6. **Automatización de Excel:** Script que lee y actualiza datos en una hoja de cálculo (`openpyxl` o `pandas`).

¡Claro! Puedo ayudarte con una aplicación en Python para la automatización de Excel.
Usaré la librería **openpyxl** que es ideal para leer y escribir en archivos Excel (.xlsx).

? Automatización de Excel con Python (openpyxl)

Este script realizará dos tareas principales:

1. **Leer** datos de una hoja de cálculo.
2. **Actualizar** un dato específico y guardar el archivo.

1. Requisitos

Asegúrate de tener la librería openpyxl instalada. Puedes instalarla usando pip:

```
pip install openpyxl
```

2. El Código Python

El siguiente código asume que tienes un archivo llamado **datos.xlsx** con una hoja llamada **Hoja1** y datos en las primeras columnas.

Python

```
import openpyxl
import os

def automatizar_excel(nombre_archivo="datos.xlsx"):
    """
    Lee y actualiza datos en un archivo Excel usando openpyxl.
    """

    # --- 1. Crear un archivo de prueba si no existe ---
    # Esto es solo para asegurar que el script se pueda ejecutar
    if not os.path.exists(nombre_archivo):
        print(f"El archivo {nombre_archivo} no existe. Creando uno de ejemplo...")
        try:
            workbook = openpyxl.Workbook()
            sheet = workbook.active
            sheet.title = "DatosVentas"

            # Encabezados
            sheet['A1'] = 'ID Producto'
            sheet['B1'] = 'Nombre'
            sheet['C1'] = 'Stock'

            # Datos de ejemplo
            sheet['A2'] = 101
            sheet['B2'] = 'Laptop'
            sheet['C2'] = 15

            sheet['A3'] = 102
            sheet['B3'] = 'Monitor'
            sheet['C3'] = 22
```

```
            workbook.save(nombre_archivo)
            print(f"Archivo de ejemplo '{nombre_archivo}' creado exitosamente.")
        except Exception as e:
            print(f"Error al crear el archivo: {e}")
            return

    # --- 2. Lectura de Datos ---
    try:
        print("\n### ? Leyendo datos de Excel ###")
        workbook = openpyxl.load_workbook(nombre_archivo)
        sheet = workbook['DatosVentas'] # Asegúrate de usar el nombre de hoja correcto

        # Iterar sobre las filas y mostrar los datos
        print("Contenido actual:")
        for row in sheet.iter_rows(values_only=True):
            print(row)

    except FileNotFoundError:
        print(f"Error: El archivo '{nombre_archivo}' no se encontró.")
        return
    except KeyError:
        print("Error: La hoja 'DatosVentas' no se encontró en el archivo.")
        return
    except Exception as e:
        print(f"Ocurrió un error al leer el archivo: {e}")
        return

    # --- 3. Actualización de Datos ---
    try:
        print("\n###  Actualizando datos de Excel ###")

        # Ejemplo: Actualizar el Stock (celda C2) del 'Laptop' (ID 101) a 5 unidades.
        celda_a_actualizar = 'C2'
        nuevo_valor = 5

        # Asignar el nuevo valor a la celda
        sheet[celda_a_actualizar] = nuevo_valor
        print(f"Celda {celda_a_actualizar} actualizada a: {nuevo_valor}")

        # Guardar los cambios
        workbook.save(nombre_archivo)
        print(f"Archivo '{nombre_archivo}' guardado exitosamente con las actualizaciones.")

        # Re-leer para confirmar el cambio
        print("\nContenido después de la actualización:")
        for row in sheet.iter_rows(values_only=True):
            print(row)

    except Exception as e:
        print(f"Ocurrió un error al actualizar el archivo: {e}")

# Ejecutar la función principal
automatizar_excel()
```

