstring>
#include <string>
#include <bits/stdc++.h>

using namespace std;

void unaccessed_mtimes()
{
```

```cpp
// Open FIle pointers
fstream fin, fout;

// Open the existing file
fin.open("file_data_csv.csv", ios::in);

// Create a new file to store the non-deleted data
fout.open("new.csv", ios::out);


int M=6, access_times, row_size;
string line, word;

vector<string> row;


while (getline(fin, line)) {

    row.clear();

    stringstream s(line);

    while (getline(s, word, ',')) {
        row.push_back(word);
    }

    int row_size = row.size();
    string file_path(row[0]);
    access_times = stoi(row[2]);

    //string to char array
    const int length = file_path.length();
    char* char_array = new char[length + 1];
    strcpy(char_array, file_path.c_str());


    if (access_times < M) {
        cout << "File " << file_path << " has been accessed less than " << M
<< " times.\n";
        remove(char_array);
    }
    else {
        for (int i = 0; i < row_size - 1; i++) {
            fout << row[i] << ",";
        }
        fout << row[row_size - 1] << "\n";
    }
}
```

```
   fin.close();
   fout.close();

   // removing the existing file
   remove("file_data_csv.csv");

   // renaming the new file with the existing file name
   rename("new.csv", "file_data_csv.csv");
}

int main(){
   unaccessed_mtimes();
}
```

GUI: Python Tkinter

Here we use Python Tkinter for the GUI, we used a template available on the internet and modified it ourselves to optimize it for our folder cleaner so as it can handle our code files and we used a subprocess for running the code written in c++ and added buttons, backgrounds etc.

Note:

Download the following modules to run the GUI and ultimately the entire code for our project:
Enter the following in your command prompt:

pip install pillow

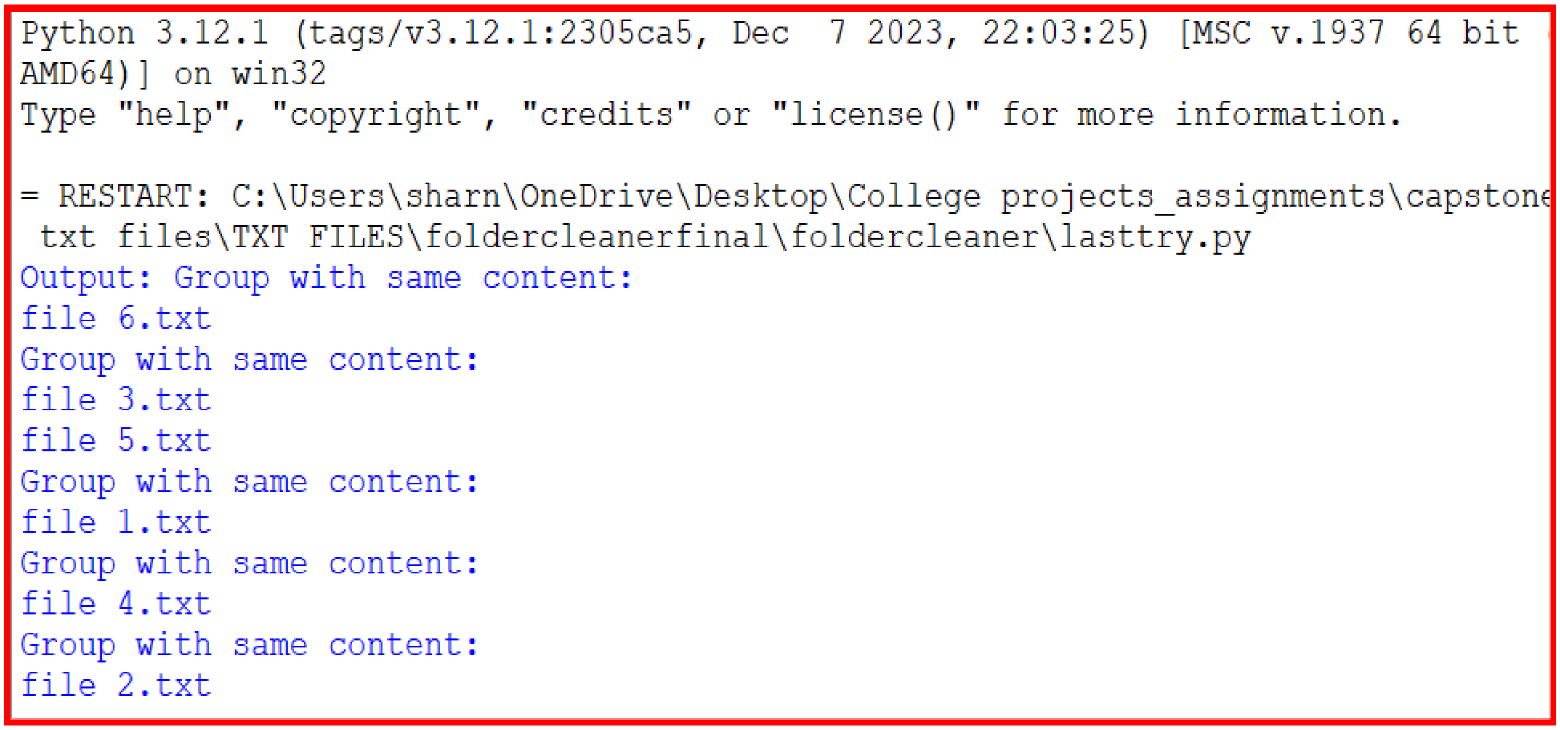
CODE:

```
import tkinter as tk
from tkinter import filedialog, PhotoImage
import subprocess
from PIL import Image, ImageTk

import tkinter as tk
from tkinter import filedialog
import subprocess

def delete_duplicates():
   # Call your C++ function here
   import subprocess
   # compile the C++ code
   subprocess.call(["g++", "redundant.cpp"])
   # run the compiled code
   output = subprocess.check_output("./a.exe")
   # print the output
   print("Output:", output.decode())


def delete_empty():
   # Call your C++ function here
   import subprocess
   # compile the C++ code
   subprocess.call(["g++", "empty_file_check.cpp"])
   # run the compiled code
   output = subprocess.check_output("./a.exe")
   # print the output
   print("Output:", output.decode())

def delete_time():
   # Call your C++ function here
   import subprocess
   # compile the C++ code
   subprocess.call(["g++", "n_months.cpp"])
   # run the compiled code
   output = subprocess.check_output("./a.exe")
   # print the output
   print("Output:", output.decode())

def delete_unaccessed():
   # Call your C++ function here
   import subprocess
   # compile the C++ code
   subprocess.call(["g++", "m_times.cpp"])
   # run the compiled code
   output = subprocess.check_output("./a.exe")
   # print the output
   print("Output:", output.decode())





def start_foldcleanernx():
   foldcleanernx_window = tk.Toplevel(foldercleaner)
   foldcleanernx_frame = tk.Frame(foldcleanernx_window, bg='light blue')
   foldcleanernx_frame.pack(fill='both', expand=True)
```

```
   header = tk.Label(foldcleanernx_frame, text="Folder Cleaner",
font=("Helvetica", 16, "bold"), bg='light blue')
   header.pack(pady=10)

   button1 = tk.Button(foldcleanernx_frame, text="Delete Duplicate Files",
command=delete_duplicates)
   button1.pack()

   button2 = tk.Button(foldcleanernx_frame, text="Delete Empty Files",
command=delete_empty)
   button2.pack()

   button3 = tk.Button(foldcleanernx_frame, text="Delete Time Files",
command=delete_time)
   button3.pack()

   button4 = tk.Button(foldcleanernx_frame, text="Delete Unaccessed Files",
command=delete_unaccessed)
   button4.pack()

foldercleaner = tk.Tk()
foldercleaner.title("Folder Cleaner")

