TIC

TALENTO TECH

Sesión 21

Programación Nivel Básico

UNIVERSIDAD LIBRE

etraining

Sesión 21:

Fundamentos del lenguaje Python

Lectura y escritura de archivos

Objetivos de la sesión

Al finalizar esta sesión estarás en capacidad de:

Aprender a manejar archivos

El objetivo principal es aprender a manejar archivos en Python de manera efectiva. Los participantes adquirirán habilidades para leer y escribir archivos diversos. Esto les permitirá gestionar datos y mejorar su productividad.

Introducción a conceptos clave

Exploraremos conceptos clave sobre el manejo de archivos en Python. Esto incluye definiciones, tipos de archivos y cómo interactuar con ellos. Al finalizar, los alumnos tendrán una base sólida de conocimientos.

Ejercicios prácticos

Realizaremos ejercicios prácticos para afianzar el aprendizaje. Los participantes aplicarán lo aprendido en ejercicios de lectura y escritura. Esto garantizará una comprensión profunda y habilidades aplicables.

Introducción a la Lectura y Escritura de Archivos en Python

Python es un lenguaje versátil que permite manejar archivos de manera sencilla y efectiva. La lectura y escritura de archivos son habilidades fundamentales para cualquier programador que desee trabajar con datos persistentes.

En esta sección, exploraremos cómo abrir, leer y escribir archivos en Python. También abordaremos los diferentes modos de apertura y algunos aspectos importantes a considerar durante el proceso.

Conceptos básicos de archivos

Qué es un archivo

Un archivo es un conjunto de datos almacenados en un dispositivo. Los archivos pueden contener texto, imágenes, audio o video. Cada tipo de archivo cumple una función específica.

Estructura de los archivos

Los archivos tienen una estructura que define cómo se almacenan los datos. Esta estructura incluye un nombre, una extensión y el contenido del archivo. La organización adecuada de un archivo es crucial para su accesibilidad.

Importancia de los archivos

Los archivos son esenciales para la gestión de datos y la comunicación. Permiten el almacenamiento y la recuperación de información. Además, facilitan la colaboración y el intercambio de conocimientos entre usuarios.

Definición de archivos

Concepto General

Un archivo es una colección de datos almacenados en un dispositivo de forma estructurada. Estos datos pueden ser texto, imágenes o cualquier otro tipo de información. Los archivos facilitan el almacenamiento y la recuperación de información en sistemas informáticos.

Importancia de los Archivos

Los archivos son esenciales para el funcionamiento de programas y aplicaciones. Almacenan configuraciones, datos de usuario y contenido, entre otros. Sin archivos, la gestión de información sería caótica e ineficiente.

Tipos de Archivos

Existen diversos tipos de archivos, como archivos de texto, binarios, de imagen, y de audio. Cada tipo tiene una estructura específica y un formato que permite su correcta interpretación. Los tipos de archivos determinan cómo son abiertos y leídos por los programas.

Tipos de Archivos

Archivos de texto

Los archivos de texto son simples y legibles. Almacenan datos en un formato que es fácilmente accesible. Se utilizan comúnmente para documentos y scripts de código.

Archivos binarios

Los archivos binarios almacenan datos en un formato no legible. Son más eficientes para almacenar imágenes y audio. Este tipo permite una mayor compresión y un acceso más rápido.

Archivos CSV

Los archivos CSV utilizan comas para separar valores. Son ideales para el intercambio de datos entre aplicaciones. Su estructura simple facilita la manipulación y el análisis de datos.

