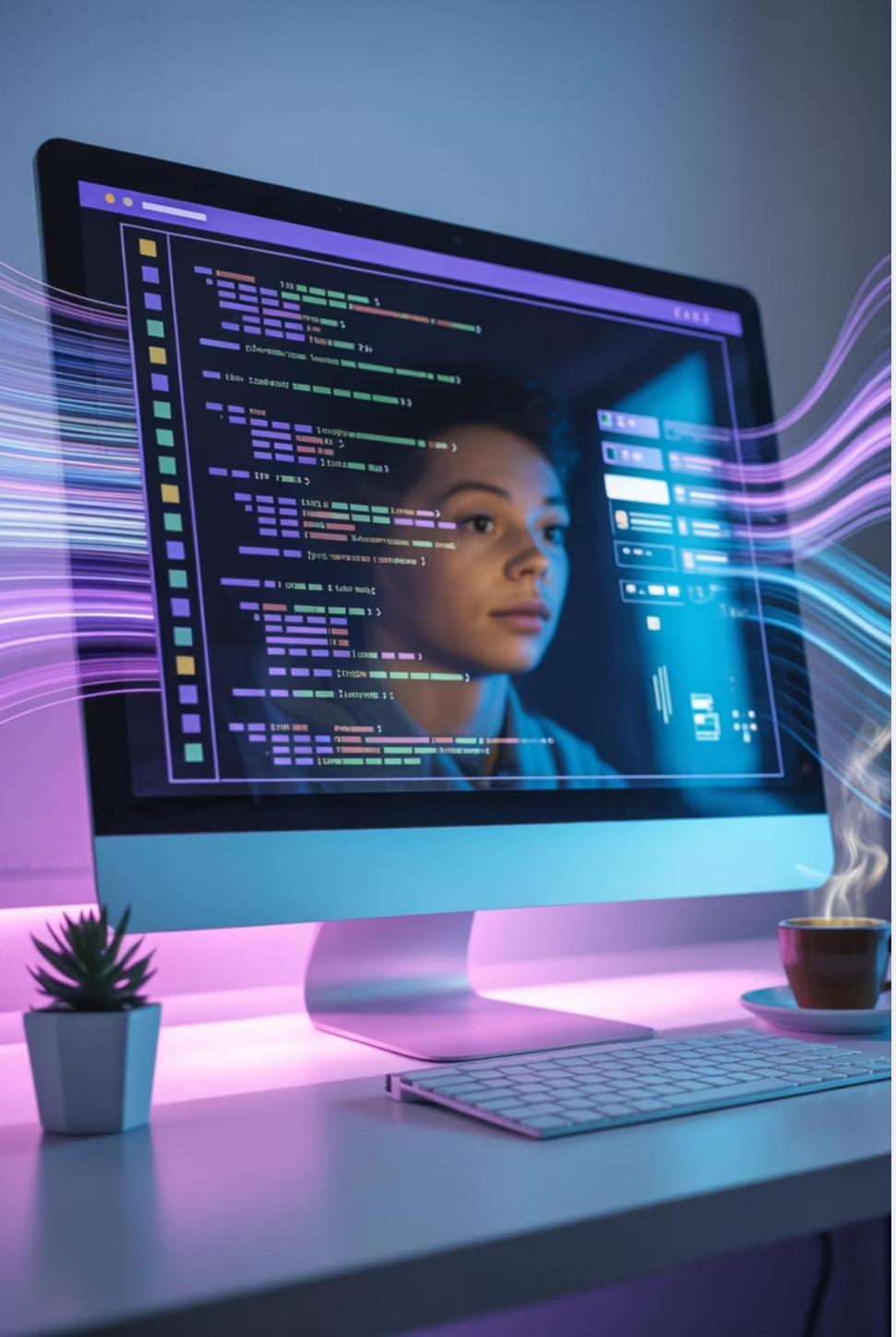


Manipulación de Archivos en Python

Una guía completa para el manejo de archivos de texto y CSV en Python, desde conceptos básicos hasta aplicaciones prácticas.

¿Por qué trabajar con archivos?

Persistencia de datos

Los datos en RAM se pierden cuando el programa finaliza. Los archivos permiten guardar información de forma permanente en el disco duro.

Intercambio de información

Los archivos facilitan el intercambio de datos entre diferentes sistemas y aplicaciones, mejorando la interoperabilidad.

Configuración y logs

Permiten leer configuraciones del sistema y registrar acciones importantes del programa para debugging y auditoría.

Tipos de archivos más utilizados

Archivos .txt

- Contienen texto plano legible
- Se pueden abrir con cualquier editor
- Útiles para listas, nombres, mensajes simples
- Formato universal y simple

Archivos .csv

- Cada línea representa una fila de tabla
- Valores separados por comas
- Fáciles de procesar con split()
- Compatibles con Excel y bases de datos

También se utilizan archivos .json para configuraciones complejas y .log para registros de eventos del sistema.

Apertura y cierre de archivos

```
# Método básico (no recomendado)archivo = open("archivo.txt", "r")contenido = archivo.read()archivo.close()# Método recomendado con withwith open("datos.txt", "r") as archivo:    contenido = archivo.read()    # El archivo se cierra automáticamente
```

El uso de `with` garantiza que el archivo se cierre correctamente, incluso si ocurre un error durante la ejecución del programa.

Modos de apertura de archivos

Modo 'r' - Lectura

Abre el archivo solo para leer. Si el archivo no existe, genera un error. Es el modo por defecto.