# Load your background image
image = Image.open("Bg for folder cleaner 6.png")
image = image.resize((foldercleaner.winfo_screenwidth(),
foldercleaner.winfo_screenheight()))
bg_image = ImageTk.PhotoImage(image)

# Create a canvas for your background image
canvas = tk.Canvas(foldercleaner, width=foldercleaner.winfo_screenwidth(),
height=foldercleaner.winfo_screenheight())
canvas.pack(fill='both', expand=True)
# Add the image to the canvas
canvas.create_image(0, 0, image=bg_image, anchor='nw')


# Add your text
canvas.create_text(foldercleaner.winfo_screenwidth()//2,
foldercleaner.winfo_screenheight()//2, text="Hello, we are O(1)lympians",
font=("Helvetica", 16, "bold"), justify='center', anchor='center')

# Create a button on the canvas to start the cleaner
start_button = tk.Button(foldercleaner, text="Start the Folder Cleaner",
command=start_foldcleanernx)
canvas.create_window(foldercleaner.winfo_screenwidth()//2,
foldercleaner.winfo_screenheight()//2 + 50, window=start_button)

foldercleaner.mainloop()
```

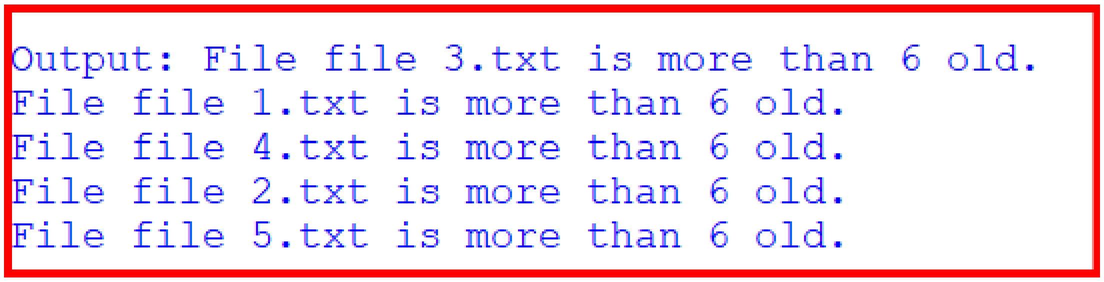
Outputs:

For Reductants files:



```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\sharn\OneDrive\Desktop\College projects_assignments\capstone
 txt files\TXT FILES\foldercleanerfinal\foldercleaner\lasttry.py
Output: Group with same content:
file 6.txt
Group with same content:
file 3.txt
file 5.txt
Group with same content:
file 1.txt
Group with same content:
file 4.txt
Group with same content:
file 2.txt
```

For File Old for more than N number of months:



```
Output: File file 3.txt is more than 6 old.
File file 1.txt is more than 6 old.
File file 4.txt is more than 6 old.
File file 2.txt is more than 6 old.
File file 5.txt is more than 6 old.
```

For Empty File:

```
Output: File file 6.txt is empty
```

For Unaccessed Files:

```
Output: File file 6.txt has been accessed less than 6 times.
```

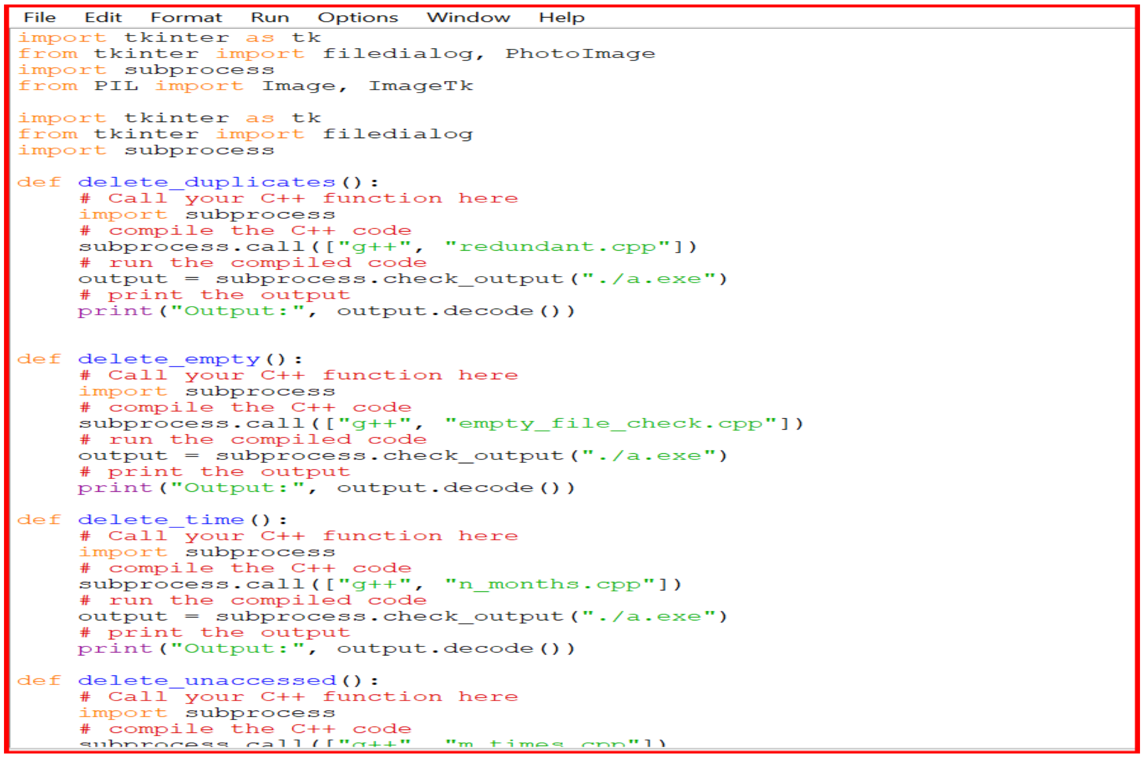
{This page ends here}

Running and opening our code:

Step 1: Download the zip file from our Github Repository.

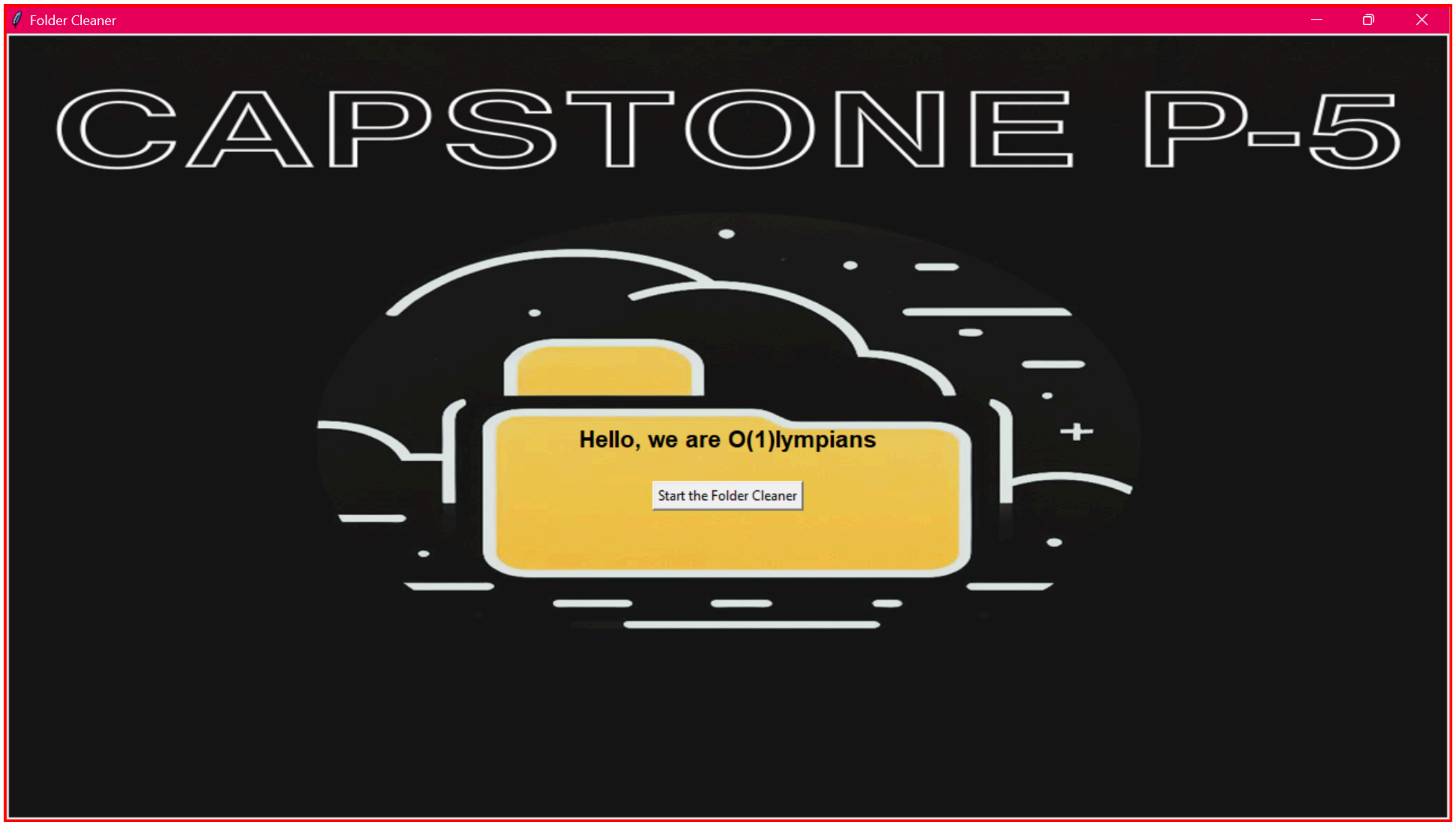