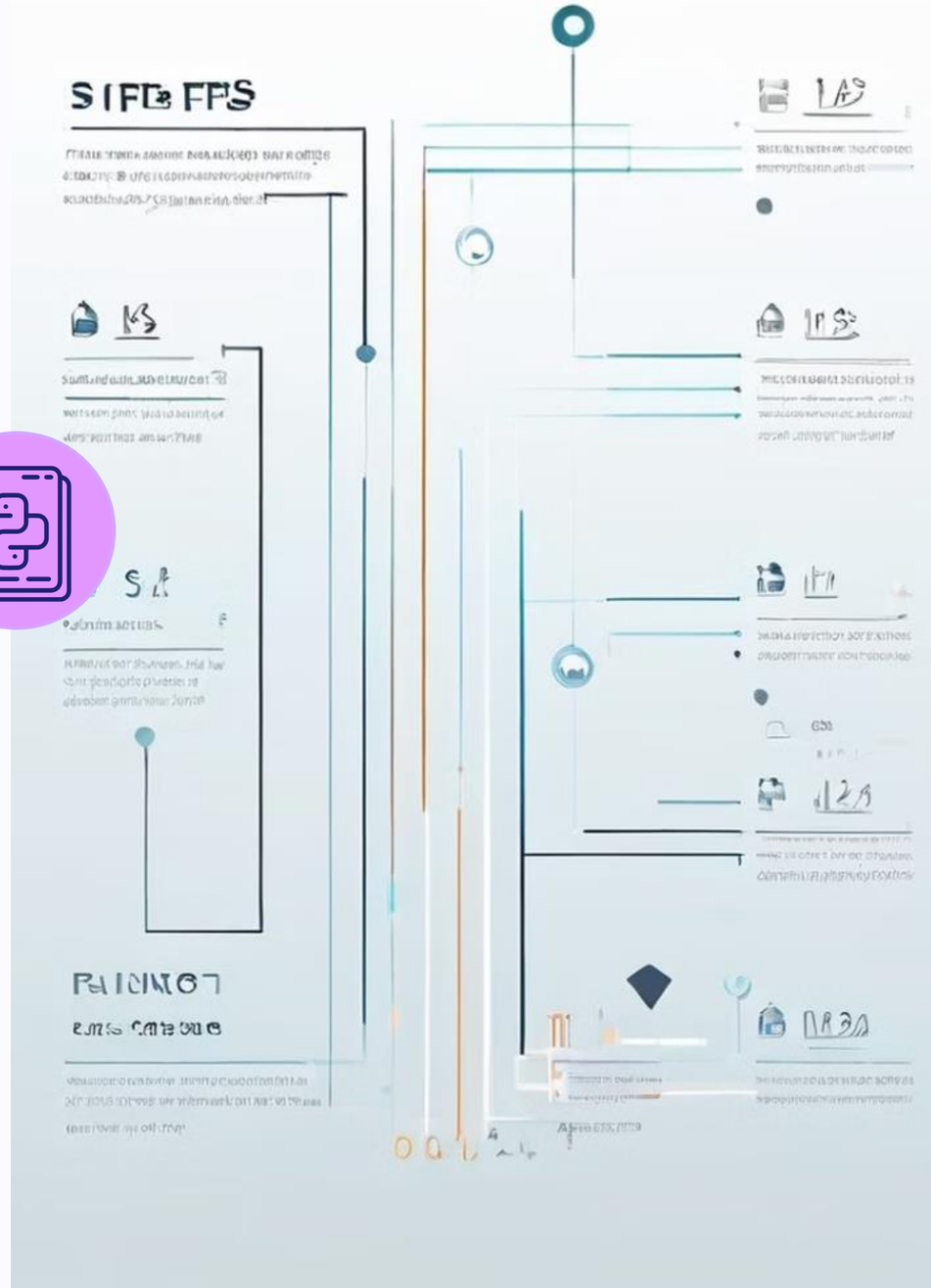
Archivos JSON

JSON es un formato de texto ligero para el intercambio de datos. Es fácil de leer y escribir para humanos y máquinas. Se utiliza frecuentemente en aplicaciones web y APIs.

Estructura de un archivo

La estructura de un archivo es fundamental para su correcto funcionamiento. Un archivo bien estructurado permite un acceso eficiente a la información. Comprender su diseño ayuda a optimizar la lectura y escritura de datos.

Generalmente, un archivo está compuesto por encabezados, datos y metadatos. Los encabezados contienen información esencial sobre el archivo. Los datos son el contenido principal, mientras que los metadatos proporcionan detalles sobre el formato y la creación.



Modos de apertura de archivos

Modo de Lectura (r)

Este modo se utiliza para leer el contenido de un archivo. No se permiten modificaciones en el archivo. Si el archivo no existe, se generará un error.

Modo de Escritura (w)

Este modo se utiliza para escribir en un archivo. Si el archivo ya existe, se sobrescribirá su contenido. Si no existe, se creará un nuevo archivo.

