7. Manejo de Archivos

El manejo de archivos es una habilidad esencial en programación. Los archivos son una forma de almacenar y compartir información. Python ofrece varias formas de trabajar con ellos, en esta sección, nos enfocaremos en su manejo, los diferentes formatos de archivo, y la lectura y escritura de diferentes tipos de archivos.

a) Introducción

Los archivos son una forma de almacenar información en una computadora, pueden ser de diferentes formatos, como texto, CSV, Excel, JSON, entre otros. Python ofrece varias formas de manejar archivos, incluyendo la apertura, lectura, escritura y cierre de los mismos.

b) Diferentes formatos de archivo.

Los diferentes formatos de archivo tienen diferentes formas de almacenar información
Los formatos de archivo más comunes son de texto, CSV, Excel y JSON.

c) Archivos de texto

Los archivos de texto son los más básicos y sencillos de manejar.

Para leer un archivo de texto en Python, utilizamos la función **open()** que recibe como argumento el nombre del archivo y el modo de apertura-lectura **“r”**. Luego, podemos leer el contenido del archivo utilizando el método **read()** o **readlines()**.

Supongamos que tenemos un archivo llamado "datos.txt". Podemos leer el archivo y visualizarlo línea a línea:

- Abrir el archivo en modo lectura

```
archivo = open("datos.txt", "r")
```

- Leer el contenido del archivo y guardarlo en una lista

```
lineas = archivo.readlines()
```

- Cerrar el archivo

```
archivo.close()
```

- Mostrar el contenido del archivo

```
for linea in lineas:
    print(linea)
```

Para escribir en un archivo de texto en Python, utilizamos la función **open()** con el modo de apertura-escritura "**w**". Luego, podemos escribir el contenido en el archivo utilizando el método **write()**.

- Abrir el archivo en modo escritura

```
archivo = open("salida.txt", "w")
```

- Escribir en el archivo

```
archivo.write("Esto es una prueba\n")
```

- Cerrar el archivo

```
archivo.close()
```

d) Archivos CSV

CSV significa Comma Separated Values (valores separados por comas) y es un formato de archivo comúnmente utilizado para almacenar información tabular.

Para leer un archivo CSV en Python, podemos utilizar la biblioteca estándar **csv**. Esta biblioteca proporciona una serie de funciones que nos permiten leer y escribir archivos CSV de manera sencilla.

Supongamos que tenemos un archivo llamado "datos.csv" ,podemos leerlo y mostrar su contenido de la siguiente manera:

- Importamos la librería csv

```
import csv
```

- Abrir el archivo en modo lectura

```
with open("datos.csv", newline='') as archivo:
    lector = csv.reader(archivo)
)
```

- Leer el contenido del archivo y guardarlo en una lista

```
lineas = list(lector)
```

- Mostrar el contenido del archivo

```
for linea in lineas:
    print(linea)
```

En las líneas anteriores utilizamos el **with statement** para abrir el archivo, lo que nos permite no tener que preocuparnos por cerrarlo manualmente. Además, utilizamos el argumento **newline=''** para evitar problemas con las líneas en blanco en el archivo.

Para escribir información que se encuentra en una lista llamada *datos* en un archivo CSV llamado *"nuevos_datos.csv"*, podemos usar el siguiente código:

```
import csv
datos = [['Juan', 'Pérez', '25'],
        ['María', 'González', '30']]

with open('nuevos_datos.csv','w',newline='') as archivo_csv:
    escritor_csv = csv.writer(archivo_csv)
    for fila in datos:
```

```
    escritor_csv.writerow(fila)
```

e) Archivos de Excel

Los archivos de Excel son una forma común de almacenar información tabular, en Python se recomienda utilizar la biblioteca pandas. A continuación, se presentan algunos ejemplos de cómo leer y escribir archivos de Excel usando esta biblioteca:
Para trabajar con archivos Excel en Python, se recomienda utilizar la biblioteca pandas. A continuación, se presentan algunos ejemplos de cómo leer y escribir archivos de Excel usando esta biblioteca:

Supongamos que tenemos un archivo en Excel llamado "archivo.xlsx" y los datos seran cargados en un DataFrame de pandas llamado df. El método head() se utiliza para mostrar las primeras filas del DataFrame.