Step 2: Download python and download the modules given in GUI “NOTE” section. (i.e. pip install pillow)

Step 3: In python IDLE open the “lasttry” file



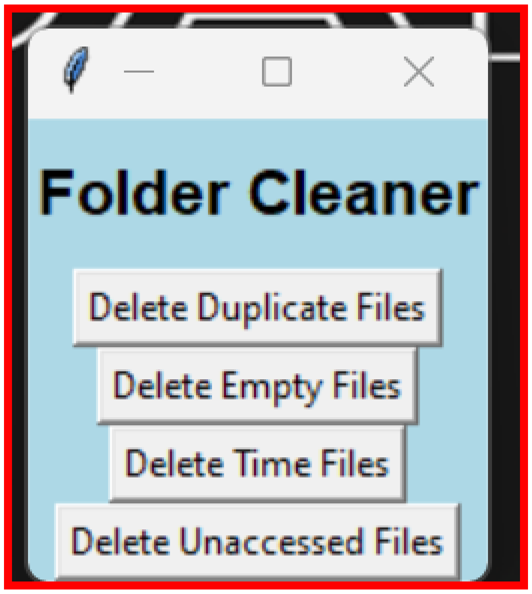
```
import tkinter as tk
from tkinter import filedialog, PhotoImage
import subprocess
from PIL import Image, ImageTk