Modo de Adición (a)

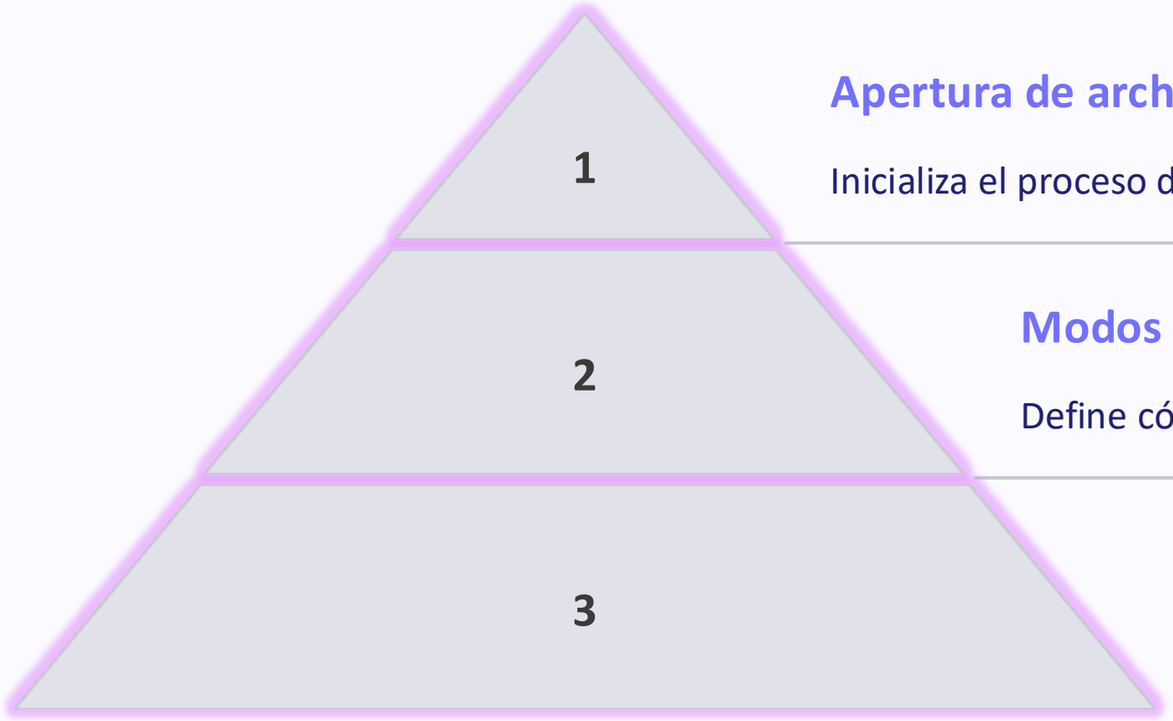
Permite añadir contenido al final de un archivo existente. No se modifica el contenido previo del archivo. Ideal para registro de datos.

Modo de Lectura y Escritura (r+)

Este modo permite tanto leer como escribir en el archivo. El contenido existente no se sobrescribirá a menos que se realice una operación de escritura. Ofrece flexibilidad en la manipulación de datos.

Función open()

La función **open()** es fundamental en Python para trabajar con archivos. Permite abrir archivos en diferentes modos, cada uno diseñado para una tarea específica. A continuación, se presenta un esquema sobre sus características y usos.



Apertura de archivos

Inicializa el proceso de lectura o escritura.

Modos de apertura

Define cómo interactuar con el archivo.

Gestión de errores

Previene problemas al abrir archivos inexistentes.

El uso correcto de la función **open()** garantiza la manipulación efectiva de información. Familiarizarse con sus parámetros es esencial para programadores de todos los niveles.

Lectura de archivos

1

1. Abrir archivo

El primer paso es abrir el archivo para su lectura.

2

2. Leer contenido

Utiliza métodos específicos para acceder al contenido del archivo.

3

3. Cerrar archivo

Finalmente, cierra el archivo para liberar recursos.

La lectura de archivos en Python permite obtener datos almacenados. Es crucial abrir el archivo en un modo adecuado para acceder a su contenido. Después de leer, siempre recuerda cerrar el archivo adecuadamente para evitar fugas de memoria.

