```
# Instalar librerías

import pandas as pd
import io
import os
import gspread

!pip install msoffcrypto-tool
import msoffcrypto
from openpyxl import load_workbook
```

```
Requirement already satisfied: msoffcrypto-tool in /usr/local/lib/python3.12/dist-packages (6.0.0)
Requirement already satisfied: cryptography>=39.0 in /usr/local/lib/python3.12/dist-packages (from msoffcrypto-tool) (43.0.3)
Requirement already satisfied: olefile>=0.46 in /usr/local/lib/python3.12/dist-packages (from msoffcrypto-tool) (0.47)
Requirement already satisfied: cffi>=1.12 in /usr/local/lib/python3.12/dist-packages (from cryptography>=39.0->msoffcrypto-tool)
Requirement already satisfied: pycparser in /usr/local/lib/python3.12/dist-packages (from cffi>=1.12->cryptography>=39.0->msoffc
```

```
# Acceder a google Drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
# Autenticar usuario de Google Colab

import gspread
from google.colab.auth import authenticate_user
from google.auth import default

authenticate_user()
print('Google Colab authentication successful.')

creds, _ = default()

gc = gspread.authorize(creds)
print('gspread initialized successfully.')
```

```
Google Colab authentication successful.
gspread initialized successfully.
```

```
# Definir rutas
base_path = '/content/drive/MyDrive/2026_PROYECTO'
ips_folder = os.path.join(base_path, 'ANEXOS_PROYECTO', 'ANEXO 4')
output_folder = os.path.join(base_path, 'ANEXOS_PROYECTO', 'ANEXO 7')
```

```
# Crear directorio de salida
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
```

```
# Archivos
file_names = [
    'ESTADO_FLUJO_EFECTIVO.xlsx',
    'ESTADO_RESULTADOS.xlsx',
    'ESTADO_SITUACION_FINANCIERA.xlsx'
    ]
```

```
# Hojas
sheet_names = [
    'TABLA_ER',
    'TABLA_EFE',
    'TABLA_ESF'
    ]
```

```
# Iniciar dataframes

consolidated_dfs = {
    'ESTADO_RESULTADOS': pd.DataFrame(),
```

```
    'ESTADO_FLUJO_EFECTIVO': pd.DataFrame(),
    'ESTADO_SITUACION_FINANCIERA': pd.DataFrame()
}
```

```
# Definir contraseña que desbloquea los archivos

password_archivos = "123"

def cargar_excel_dinamico(path_archivo, password, sheet_name):
    try:
        # Apertura como archivo protegido
        with open(path_archivo, "rb") as f:
            office_file = msoffcrypto.OfficeFile(f)
            office_file.load_key(password=password)
            decrypted = io.BytesIO()
            office_file.decrypt(decrypted)
            # Leer la hoja específica en un DataFrame después del descifrado
            df = pd.read_excel(decrypted, sheet_name=sheet_name)
    except Exception as e:
        if "File is not encrypted" in str(e) or "Unencrypted document" in str(e) or "Document is not encrypted" in str(e):
            # Apertura directa sin contraseña
            df = pd.read_excel(path_archivo, sheet_name=sheet_name)
        else:
            raise e
    return df
```

```
# Carpetas por años

years = [2020,2021,2022,2023,2024]

for year in years:
    year_folder = os.path.join(ips_folder, str(year))
    if not os.path.exists(year_folder):
        print(f"Folder for year {year} not found: {year_folder}")
        continue

    # Crea un diccionario para asignar los nombres de los archivos a sus respectivos nombres de hojas
    file_sheet_map = {
        'ESTADO_FLUJO_EFECTIVO.xlsx': 'TABLA_EFE',
        'ESTADO_RESULTADOS.xlsx': 'TABLA_ER',
        'ESTADO_SITUACION_FINANCIERA.xlsx': 'TABLA_ESF'
    }

    for file_name in file_names:
        file_path = os.path.join(year_folder, file_name)
        if not os.path.exists(file_path):
            print(f"File not found: {file_path}")
            continue

        try:
            # Obtener el nombre de hoja correcto para el archivo actual
            sheet_name = file_sheet_map[file_name]

            # Cargar datos - Lee solo la hoja relevante
            df = cargar_excel_dinamico(file_path, password_archivos, sheet_name)

            # Determinar cada hoja
            if sheet_name == 'TABLA_ER':
                statement_type = 'ESTADO_RESULTADOS'
            elif sheet_name == 'TABLA_EFE':
                statement_type = 'ESTADO_FLUJO_EFECTIVO'
            elif sheet_name == 'TABLA_ESF':
                statement_type = 'ESTADO_SITUACION_FINANCIERA'
            else:
                print(f"Unknown sheet name: {sheet_name} in file {file_name}")
                continue

            # Agregar año
            df['Year'] = year

            # Dataframe Consolidado
            consolidated_dfs[statement_type] = pd.concat([consolidated_dfs[statement_type], df], ignore_index=True)
```

```
        except Exception as e:
            print(f"Error processing file {file_path}: {e}")
```

```
# Crear un cliente de servicio de API de Drive

from googleapiclient.discovery import build

drive_service = build('drive', 'v3', credentials=creds)
print('Google Drive API service client initialized successfully.')
```

```
Google Drive API service client initialized successfully.