? Explicación del Código

- **import openpyxl**: Importa la librería necesaria para interactuar con Excel.
- **Creación del Archivo de Ejemplo**: La primera parte del código chequea si el archivo datos.xlsx existe. Si no, lo crea con datos básicos (ID Producto, Nombre, Stock) para que el script siempre sea ejecutable.
- **openpyxl.load_workbook(nombre_archivo)**: Carga el archivo de Excel existente en memoria.
- **workbook['DatosVentas']**: Selecciona la hoja de cálculo por su nombre.
- **sheet.iter_rows(values_only=True)**: Es la forma eficiente de leer todos los datos de la hoja fila por fila. values_only=True nos da directamente los valores, no los objetos de celda.
- **sheet['C2'] = nuevo_valor**: Esta es la sintaxis clave para la **escritura** (actualización). Simplemente accedes a la celda por su notación (ej: C2) y le asignas un nuevo valor.
- **workbook.save(nombre_archivo)**: **Guarda** todos los cambios de vuelta al archivo físico, sobrescribiendo la versión anterior con las modificaciones.
  ¿Te gustaría que adapte este script para que use **pandas** en su lugar, lo cual es mejor para manipulación y análisis de grandes volúmenes de datos?

¡Excelente! Aquí tienes el código para la **Automatización de Excel** usando la librería **pandas**, que es ideal para la manipulación avanzada de datos.

? Automatización de Excel con Python (pandas)

Pandas utiliza objetos llamados **DataFrames** para representar los datos, lo que facilita filtrar, modificar y realizar cálculos antes de guardar. Para trabajar con archivos Excel, también necesitaremos la librería openpyxl (que pandas usa internamente para la entrada/salida de Excel).

1. Requisitos

Asegúrate de tener ambas librerías instaladas:

```
pip install pandas openpyxl
```

2. El Código Python

Este script realizará las mismas tareas: **leer** datos de Excel, **actualizar** un valor en el DataFrame y **guardar** el resultado.

Python

```
import pandas as pd
import os

def automatizar_excel_pandas(nombre_archivo="datos_pandas.xlsx", nombre_hoja="Inventario"):
    """
    Lee, modifica y guarda datos en un archivo Excel usando pandas.
    """

    # --- 1. Crear un archivo de prueba si no existe ---
    if not os.path.exists(nombre_archivo):
        print(f"El archivo {nombre_archivo} no existe. Creando uno de ejemplo...")
        try:
            # Crear un DataFrame de ejemplo
            datos = {
                'ID Producto': [201, 202, 203],
                'Nombre': ['Teclado', 'Ratón', 'Webcam'],
```

```
            'Stock': [35, 40, 12],
            'Precio': [25.50, 15.00, 45.99]
        }
        df_ejemplo = pd.DataFrame(datos)

        # Guardar el DataFrame al archivo Excel
        df_ejemplo.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
        print(f"Archivo de ejemplo '{nombre_archivo}' creado exitosamente.")
    except Exception as e:
        print(f"Error al crear el archivo: {e}")
        return

# --- 2. Lectura de Datos ---
try:
    print("\n### ? Leyendo datos de Excel con Pandas ###")
    # Leer el archivo Excel en un DataFrame de Pandas
    df = pd.read_excel(nombre_archivo, sheet_name=nombre_hoja)

    print("Contenido actual del DataFrame:")
    # Mostrar el DataFrame, que es la representación tabular de los datos
    print(df)

except FileNotFoundError:
    print(f"Error: El archivo '{nombre_archivo}' no se encontró.")
    return
except Exception as e:
    print(f"Ocurrió un error al leer el archivo: {e}")
    return

# --- 3. Actualización de Datos (Manipulación del DataFrame) ---
try:
    print("\n###  Actualizando datos en el DataFrame ###")

    # 3.1. Encontrar la fila que queremos modificar (e.g., el producto 'Ratón')
    producto_a_modificar = 'Ratón'

    # 3.2. Usar .loc para encontrar la fila y actualizar la columna 'Stock'
    # La condición (df['Nombre'] == producto_a_modificar) devuelve True/False para cada fila
    # .loc localiza la(s) fila(s) donde es True, y luego la columna 'Stock'
    nuevo_stock = 50
    df.loc[df['Nombre'] == producto_a_modificar, 'Stock'] = nuevo_stock

    print(f"El Stock del producto '{producto_a_modificar}' ha sido actualizado a: {nuevo_stock}")

    # 3.3. Aplicar una operación a toda una columna (e.g., aumentar todos los precios un 10%)
    df['Precio'] = df['Precio'] * 1.10
    print("Todos los precios han sido incrementados en un 10%.")

except Exception as e:
    print(f"Ocurrió un error al manipular el DataFrame: {e}")
    return

# --- 4. Guardar los Cambios ---
try:
    print("\n### ? Guardando el DataFrame modificado a Excel ###")
```

```
        # Usar .to_excel para sobrescribir el archivo con el DataFrame actualizado
        # index=False es crucial para evitar que pandas escriba el índice (0, 1, 2...) como una columna extra
        df.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
        print(f"Archivo '{nombre_archivo}' guardado exitosamente con las actualizaciones.")

        print("\nDataFrame final:")
        print(df)

    except Exception as e:
        print(f"Ocurrió un error al guardar el archivo: {e}")

# Ejecutar la función principal
automatizar_excel_pandas()
```