Función read()

1

Descripción

La función **read()** se utiliza para leer el contenido de un archivo completo de una sola vez. Es ideal para archivos pequeños donde se desea obtener todo el texto en una variable. El método devuelve el contenido del archivo como una cadena.

2

Uso Básico

Para utilizar **read()**, primero se debe abrir el archivo en modo lectura. Luego, al llamar a la función, se obtiene el texto completo. Es importante cerrar el archivo después de su uso para liberar recursos.

3

Ejemplo Práctico

Considera un archivo de texto llamado *ejemplo.txt*. Al usar **read()**, puedes imprimir su contenido fácilmente. Esto facilita manipular el texto para análisis o procesamiento posterior.

Función readline()

1

Lectura de una Línea

La función readline() se utiliza para leer una línea completa de un archivo. Al usar esta función, el puntero se mueve a la siguiente línea del archivo después de la lectura. Esto es útil cuando se desea procesar archivos línea por línea.

2

Uso de la Función

Esta función puede ser utilizada en bucles para leer todas las líneas de un archivo secuencialmente. Además, puede manejar archivos de texto de gran tamaño sin cargar todo en memoria. Al usar readline(), se puede realizar un análisis específico de cada línea.

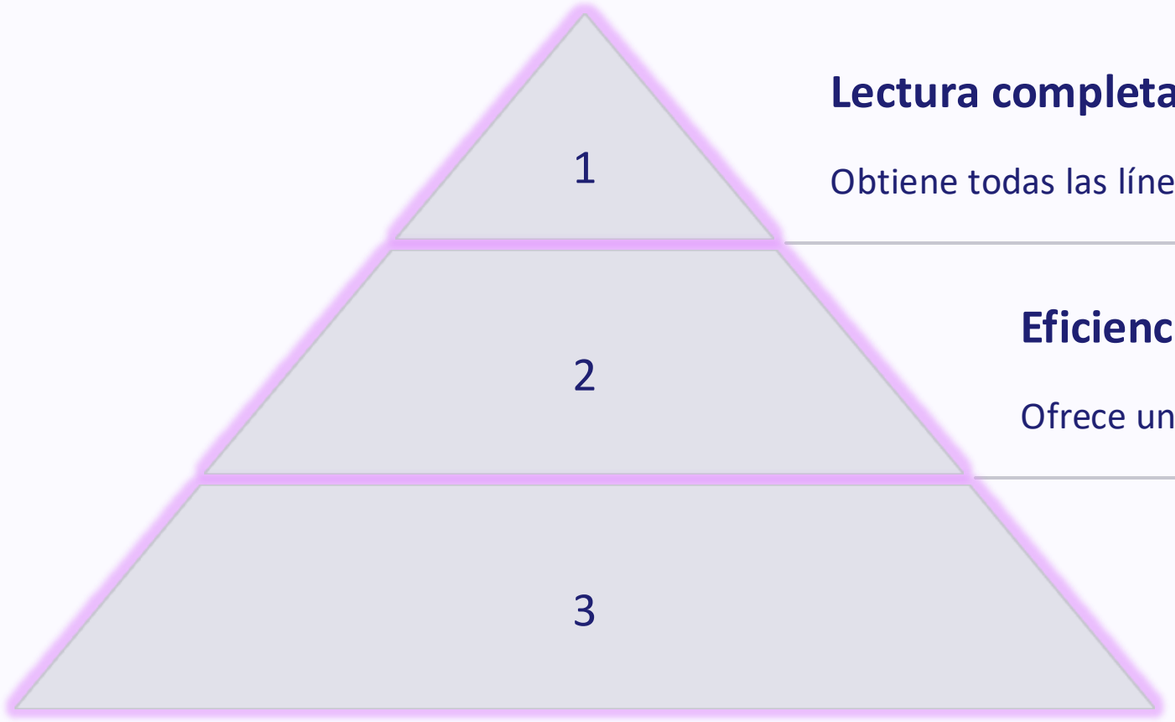
3

Ejemplo Práctico

Al abrir un archivo y utilizar readline(), se puede extraer información línea por línea. Por ejemplo, leer un archivo de registro para analizar eventos de sistema. Esto permite un tratamiento eficaz de datos y hace el código más limpio.

