Manipulación de Archivos en Python

1. Introducción

En los programas que desarrollamos, los datos normalmente viven en la memoria RAM mientras el programa está en ejecución. Cuando el programa finaliza, toda la información se pierde. Para poder guardar datos de forma permanente y reutilizarlos en otras ejecuciones, utilizamos archivos.

Un archivo es un conjunto de datos almacenado en el disco. Estos datos pueden ser texto, números, imágenes, configuraciones, entre otros. Algunos ejemplos comunes de archivos que se utilizan en el desarrollo de software son:

- Archivos **.txt** para guardar texto plano.
- Archivos **.csv** para almacenar datos en forma tabular, separados por comas.
- Archivos **.json** para registrar eventos o errores.

Gracias al uso de archivos, nuestros programas pueden:

- **Guardar datos para usarlos luego** (persistencia).
- **Intercambiar información con otros sistemas** (interoperabilidad).
- **Leer configuraciones, exportar resultados o registrar acciones**.

2. Tipos de archivos que trabajaremos

Archivos de texto plano (.txt)

Contienen solo caracteres legibles por humanos. Se pueden abrir con cualquier editor de texto (como el Bloc de Notas). Son ideales para almacenar listas, mensajes, o estructuras simples como nombres, productos, etc. Ejemplo de archivo **nombres.txt**

Archivos CSV (.csv)

Los archivos CSV (Comma Separated Values) son similares a planillas: cada línea representa una fila, y los valores están separados por comas.

Ejemplo de archivo **productos.csv**

3. Operaciones básicas con archivos en Python

3.1 Abrir y cerrar archivos

Para trabajar con archivos, Python ofrece la función **open()** especificando nombre y/ruta del archivo entre comillas y modo de apertura también entre comillas, por último, para cerrarlos **close()**:

```
archivo = open("archivo.txt", "r")  # Abrir archivo en modo lectura
archivo.close()  # Cerrar el archivo
```

Modos de apertura:

- **'r'**: lectura (read). El archivo debe existir.
- **'w'**: escritura (write). Si el archivo existe, se sobreescribe. Si no, se crea.
- **'a'**: agregar (append). Se abre para escribir al final del archivo sin borrar lo anterior.

Cierre automático con with:

Es recomendable usar **with** para abrir archivos, ya que se cierran automáticamente al finalizar el bloque:

```
with open("datos.txt", "r") as archivo:
    # se puede trabajar con el archivo dentro del bloque
    contenido = archivo.read()
```

3.2 Leer archivos línea por línea

Para leer un archivo texto, una de las formas más comunes es:

```
with open("nombres.txt", "r") as archivo:
    for linea in archivo:
        print(linea.strip())
```

- **for linea in archivo**: recorre el archivo línea por línea.
- **.strip()**: elimina los saltos de línea u otros caracteres vacíos.

Separar datos con .split()

Si las líneas tienen varios valores separados por algún delimitador (coma, espacio, etc.), se puede usar **split()** para dividirlos:

```
linea = "Lapicera,120.5,30"
partes = linea.split(",")
print(partes)  # ['Lapicera', '120.5', '30']
```

Esto permite transformar cada línea en una **lista**, y luego convertirla en **tupla** o **diccionario** si se desea trabajar con estructuras más complejas.

3.3 Escribir en archivos

Para guardar información en un archivo:

```
with open("salida.txt", "w") as archivo:
    archivo.write("Primera línea\n")
    archivo.write("Segunda línea\n")
```

Si se desea **agregar** contenido sin borrar lo anterior:

```
with open("salida.txt", "a") as archivo:
    archivo.write("Nueva línea al final\n")
```

4. Buenas prácticas al trabajar con archivos

- Usar **with** para garantizar el cierre del archivo incluso si ocurre un error.
- Verificar si el archivo existe antes de leerlo usando **os.path.exists()** (si se ve necesario).
- Cuidar no sobrescribir archivos importantes. Usar **'a'** si solo se desea agregar contenido.
- Evitar abrir archivos con rutas incorrectas. Preferir rutas relativas al archivo del programa.
- Validar siempre los datos leídos del archivo, ya que pueden tener errores o estar incompletos.