? Ventajas de Usar pandas

Característica	openpyxl	pandas
Enfoque	Interacción directa celda por celda.	Manipulación de datos en bloque (DataFrames).
Lectura/Escritura	Más lento para grandes archivos.	**Más rápido** para grandes volúmenes de datos.
Manipulación	Tareas como filtrar o modificar solo una celda son más directas.	Operaciones complejas (filtrar, agrupar, cálculos en columnas) son mucho **más fáciles** y eficientes.

Con pandas, el proceso de automatización se centra en **modificar los datos en la memoria** (el DataFrame) y luego **guardar el resultado final**, en lugar de ir y venir al archivo Excel para cada cambio.
¿Necesitas ayuda para implementar una función de **búsqueda** o **filtrado** específica en este DataFrame de Pandas?

¡Perfecto! Aquí tienes el código para implementar una función de **búsqueda y filtrado** que te permitirá encontrar productos específicos en tu DataFrame de Pandas.

? Búsqueda y Filtrado en Pandas

Añadiremos una función que toma el DataFrame, un nombre de columna y un valor de búsqueda para mostrar solo las filas que coincidan.

1. El Código Python Actualizado

Hemos integrado la nueva función **filtrar_datos** y la hemos llamado después de cargar el DataFrame.
Python

```
import pandas as pd
import os

def crear_archivo_ejemplo(nombre_archivo, nombre_hoja):
    """Crea el archivo Excel de ejemplo si no existe."""
    datos = {
```

```
            'ID Producto': [201, 202, 203, 204, 205],
            'Nombre': ['Teclado', 'Ratón', 'Webcam', 'Monitor', 'Cable HDMI'],
            'Stock': [35, 40, 12, 18, 55],
            'Precio': [25.50, 15.00, 45.99, 199.99, 8.50]
        }
        df_ejemplo = pd.DataFrame(datos)
        df_ejemplo.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
        print(f"Archivo de ejemplo '{nombre_archivo}' creado exitosamente.")

def filtrar_datos(df, columna, valor):
        """Filtra el DataFrame basado en una columna y un valor específico."""
        print(f"\n### ? Filtrando: Buscando '{valor}' en la columna '{columna}' ###")

        # 1. Crear una máscara booleana (True/False)
        # Ejemplo: df['Stock'] > 20
        mascara = df[columna] == valor

        # 2. Aplicar la máscara al DataFrame para obtener solo las filas que son True
        df_filtrado = df[mascara]

        if df_filtrado.empty:
            print("No se encontraron resultados para este filtro.")
        else:
            print(f"Resultados encontrados (DataFrame Filtrado):\n")
            print(df_filtrado)

        return df_filtrado

def automatizar_excel_pandas(nombre_archivo="datos_pandas.xlsx", nombre_hoja="Inventario"):
        """
        Lee, filtra, modifica y guarda datos en un archivo Excel usando pandas.
        """

        # --- 1. Crear archivo de prueba si no existe ---
        if not os.path.exists(nombre_archivo):
            crear_archivo_ejemplo(nombre_archivo, nombre_hoja)

        # --- 2. Lectura de Datos ---
        try:
            print("\n### ? Leyendo datos de Excel con Pandas ###")
            df = pd.read_excel(nombre_archivo, sheet_name=nombre_hoja)
            print("Contenido actual del DataFrame:")
            print(df)

        except Exception as e:
            print(f"Ocurrió un error al leer el archivo: {e}")
            return

        # --- 3. Uso de la función de Filtrado ---

        # Ejemplo 1: Buscar un producto por nombre exacto
        filtrar_datos(df, columna='Nombre', valor='Webcam')

        # Ejemplo 2: Filtrar usando condiciones (en este caso, productos con Stock bajo)
        print(f"\n### ? Filtrando por Condición: Stock < 20 ###")
```

```
    df_bajo_stock = df[df['Stock'] < 20]
    print(df_bajo_stock)

    # --- 4. Actualización de Datos ---

    print("\n###  Actualizando datos en el DataFrame ###")

    # Actualizar el Stock del 'Monitor'
    nuevo_stock = 50
    df.loc[df['Nombre'] == 'Monitor', 'Stock'] = nuevo_stock
    print(f"El Stock del producto 'Monitor' ha sido actualizado a: {nuevo_stock}")

    # --- 5. Guardar los Cambios ---

    print("\n### ? Guardando el DataFrame modificado a Excel ###")
    df.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
    print(f"Archivo '{nombre_archivo}' guardado exitosamente con las actualizaciones.")

    print("\nDataFrame final (incluyendo el cambio del Monitor):")
    print(df)

# Ejecutar la función principal
automatizar_excel_pandas()
```