Función readlines()

La función **readlines()** se utiliza para leer todas las líneas de un archivo de texto a la vez. Esto es útil cuando se necesita procesar cada línea de forma individual en un bucle. En este caso, el archivo se abre en modo lectura.



Lectura completa

Obtiene todas las líneas de un archivo.

Eficiencia

Ofrece un método rápido para acceder a los datos.

Uso común

Ideal para archivos de texto pequeños o medianos.

Recuerda cerrar siempre el archivo después de leerlo. Esto asegura que no se consumen recursos innecesarios. Implementar un manejo adecuado de archivos es crucial en cualquier programa de Python.

Escritura de archivos

1

Paso 1: Abrir el archivo

Para escribir en un archivo, primero debes abrirlo. Utiliza la función *open()* y especifica el modo de escritura. El modo *'w'* crea un nuevo archivo o sobrescribe uno existente.

2

Paso 2: Escribir en el archivo

Después de abrir el archivo, puedes usar la función *write()* para agregar contenido. Es importante que el archivo esté abierto en modo escritura para evitar errores.

3

Paso 3: Cerrar el archivo

Una vez que termines de escribir, cierra el archivo con *close()*. Esto asegura que se guarden todos los cambios realizados en el archivo.

Función write()

1

Uso Básico

La función **write()** se utiliza para escribir texto en un archivo. Este método necesita un argumento, que es la cadena de texto a escribir. Después de ejecutar esta función, el texto se añade al archivo abierto.

2

Importante Consideración

Es esencial que el archivo esté en modo de escritura. De lo contrario, no podrás escribir en él. Si el archivo ya existe, al usar **write()**, su contenido se sobrescribirá.

3

Ejemplo Práctico

Por ejemplo, puedes usar **write()** para crear informes. En este caso, se puede generar un archivo con los resultados de una operación. Así, facilitas la lectura y保存 de datos para uso posterior.

Función writelines()

1

1. Introducción

La función writelines() permite escribir una lista de cadenas en un archivo.

2

2. Uso

Se utiliza principalmente para escribir múltiples líneas a la vez.

3

3. Ejemplo

Se puede usar para generar reportes o registros fácilmente.

La función writelines() resulta muy útil cuando se necesita escribir varias líneas de texto en un archivo. A diferencia de write(), que escribe una sola cadena, writelines() permite pasar toda una lista de líneas como argumento. Este enfoque simplifica mucho el proceso de escritura y es especialmente ventajoso en casos de procesamiento de datos en lote.

Manejo de errores

Identificación de Errores

El primer paso en el manejo de errores es identificarlos. Es importante saber cuándo y dónde ocurren. Esto ayuda a determinar la mejor manera de abordarlos.

Técnicas de Manejo

Existen varias técnicas para manejar errores en Python. Las más comunes incluyen *try*, *except* y *finally*. Estas técnicas permiten prevenir que el programa se detenga inesperadamente.

Recuperación de Errores

La recuperación de errores es crucial para el desarrollo eficiente de programas. Permite que los programas sigan funcionando incluso si ocurren problemas. Esto mejora la experiencia del usuario y la estabilidad del sistema.

Excepciones en archivos

Identificación de errores

Las excepciones son errores que ocurren durante la ejecución del programa. Es fundamental identificarlas para evitar que el programa se detenga abruptamente.

Manejo de excepciones

Utilizar bloques try-except permite manejar excepciones de manera controlada. Esto ayuda a continuar la ejecución del código, incluso si ocurre un error.

Tipos comunes de excepciones

Algunas excepciones comunes son FileNotFoundError y IOError. Conocerlas es esencial para el manejo adecuado de archivos.

Ejercicio 1: Leer un archivo de texto

1

Preparar el archivo

Antes de leer un archivo de texto, asegúrate de que esté disponible en la ubicación correcta. Verifica su nombre y extensión, como .txt. Además, puedes crear un archivo de texto simple para comenzar.

