Escribiendo en un archivo con Open()

Tiempo estimado necesario: 10 minutos

Objetivo

1. Crear y escribir datos en un archivo en Python
2. Escribir múltiples líneas de texto en un archivo utilizando listas y bucles
3. Agregar nueva información a un archivo ya existente sin borrar su contenido
4. Comparar y contrastar los diferentes modos de archivo en Python, qué significan y cuándo usarlos

Escribir en un archivo

Puedes crear un nuevo archivo de texto y escribir datos en él utilizando la función `open()` de Python. La función `open()` toma dos argumentos principales: la ruta del archivo (incluido el nombre del archivo) y el parámetro de modo, que especifica la operación que deseas realizar en el archivo. Para escribir, debes usar el modo 'w'. Aquí tienes un ejemplo:

```
# Create a new file Example2.txt for writing
with open('Example2.txt', 'w') as file1:
    file1.write("This is line A\n")
    file1.write("This is line B\n")
    # file1 is automatically closed when the 'with' block exits
```

Explicación de la línea 2: `with open('Example2.txt', 'w') as file1:`

- Comenzamos utilizando la función open para abrir o crear un archivo llamado `Example2.txt` para escritura (modo 'w').
- El modo `'w'` especifica que tenemos la intención de escribir datos en el archivo.
- Usamos la declaración `with` para asegurarnos de que el archivo se cierre automáticamente cuando el bloque de código finalice. Esto ayuda a gestionar los recursos de manera eficiente.

Explicación de la línea 3: `file1.write("This is line A\n")`

- Aquí, usamos el método `write()` en el objeto de archivo, `file1`, para agregar el texto `This is line A` al archivo.
- El `\n` al final representa un carácter de nueva línea, que inicia una nueva línea en el archivo.

Explicación de la línea 4 `file1.write("This is line" B\n")`

- De manera similar, utilizamos el método `write()` nuevamente para agregar el texto `This is line B` al archivo en una nueva línea.

Escribiendo múltiples líneas en un archivo usando una lista y un bucle

En Python, puedes usar una lista para almacenar múltiples líneas de texto y luego escribir estas líneas en un archivo usando un bucle. Aquí hay un ejemplo de código que demuestra esto:

```
# List of lines to write to the file
Lines = ["This is line 1", "This is line 2", "This is line 3"]
# Create a new file Example3.txt for writing
with open('Example3.txt', 'w') as file2:
    for line in Lines:
        file2.write(line + "\n")
    # file2 is automatically closed when the 'with' block exits
```

Aquí tienes una explicación del código:

- Línea 2: Comenzamos definiendo una lista llamada `Lines`, que contiene múltiples líneas de texto que queremos escribir en el archivo. Cada línea es una cadena.
- Línea 5: A continuación, utilizamos la función `open()` para crear un nuevo archivo de texto llamado `Example3.txt` para escritura, en modo `'w'`. El modo `'w'` indica que tenemos la intención de escribir datos en el archivo.
- Línea 6: Luego entramos en un bucle for para iterar a través de cada elemento (línea) en la lista `Lines`.
- Línea 7: Dentro del bucle, utilizamos el método `write()` en el objeto de archivo `file2` para escribir la línea de texto actual (línea) en el archivo. Agregamos `\n` al final de cada línea para asegurarnos de que cada línea esté seguida por un carácter de nueva línea, que las separa en el archivo.
- Línea 8: Finalmente, añadimos un comentario indicando que el archivo `file2` se cerrará automáticamente cuando el bloque de código dentro de la declaración with salga. Cerrar adecuadamente el archivo es esencial para una buena gestión de recursos.

Agregar datos a un archivo existente

En Python, puedes usar el modo `'a'` al abrir un archivo para agregar nuevos datos a un archivo existente sin sobrescribir su contenido. Aquí hay un fragmento de código de ejemplo que demuestra esto:

```
# Data to append to the existing file
new_data = "This is line C"
# Open an existing file Example2.txt for appending
with open('Example2.txt', 'a') as file1:
    file1.write(new_data + "\n")
    # file1 is automatically closed when the 'with' block exits
```

Aquí hay una explicación del código:

- Línea 2: Comenzamos definiendo una variable `new_data` que contiene el texto que queremos agregar al archivo existente. En este caso, es la cadena `This is line C.`
- Línea 5: A continuación, usamos la función `open()` para abrir un archivo existente llamado `Example2.txt` en modo de adición, `'a'`. El modo `'a'` indica que tenemos la intención de agregar datos al archivo, y si el archivo no existe, se creará.
- Línea 6: Dentro del bloque with, usamos el método `write()` en el objeto de archivo `file1` para agregar el `new_data` al archivo. Agregamos `"\n"` al final para asegurarnos de que los datos añadidos comiencen en una nueva línea, manteniendo la legibilidad del archivo.
- Finalmente, añadimos un comentario indicando que el archivo `file1` se cerrará automáticamente cuando el bloque de código dentro de la declaración `with` termine. Cerrar adecuadamente el archivo es esencial para

una buena gestión de recursos.