- Importamos la librería pandas

```
import pandas as pd
```

- Leer un archivo Excel

```
df = pd.read_excel('archivo.xlsx', sheet_name='Hoja1')
```

- Mostrar el contenido del DataFrame

```
print(df.head())
```

Ahora crearemos un DataFrame de pandas con dos columnas y lo guardaremos en un nuevo archivo Excel llamado "nuevo_archivo.xlsx" sin incluir el índice del DataFrame. Luego, leeremos el archivo de Excel recién creado y mostraremos su contenido mediante el método head().

- Crear un DataFrame

```
df= pd.DataFrame({'col1':[1,2,3],'col2':['a','b','c']})
```

- Escribir los datos del DataFrame df en un archivo Excel

```
df.to_excel('nuevo_archivo.xlsx', index=False)
```

- Leer el archivo Excel recién creado

```
df_leido = pd.read_excel('nuevo_archivo.xlsx')
```

- Mostrar el contenido del DataFrame leído

```
df_leido.head()
```

f) Archivos JSON

JSON significa JavaScript Object Notation (Notación de Objetos de JavaScript) y es un formato de archivo comúnmente utilizado para intercambiar información entre aplicaciones web. Python ofrece la biblioteca json para trabajar con archivos JSON.

Supongamos que tenemos un archivo data.json, podemos leer y cargar estos datos en una variable de Python utilizando la función json.load() de la biblioteca json:

```
import json

with open('data.json', 'r') as f:
    data = json.load(f)
```

Supongamos que tenemos un diccionario de Python con los siguientes datos:

```
data = { "nombre": "Juan",
         "apellido": "Pérez",
         "edad": 30 }
```

Podemos escribir estos datos en un archivo data.json utilizando la función json.dump() de la biblioteca json. Este siguiente código guardará el diccionario en un archivo data.json. Si el archivo ya existe, se sobrescribirá con los nuevos datos. Si no existe, se creará un nuevo archivo con los datos proporcionados.

```
with open('data.json', 'w') as f:
    json.dump(data, f)
```

g) Otros tipos de archivos

Python es capaz de manejar una gran variedad de tipos de archivos, algunos ejemplos son:

Imágenes

Archivos con extensiones JPG, PNG, BMP, entre otros, se pueden leer y escribir utilizando las bibliotecas Pillow o OpenCV.

Archivos de audio

Archivos con extensiones MP3, WAV entre otros, se pueden leer y escribir utilizando las bibliotecas pydub o wave.

Archivos de video

Archivos con extensiones MP4, AVI entre otros se pueden leer y escribir utilizando las bibliotecas OpenCV o moviepy.

Archivos de base de datos

Archivos con extensiones SQL, NoSQL entre otros se pueden leer y escribir utilizando bibliotecas específicas para cada tipo de base de datos, como psycopg2 para PostgreSQL o pymongo para MongoDB.

Archivos de texto estructurados

Archivos con extensiones XML, HTML entre otros se pueden leer y escribir utilizando la biblioteca BeautifulSoup o la biblioteca xml.etree.ElementTree.

Archivos de texto plano

Archivos con extensiones de formato específico como INI, YAML, entre otros se pueden leer y escribir utilizando bibliotecas específicas para cada tipo de formato, como configparser para INI o PyYAML para YAML.

En resumen, el manejo de archivos es una habilidad esencial en programación y Python ofrece varias formas de trabajar con diferentes formatos de archivos. Desde archivos de texto hasta archivos de Excel y JSON, Python ofrece bibliotecas y funciones para manejarlos todos. Aprender a manejar archivos en Python puede ayudar a hacer el trabajo de manipulación de datos más eficiente y efectivo.