2

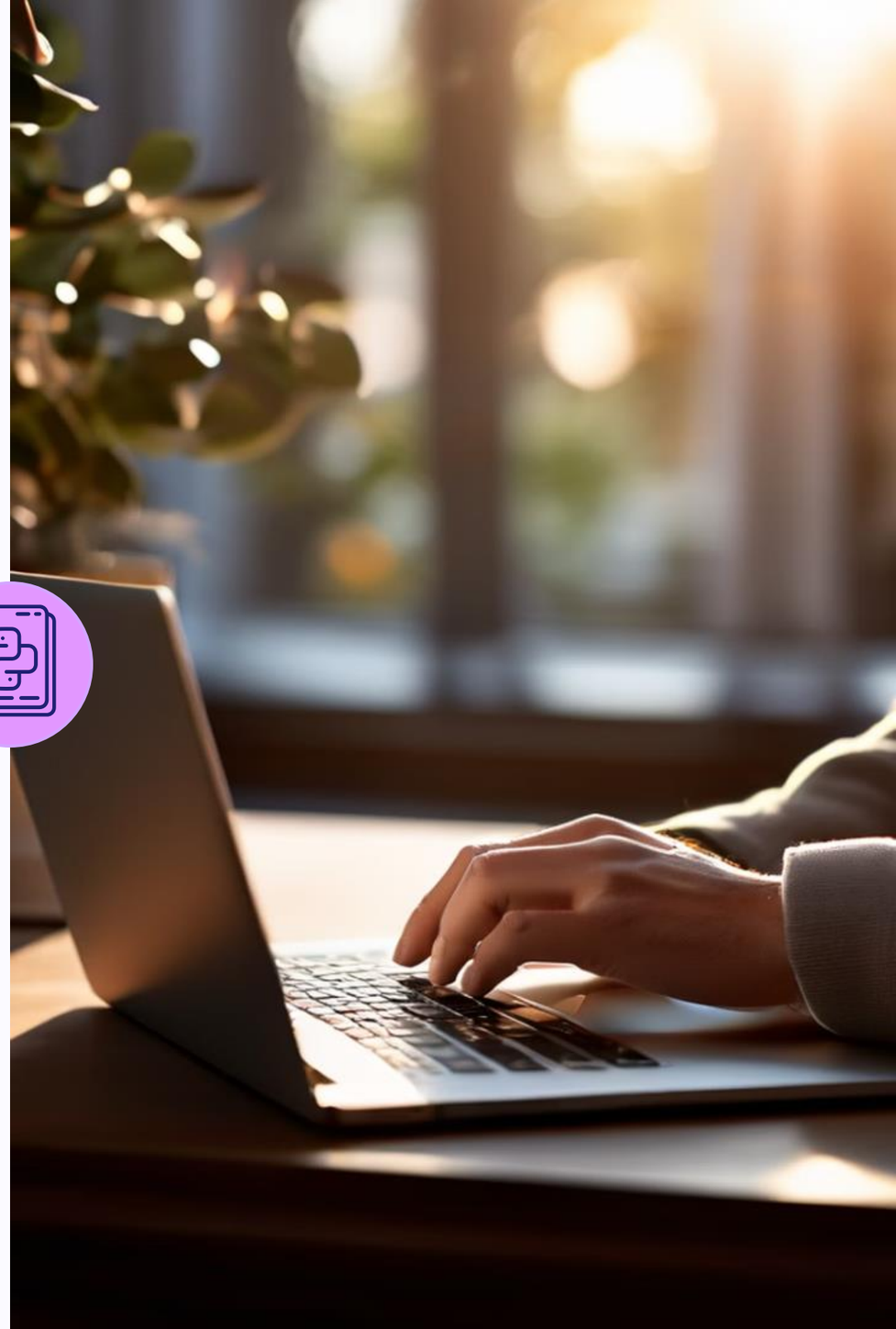
Abrir el archivo

Usa la función *open()* para abrir el archivo en modo lectura. El modo correcto es 'r' que significa 'read'. Siempre recuerda cerrar el archivo después de terminar con *close()*.

3

Leer el contenido

Existen varias funciones para leer archivos. Puedes usar *read()* para todo el contenido. También puedes usar *readline()* o *readlines()* para leer líneas específicas o todas las líneas en una lista.



Ejercicio 1: Leer un archivo de texto

```
### Lectura de archivos

```python
# Abre el archivo en modo lectura ('r')
with open('archivo.txt', 'r') as archivo:
    # Lee todo el contenido del archivo
    contenido = archivo.read()

print("Contenido del archivo:")
print(contenido)
```
```

Ejercicio 2: Escribir en un archivo de texto

1

Paso 1: Abrir el archivo

El primer paso es abrir un archivo en modo de escritura. Utiliza la función `open()` y especifica el modo 'w'. Si el archivo no existe, se creará uno nuevo.