```

```
# Obtener el ID de la carpeta de salida
# Ajuste la ruta para que sea relativa a 'Mi unidad' de Google Drive del usuario

def get_folder_id(drive_service, target_folder_path):
    current_parent_id = 'root'
    # Remove leading/trailing slashes and split by '/' to get folder parts
    target_folder_path_parts = [part for part in target_folder_path.strip('/').split('/') if part]

    for folder_name in target_folder_path_parts:
        # Search for the folder by name within the current parent
        query = f"'{current_parent_id}' in parents and name = '{folder_name}' and mimeType = 'application/vnd.google-apps.folde
        response = drive_service.files().list(q=query, spaces='drive', fields='nextPageToken, files(id, name)').execute()
        files = response.get('files', [])

        if files:
            # Folder found, update current_parent_id
            current_parent_id = files[0]['id']
        else:
            # Folder not found, create it
            file_metadata = {
                'name': folder_name,
                'mimeType': 'application/vnd.google-apps.folder',
                'parents': [current_parent_id]
            }
            folder = drive_service.files().create(body=file_metadata, fields='id').execute()
            current_parent_id = folder.get('id')
            print(f"Created folder: {folder_name} with ID: {current_parent_id}")
    return current_parent_id

drive_relative_output_folder_path = output_folder.replace('/content/drive/MyDrive/', '', 1)
destination_drive_folder_id = get_folder_id(drive_service, drive_relative_output_folder_path)
print(f"Destination Google Drive Folder ID for '{output_folder}' (Drive relative path: '{drive_relative_output_folder_path}'):
```

```
WARNING:google_auth_httplib2:httplib2 transport does not support per-request timeout. Set the timeout when constructing the http
WARNING:google_auth_httplib2:httplib2 transport does not support per-request timeout. Set the timeout when constructing the http
Destination Google Drive Folder ID for '/content/drive/MyDrive/2026_PROYECTO/ANEXOS_PROYECTO/ANEXO 7' (Drive relative path: '202
```

```
# Definir la ruta de salida
output_folder = '/content/drive/MyDrive/2026_PROYECTO/ANEXOS_PROYECTO/ANEXO 7'

drive_relative_output_folder_path = output_folder.replace('/content/drive/MyDrive/', '', 1)
destination_drive_folder_id = get_folder_id(drive_service, drive_relative_output_folder_path)
print(f"Destination Google Drive Folder ID for '{output_folder}' (Drive relative path: '{drive_relative_output_folder_path}'):
```

```
Destination Google Drive Folder ID for '/content/drive/MyDrive/2026_PROYECTO/ANEXOS_PROYECTO/ANEXO 7' (Drive relative path: '202
```

```
google_sheet_links = []

for statement_type, df in consolidated_dfs.items():
    # a. Construir un nombre para la nueva hoja de cálculo de Google.
    new_spreadsheet_name = f'{statement_type}_CONSOLIDADO_SHEET'

    # b. Crear una nueva hoja de cálculo de Google
    spreadsheet = gc.create(new_spreadsheet_name)

    # c. Obtener la primera hoja de trabajo
    worksheet = spreadsheet.get_worksheet(0)

    # d. Convertir el DataFrame df a una lista de listas
    data_to_upload = [df.columns.values.tolist()] + df.values.tolist()

    # e. Cargar los datos en la hoja de trabajo
    worksheet.update(range_name='A1', values=data_to_upload)
```

```
        worksheet.update(range_name='A1', values=data_to_upload)

        # f. Obtener el ID de la hoja de cálculo creada
        spreadsheet_id = spreadsheet.id

        # g. Obtener el ID de la carpeta actual de la hoja de cálculo
        # Las hojas de cálculo creadas por gspread se colocan por defecto en la carpeta raíz.
        file = drive_service.files().get(fileId=spreadsheet_id, fields='parents').execute()
        previous_parents = ','.join(file.get('parents'))

        # h. Eliminar la hoja de cálculo de su carpeta actual (generalmente la raíz)
        # y moverla a la carpeta de destino especificada por destination_drive_folder_id.
        drive_service.files().update(
            fileId=spreadsheet_id,
            removeParents=previous_parents,
            addParents=destination_drive_folder_id,
            fields='id, parents'
        ).execute()

        # i. Añadir el ID de la hoja de cálculo a la lista google_sheet_links
        google_sheet_links.append(spreadsheet_id)

        print(f"Spreadsheet '{new_spreadsheet_name}' created and moved. ID: {spreadsheet_id}")

    print("All consolidated DataFrames have been uploaded to Google Sheets and moved to the destination folder.")
    print(f"Google Sheet IDs: {google_sheet_links}")
```

```
Spreadsheet 'ESTADO_RESULTADOS_CONSOLIDADO_SHEET' created and moved. ID: 1iPk-xwWBW3xIPqr9lq2HDFHwaU8mphMy7enxOcef0pE
Spreadsheet 'ESTADO_FLUJO_EFECTIVO_CONSOLIDADO_SHEET' created and moved. ID: 1FM2qPWkyyibC1SjgPRrz9n3ohV_LO7u2A0pAGUapa9c
Spreadsheet 'ESTADO_SITUACION_FINANCIERA_CONSOLIDADO_SHEET' created and moved. ID: 1RNgibJt3_jsGUsMaqfW_US1CaoMDblzwy__2GQWk3X8
All consolidated DataFrames have been uploaded to Google Sheets and moved to the destination folder.
Google Sheet IDs: ['1iPk-xwWBW3xIPqr9lq2HDFHwaU8mphMy7enxOcef0pE', '1FM2qPWkyyibC1SjgPRrz9n3ohV_LO7u2A0pAGUapa9c', '1RNgibJt3_js
```