import tkinter as tk
from tkinter import filedialog
import subprocess

def delete_duplicates():
    # Call your C++ function here
    import subprocess
    # compile the C++ code
    subprocess.call(["g++", "redundant.cpp"])
    # run the compiled code
    output = subprocess.check_output("./a.exe")
    # print the output
    print("Output:", output.decode())


def delete_empty():
    # Call your C++ function here
    import subprocess
    # compile the C++ code
    subprocess.call(["g++", "empty_file_check.cpp"])
    # run the compiled code
    output = subprocess.check_output("./a.exe")
    # print the output
    print("Output:", output.decode())

def delete_time():
    # Call your C++ function here
    import subprocess
    # compile the C++ code
    subprocess.call(["g++", "n_months.cpp"])
    # run the compiled code
    output = subprocess.check_output("./a.exe")
    # print the output
    print("Output:", output.decode())

def delete_unaccessed():
    # Call your C++ function here
    import subprocess
    # compile the C++ code
    subprocess.call(["g++", "m_times.cpp"])
```

Step 4: Now run the file and this window will pop up ⬇️

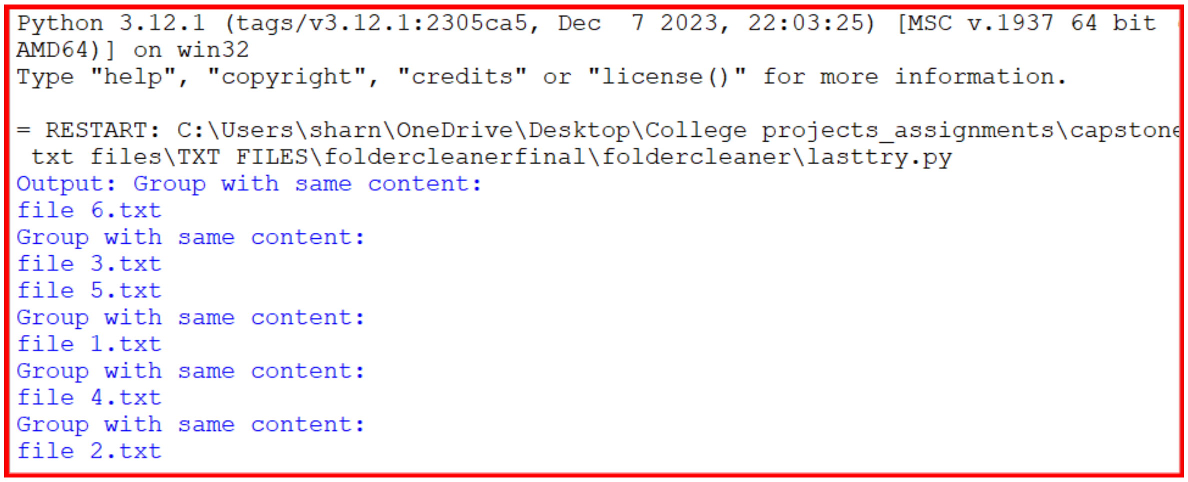


{This Page ends here}

Step 5: Click on “Start the Folder Cleaner”, you will see this when you do so ⬇️



Step 6: If we want to Delete Duplicate files, i.e. The Redundant files we click on that option and we get the following output ⬇️



```
Python 3.12.1 (tags/v3.12.1:2305ca5, Dec  7 2023, 22:03:25) [MSC v.1937 64 bit
AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.

= RESTART: C:\Users\sharn\OneDrive\Desktop\College projects_assignments\capstone
 txt files\TXT FILES\foldercleanerfinal\foldercleaner\lasttry.py
Output: Group with same content:
file 6.txt
Group with same content:
file 3.txt
file 5.txt
Group with same content:
file 1.txt
Group with same content:
file 4.txt
Group with same content:
file 2.txt
```

Step 7: Similarly if we want delete:

-**empty file** we get the following output ⬇️

```
Output: File file 6.txt is empty
```

-**Time files**, i.e. files which have **not been accessed for more than M number of times**, we get the following output ⬇️

```
Output: File file 6.txt has been accessed less than 6 times.
```

-**Unaccassed files**, i.e the Files which **haven't been accessed for more than N number of months** ⬇️

```
Output: File file 3.txt is more than 6 old.
File file 1.txt is more than 6 old.
File file 4.txt is more than 6 old.
File file 2.txt is more than 6 old.
File file 5.txt is more than 6 old.
```

{End of file, Thanks for reading- Team O(1)lympians 😀}