? Cómo Funciona el Filtrado en Pandas

La clave del filtrado en pandas es el concepto de **Máscara Booleana**.

1. **Condición:** Creas una condición que se aplica a toda una columna.
   `df['Columna'] == valor`

   Esto devuelve una Serie de True/False. Por ejemplo:

Nombre	Condición ('Nombre'=='Webcam')
Teclado	False
Ratón	False
Webcam	True
Monitor	False

2. **Aplicación:** Pasas esa Serie de True/False de vuelta al DataFrame:
   `df[mascara]`

   El DataFrame (df) solo devuelve las filas donde la máscara es **True**, logrando así el filtrado.

Esta técnica te permite hacer filtros mucho más avanzados, como:

- Filtrar por **múltiples condiciones** usando operadores lógicos (& para AND, | para OR).
  `df[(df['Stock'] < 20) & (df['Precio'] > 100)]` (Stock bajo **Y** precio alto)
- Filtrar por **contenido de texto** usando .str.contains().
  `df[df['Nombre'].str.contains('a', case=False)]` (Productos que contienen la letra 'a')

¿Te gustaría que te muestre un ejemplo de **filtrado con múltiples condiciones**?

¡Absolutamente! El filtrado con múltiples condiciones es una de las funciones más poderosas de Pandas.

Aquí te muestro cómo aplicar condiciones combinadas usando los operadores lógicos de Python: **&** (AND) y | (OR).

? Filtrado con Múltiples Condiciones

Para combinar condiciones en Pandas, debes recordar dos reglas clave:

1. Cada condición debe estar **envuelta en paréntesis** ().
2. Debes usar los operadores lógicos de Pandas:

- **&** para **AND**: Ambas condiciones deben ser ciertas.
- | para **OR**: Al menos una de las condiciones debe ser cierta.

1. El Código Python con AND y OR

Hemos añadido una nueva sección para demostrar los filtros avanzados en el script de automatización:

Python

```
import pandas as pd
import os

# ... (Las funciones crear_archivo_ejemplo y filtrar_datos son las mismas que antes) ...

def crear_archivo_ejemplo(nombre_archivo, nombre_hoja):
    """Crea el archivo Excel de ejemplo si no existe."""
    datos = {
        'ID Producto': [201, 202, 203, 204, 205, 206],
        'Nombre': ['Teclado', 'Ratón', 'Webcam', 'Monitor', 'Cable HDMI', 'Altavoces'],
        'Stock': [35, 40, 12, 18, 55, 7],
        'Precio': [25.50, 15.00, 45.99, 199.99, 8.50, 65.00]
    }
    df_ejemplo = pd.DataFrame(datos)
    df_ejemplo.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
    print(f"Archivo de ejemplo '{nombre_archivo}' creado exitosamente.")

def automatizar_excel_pandas(nombre_archivo="datos_pandas.xlsx", nombre_hoja="Inventario"):
    """
    Lee, filtra, modifica y guarda datos en un archivo Excel usando pandas.
    """

    # --- 1. Crear archivo de prueba si no existe ---
    if not os.path.exists(nombre_archivo):
        crear_archivo_ejemplo(nombre_archivo, nombre_hoja)

    # --- 2. Lectura de Datos ---
    try:
        print("\n### ? Leyendo datos de Excel con Pandas ###")
        df = pd.read_excel(nombre_archivo, sheet_name=nombre_hoja)
        print("Contenido actual del DataFrame:")
        print(df)

    except Exception as e:
        print(f"Ocurrió un error al leer el archivo: {e}")
        return
```

```
    # ----------------------------------------------------
    # --- 3. Filtrado con Múltiples Condiciones (AND / OR) ---
    # ----------------------------------------------------

    print("\n### ?□ FILTRADO AVANZADO (Múltiples Condiciones) ###")

    # Ejemplo 1: Operador AND (&)
    # Queremos productos que tengan un Stock MENOR a 20 Y un Precio MAYOR a 50.

    print("\n Filtro AND (&): Stock < 20 Y Precio > 50")

    condicion_stock_bajo = df['Stock'] < 20
    condicion_precio_alto = df['Precio'] > 50.00

    # Combinamos ambas condiciones usando el operador &
    df_and = df[condicion_stock_bajo & condicion_precio_alto]
    print(df_and)
    # Resultado esperado: Monitor y Altavoces (18 y 7 en Stock, 199.99 y 65.00 en Precio)


    # Ejemplo 2: Operador OR (|)
    # Queremos productos que tengan un ID Producto IGUAL a 201 O que tengan un Stock MAYOR a 50.

    print("\n Filtro OR (|): ID Producto = 201 O Stock > 50")

    condicion_id_especifico = df['ID Producto'] == 201
    condicion_stock_muy_alto = df['Stock'] > 50

    # Combinamos ambas condiciones usando el operador |
    df_or = df[condicion_id_especifico | condicion_stock_muy_alto]
    print(df_or)
    # Resultado esperado: Teclado (ID 201) y Cable HDMI (Stock 55)

    # ----------------------------------------------------
    # --- 4. Actualización de Datos (Usando Filtro AND) ---
    # ----------------------------------------------------

    print("\n###  Actualizando datos del filtro AND ###")

    # Usamos el filtro AND del Ejemplo 1 para actualizar una columna
    # A todos los productos con Stock < 20 Y Precio > 50 (Monitor y Altavoces), les bajamos el precio un 10%.

    df.loc[condicion_stock_bajo & condicion_precio_alto, 'Precio'] *= 0.90
    print("El precio de los productos de alto valor con stock bajo se redujo en un 10%.")

    # --- 5. Guardar los Cambios ---

    print("\n### ? Guardando el DataFrame modificado a Excel ###")
    df.to_excel(nombre_archivo, sheet_name=nombre_hoja, index=False)
    print(f"Archivo '{nombre_archivo}' guardado exitosamente con las actualizaciones.")

    print("\nDataFrame final (con los precios actualizados):")
    print(df)

# Ejecutar la función principal
automatizar_excel_pandas()
```

? Puntos Clave en la Sintaxis

La línea esencial para el filtro **AND** es:
$$\text{df}[\underbrace{(\text{df['Stock']} < 20)}_{\text{Condición 1}} \quad\mathbf{\&}\quad \underbrace{(\text{df['Precio']} > 50.00)}_{\text{Condición 2}}] $$Y la línea esencial para actualizar el DataFrame usando un filtro **AND** es: $$\text{df}\textbf{.loc}[\underbrace{\text{Condición 1}}\mathbf{\&} \text{Condición 2}}*{\text{Máscara Booleana}}, \quad \underbrace{\text{'Precio'}}*{\text{Columna a cambiar}}] = \text{nuevo valor} $$El uso de **`.loc`** con la máscara booleana es la forma más segura y recomendada de actualizar solo las filas que cumplen la condición. ¿Hay algún otro tipo de operación en Excel, como **agrupación de datos** (`GROUP BY` en SQL), que te gustaría aprender a hacer con Pandas?$$