2

Paso 2: Escribir en el archivo

Una vez abierto, puedes comenzar a escribir. Usa el método `write()` para agregar texto al archivo. Asegúrate de cerrar el archivo al finalizar para guardar los cambios.

3

Paso 3: Cerrar el archivo

Cerrar el archivo es esencial para evitar pérdidas de datos. Usa el método `close()` para liberar recursos. Esto asegura que todos los datos se guarden correctamente.

Ejercicio 2: Escribir en un archivo de texto

```
### Escritura de archivos

```python
# Abre el archivo en modo escritura ('w'). Si el archivo no existe, se crea uno nuevo.
with open('archivo.txt', 'w') as archivo:
    # Escribe una línea de texto en el archivo
    archivo.write('Hola, este es un archivo de ejemplo.\n')
    archivo.write('Esta es otra línea de texto.\n')

print("Escritura completa.")
```
```

Ejercicio 3: Lectura línea por línea, Escritura en modo anexar

````
### Lectura línea por línea

```python
# Abre el archivo en modo lectura ('r')
with open('archivo.txt', 'r') as archivo:
    # Itera sobre cada línea del archivo
    for linea in archivo:
        print(linea.strip())  # .strip() elimina los espacios en blanco al principio y al final

print("Lectura línea por línea completa.")
```

### Escritura en modo anexar

```python
# Abre el archivo en modo anexar ('a'). Si el archivo no existe, se crea uno nuevo.
with open('archivo.txt', 'a') as archivo:
    # Escribe una línea adicional de texto en el archivo
    archivo.write('Esta es una línea añadida al final del archivo.\n')

print("Anexado completo.")
```
````

Buenas prácticas en el manejo de archivos

Organización

La organización de archivos es clave para un manejo eficiente. Usa nombres descriptivos y categorías lógicas. Así, podrás localizar archivos fácilmente en el futuro.

Seguridad

Implementa medidas de seguridad para proteger tus datos. Utiliza permisos adecuados y respaldos regulares. Esto minimizará el riesgo de pérdida de información valiosa.

Consistencia

Mantén estándares consistentes en la creación y modificación de archivos. Esto incluye formato, estructura y contenido. La consistencia facilita la colaboración y el mantenimiento a largo plazo.

Recursos Adicionales

Documentación Oficial

La documentación oficial de Python es un recurso invaluable. Contiene ejemplos prácticos y descripciones detalladas. Este sitio es ideal para profundizar en funciones específicas.

Libros Recomendados

Libros como "Automate the Boring Stuff with Python" son excelentes para principiantes. También "Python Crash Course" ofrece un enfoque práctico y accesible.

Videos Tutoriales

Los tutoriales en video pueden ser muy útiles. Plataformas como YouTube y Udemy ofrecen cursos completos. Pueden facilitar el aprendizaje visual y práctico.

Comunidades en Línea

Las comunidades pueden proporcionar apoyo continuo. Foros como Stack Overflow son ideales para resolver dudas. Unirse a grupos de discusión amplía la red de aprendizaje.

Conclusiones y Resumen

Importancia de la Lectura y Escritura de Archivos

La lectura y escritura de archivos son habilidades fundamentales en Python. Permiten manejar datos de manera eficiente y automatizar tareas. Estas prácticas son esenciales para cualquier programador que desee trabajar con información persistente.

Aplicaciones Prácticas

A través de los ejercicios y casos de uso, los estudiantes pueden aplicar los conceptos aprendidos. Estos ejemplos muestran cómo la manipulación de archivos resuelve problemas reales. Aprender a gestionar archivos brinda a los programadores un valioso conjunto de herramientas.

Recomendaciones Finales

Es importante seguir buenas prácticas al trabajar con archivos. La gestión adecuada evita errores y asegura la integridad de los datos. Con el tiempo y la práctica, estas habilidades se convertirán en parte de la rutina del programador.

Ejercicios de Práctica

¡Gracias

Por ser parte de esta Experiencia de aprendizaje!