Copiando contenidos de un archivo a otro

En Python, puedes copiar el contenido de un archivo a otro leyendo del archivo fuente y escribiendo en el archivo de destino. Aquí hay un ejemplo de código que demuestra esto:

```
# Open the source file for reading
with open('source.txt', 'r') as source_file:
    # Open the destination file for writing
    with open('destination.txt', 'w') as destination_file:
        # Read lines from the source file and copy them to the destination file
        for line in source_file:
            destination_file.write(line)
    # Destination file is automatically closed when the 'with' block exits
# Source file is automatically closed when the 'with' block exits
```

Aquí tienes una explicación del código:

- Línea 2: Comenzamos abriendo el archivo fuente, `source.txt` para lectura, en modo `r`, utilizando la declaración `with` y la función `open()`. Esto nos permite leer datos del archivo fuente.
- Línea 4: Dentro del primer bloque `with`, abrimos el archivo de destino, `destination.txt` para escritura, en modo `w`, utilizando otra declaración `with` y la función `open()`. Esto prepara el archivo de destino para la escritura.
- Línea 6: Usamos un bucle `for` para iterar a través de cada línea en el archivo fuente `source_file`. Este bucle lee cada línea del archivo fuente una por una.
- Línea 7: Dentro del bucle, utilizamos el método `write()` para escribir cada línea del archivo fuente en el archivo de destino `destination_file`. Esto copia efectivamente el contenido del archivo fuente al archivo de destino.
- Líneas 8 y 9: Después de copiar todas las líneas, ambos archivos, fuente y destino, se cierran automáticamente cuando sus respectivos bloques `with` finalizan. El cierre adecuado de archivos es crucial para gestionar los recursos de manera eficiente.

Modos de archivo en Python (sintaxis y casos de uso)

La siguiente tabla proporciona una visión general de los diferentes modos de archivo, su sintaxis y casos de uso comunes. Entender estos modos es esencial al trabajar con archivos en Python para diversas tareas de manipulación de datos.

Modo	Sintaxis	Descripción
‘r’	`'r'`	Modo de lectura. Abre un archivo existente para lectura. Genera un error si el archivo no existe.
‘w’	`'w'`	Modo de escritura. Crea un nuevo archivo para escritura. Sobrescribe el archivo si ya existe.
‘a’	`'a'`	Modo de anexado. Abre un archivo para anexar datos. Crea el archivo si no existe.

Modo	Sintaxis	Descripción
‘x’	`'x'`	Modo de creación exclusiva. Crea un nuevo archivo para escritura pero genera un error si el archivo ya existe.
‘rb’	`'rb'`	Modo de lectura binaria. Abre un archivo binario existente para lectura.
‘wb’	`'wb'`	Modo de escritura binaria. Crea un nuevo archivo binario para escritura.
‘ab’	`'ab'`	Modo de anexado binario. Abre un archivo binario para anexar datos.
‘xb’	`'xb'`	Modo de creación binaria exclusiva. Crea un nuevo archivo binario para escritura pero genera un error si ya existe.
‘rt’	`'rt'`	Modo de lectura de texto. Abre un archivo de texto existente para lectura. (Predeterminado para archivos de texto)
‘wt’	`'wt'`	Modo de escritura de texto. Crea un nuevo archivo de texto para escritura. (Predeterminado para archivos de texto)
‘at’	`'at'`	Modo de anexado de texto. Abre un archivo de texto para anexar datos. (Predeterminado para archivos de texto)
‘xt’	`'xt'`	Modo de creación de texto exclusiva. Crea un nuevo archivo de texto para escritura pero genera un error si ya existe.
‘r+’	`'r+'`	Modo de lectura y escritura. Abre un archivo existente para lectura y escritura.
‘w+’	`'w+'`	Modo de escritura y lectura. Crea un nuevo archivo para lectura y escritura. Sobrescribe el archivo si ya existe.
‘a+’	`'a+'`	Modo de anexado y lectura. Abre un archivo para anexar y leer. Crea el archivo si no existe.
‘x+’	`'x+'`	Modo de creación exclusiva y lectura/escritura. Crea un nuevo archivo para lectura y escritura pero genera un error si ya existe.

Conclusión

Trabajar con archivos es un aspecto fundamental de la programación, y Python proporciona herramientas poderosas para realizar diversas operaciones con archivos. En este resumen, cubrimos conceptos clave y ejemplos de código relacionados con la manipulación de archivos en Python, incluyendo la escritura, la adición y la copia de archivos.

Autor(es)

Akansha Yadav