ChatGPT

Manejo de archivos con Python

El **manejo de archivos** permite a los programas leer y escribir datos en archivos externos, lo cual es esencial para la persistencia de datos (por ejemplo, configuraciones, datos de usuarios, logs, etc.) o el intercambio de información entre sistemas. En Python, esto se logra usando la función `open()` para obtener un objeto archivo (file object) que abstrae un **file descriptor** del sistema operativo. El *descriptor de archivo* es un número entero que identifica internamente al archivo abierto [1]. Python lo oculta en objetos de alto nivel (con método `fileno()`), pero internamente el sistema operativo lo usa para realizar la entrada/salida en el archivo.

Apertura de archivos (`open`)

La función `open(nombre, modo, encoding)` abre un archivo y devuelve un objeto archivo [2]. Por ejemplo:

```
f = open("datos.txt", "r", encoding="utf-8")
```

Aquí `modo` es una cadena que indica cómo se usará el archivo (lectura, escritura, etc.). En modo texto (por defecto) se leen y escriben cadenas en la codificación dada; si se agrega `'b'` (por ejemplo `"rb"` o `"wb"`) se abre en modo binario (los datos se tratan como `bytes`) [3]. Es buena práctica usar el contexto `with` al abrir archivos, ya que automáticamente cierra el archivo al salir del bloque, liberando recursos [4]. Por ejemplo:

```
with open("datos.txt", "r", encoding="utf-8") as f:
    contenido = f.read()
# al salir del bloque, f.close() se invoca implícitamente
```

Modos de apertura

Python admite varios **modos de apertura**, cada uno controlando el comportamiento de lectura/escritura del archivo. Los modos más comunes son [5] [6]:

- `'r'`: *solo lectura*. El puntero se ubica al inicio. Falla si el archivo no existe [6]. (Por defecto, si no se indica modo, se asume `'r'` [5]).
- `'w'`: *solo escritura*. Crea un archivo nuevo o sobrescribe uno existente (trunca el contenido previo) [5].
- `'a'`: *solo añadir*. Crea el archivo si no existe y sitúa el puntero al final, de modo que cada `write()` anexa datos al final [5].
- `'r+'`: *lectura y escritura*. Similar a `'r'` pero permite también escribir; falla si el archivo no existe [6].
- `'w+'`: *lectura y escritura*. Similar a `'w'`, crea/abre un archivo vacío listo para leer y escribir (sobrescribe contenido existente) [7].

- `'a+'`: *lectura y añadir*. Crea el archivo si no existe; el puntero se coloca al final pero es posible leer todo el archivo [8].
- `'x'`: *crear exclusivamente*. Crea un archivo nuevo solo si no existe, y falla si ya existe [9].

Además, `'b'` (binario) y `'t'` (texto) pueden combinarse con estos (por ejemplo `"wb"`, `"r+t"`, etc.) [10].

Leer archivos en Python

Para leer datos de un archivo se suele abrir con modos de lectura (`'r'` o `'r+'`) y usar métodos del objeto archivo. Entre los principales están:

- `f.read([n])`: lee hasta `n` caracteres (o bytes) y devuelve una cadena. Si no se pasa `n`, lee todo el archivo [11].
- `f.readline()`: lee la siguiente línea (hasta el salto de línea `\n`) y la devuelve como cadena [12].
- `f.readlines()`: lee todas las líneas restantes y las devuelve como una lista de cadenas (una por línea) [13].

Por ejemplo:

```
with open("data.csv", "r", encoding="utf-8") as f:
    texto_completo = f.read()        # lee todo el contenido
    f.seek(0)                        # vuelve al inicio del archivo
    primera_linea = f.readline()     # lee la primera línea
    lineas = f.readlines()          # lee todas las líneas restantes
```

También es común iterar sobre el archivo línea por línea: `for linea in f: ...`. Esto lee una línea en cada iteración.

Escribir archivos en Python

Para escribir se abre el archivo en un modo de escritura (`'w'`, `'a'`, `'w+'`, etc.) y se usan métodos como:

- `f.write(cadena)`: escribe la cadena (o bytes en modo binario) en el archivo en la posición actual [14]. No añade salto de línea automáticamente.
- `f.writelines(lista_de_cadenas)`: escribe todas las cadenas en la lista, una tras otra [15]. No añade saltos de línea entre elementos, así que las cadenas deberían incluir `\n` si se desea separar líneas.

Por ejemplo:

```
with open("salida.txt", "w", encoding="utf-8") as f:
    f.write("Hola Mundo!\n")
    lineas = ["Línea 1\n", "Línea 2\n", "Línea 3\n"]
    f.writelines(lineas)
```

Si se abre en modo `'a'`, cada llamada a `write()` añadirá datos al final del archivo existente. Después de escribir, es importante cerrar el archivo (lo cual hace `with` automáticamente) para asegurar que todos los datos se graben correctamente en disco [4] [16].

Modificar nombre y borrar archivos

Para **renombrar** un archivo en Python se usa la función `os.rename(antiguo, nuevo)` [17]. Por ejemplo:

```
import os
os.rename("archivo_antiguo.txt", "archivo_nuevo.txt")
```

Esto cambia el nombre (o la ruta) del archivo. Si ya existe un archivo con el nuevo nombre, se generará un `FileExistsError` a menos que se maneje adecuadamente con excepciones. Para **eliminar** un archivo, se emplea `os.remove(ruta)`. Ambos requieren importar el módulo `os`. (Por ejemplo, `os.remove("archivo_nuevo.txt")` borra el archivo indicado.)

Cerrar archivos

Siempre hay que cerrar los archivos abiertos. Llamar a `f.close()` libera el descriptor y asegura que los datos pendientes se escriban [16]. Si se usa `with open(...) as f:`, Python invoca `f.close()` automáticamente al salir del bloque, incluso si ocurre un error en medio [4]. No cerrar un archivo puede causar fugas de recursos o archivos corruptos.

Ejemplos de programas en Python

1. **Lectura de datos desde un archivo externo:** supongamos que tenemos un archivo `valores.txt` con números, uno por línea. El programa lee todos los números y calcula su suma. Por ejemplo:

```
# Ejemplo: sumar números desde un archivo
total = 0
with open("valores.txt", "r", encoding="utf-8") as f:
    for linea in f:                # itera línea por línea
        if linea.strip():          # ignora líneas vacías
            num = int(linea)       # convierte a entero
            total += num
print("La suma de los valores es:", total)
```

1. **Escritura de datos a un archivo externo:** este ejemplo escribe las tablas de multiplicar del 1 al 3 en un archivo `salida.txt`.

```
# Ejemplo: escribir tablas de multiplicar en un archivo
with open("salida.txt", "w", encoding="utf-8") as f:
    for n in range(1, 4):
        linea = f"Tabla del {n}:\n"
        f.write(linea)
```

```
        for i in range(1, 11):
            f.write(f"{n} x {i} = {n*i}\n")
        f.write("\n")  # línea en blanco entre tablas
print("Tablas escritas en 'salida.txt'")
```

En estos ejemplos se utilizan los modos de apertura apropiados (`"r"` para leer, `"w"` para escribir) y los métodos `write` / `writelines` y lectura de líneas. Se maneja automáticamente el cierre de archivos con `with`, y para renombrar o eliminar archivos externos se usarían las funciones mencionadas del módulo `os`.

Referencias: La documentación oficial de Python y tutoriales reconocidos explican detalladamente estas operaciones de archivos [5] [6] [18], incluyendo los modos de apertura, métodos de lectura y escritura, y la recomendación de usar `with open(...)` para manejar el cierre automático [4] [16]. Las citas anteriores apuntan a estas fuentes para mayor profundización.

---

[1] Descriptor de archivo - Wikipedia, la enciclopedia libre
https://es.wikipedia.org/wiki/Descriptor_de_archivo

[2] [3] [4] [5] [10] 7. Entrada y salida — documentación de Python - 3.13.9
https://docs.python.org/es/3.13/tutorial/inputoutput.html

[6] [7] [8] [9] [11] [12] [13] [14] [15] [16] [18] Manejo de archivos en Python: Cómo crear, leer y escribir en un archivo
https://www.freecodecamp.org/espanol/news/manejo-de-archivos-en-python-como-crear-leer-y-escribir-en-un-archivo/

[17] ▷ Renombrar y Eliminar Archivos en Python - Oregoom.com
https://oregoom.com/python/renombrar-eliminar-archivos/