Modo 'w' - Escritura

Crea un archivo nuevo o sobrescribe completamente uno existente. ¡Cuidado! Borra todo el contenido anterior.

Modo 'a' - Agregar

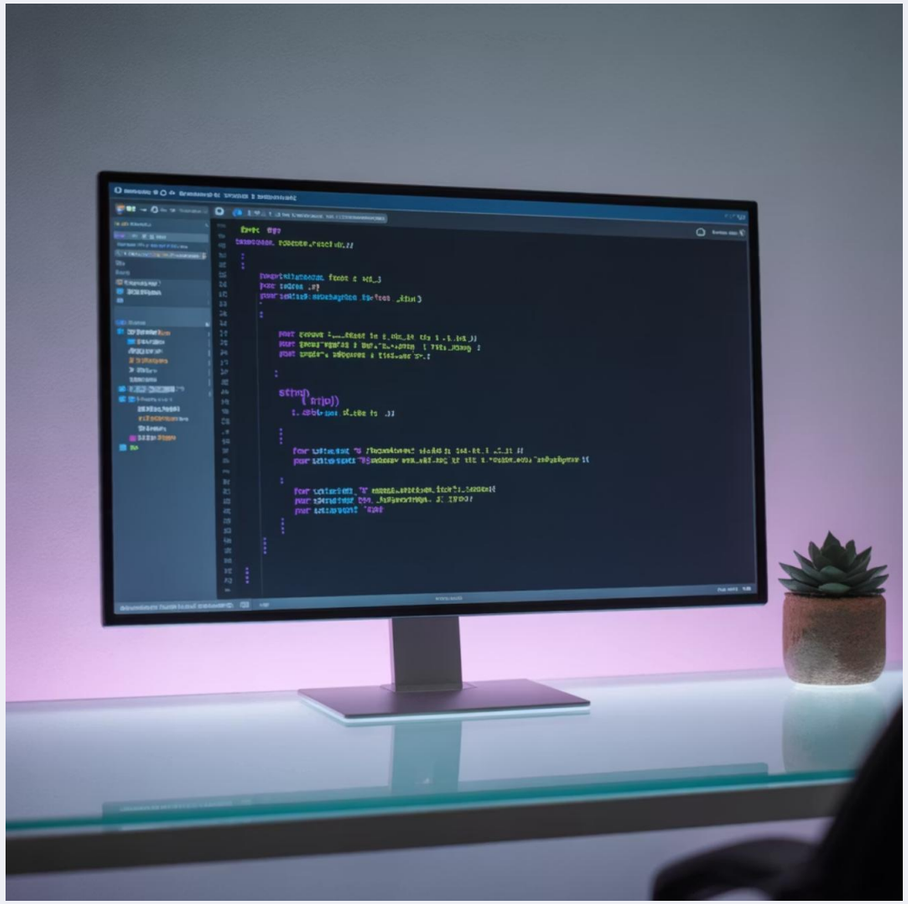
Añade contenido al final del archivo sin borrar lo que ya existe. Crea el archivo si no existe.

Lectura línea por línea

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

El método `for linea in archivo` es muy eficiente porque procesa el archivo línea por línea sin cargar todo en memoria.

La función `.strip()` elimina los saltos de línea y espacios en blanco al inicio y final de cada línea.



Procesamiento de datos con split()

```
# Ejemplo con archivo CSVlinea = "Ana García,25,España"partes = linea.split(",")# Resultado: ["Ana García", "25",
"España"]nombre = partes[0]edad = int(partes[1])pais = partes[2]
```

La función `split(",")` divide una cadena en una lista usando la coma como separador. Es fundamental para procesar archivos CSV y extraer datos estructurados.

Escritura en archivos

Sobrescribir contenido

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

Agregar contenido

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

Recuerda usar `\n` al final de cada línea para crear saltos de línea. Sin esto, todo el texto aparecerá en una sola línea.

Funciones avanzadas de lectura y escritura

readline()

Lee una sola línea del archivo. Útil cuando necesitas procesar línea por línea con control manual.

readlines()

Devuelve todas las líneas en una lista. Carga todo el archivo en memoria de una vez.

writelines()

Escribe una lista de líneas al archivo. Cada elemento de la lista se escribe como una línea separada.

Trabajando con archivos CSV

```
# Leer archivo CSV línea por líneawith open("estudiantes.csv", "r") as archivo:    for linea in archivo:        linea = linea.strip()  # Eliminar salto de línea        datos = linea.split(",")  # Dividir por comas        nombre, edad, curso = datos        print(f"Estudiante: {nombre}, Edad: {edad}, Curso: {curso}")
```

Los archivos CSV se tratan como archivos de texto normales. La diferencia está en cómo procesamos cada línea: dividiendo por comas para obtener los valores individuales.

Buenas prácticas esenciales

Usar siempre with

Garantiza que los archivos se cierren correctamente, evitando problemas de memoria y corrupción de datos.

Validar datos leídos

Los archivos pueden estar vacíos, mal formateados o contener datos inesperados. Siempre valida antes de procesar.

Cuidado con el modo 'w'

Evita sobrescribir archivos importantes accidentalmente. Usa 'a' para agregar contenido sin pérdidas.

Usar rutas relativas

Evita rutas absolutas que pueden no existir en otros sistemas. Usa nombres de archivo simples o rutas relativas.

Verificación de existencia de archivos

```
import osif os.path.exists("mi_archivo.txt"):    with open("mi_archivo.txt", "r") as archivo:        contenido = archivo.read()        print("Archivo leído correctamente")else:    print("El archivo no existe")    # Crear archivo nuevo o mostrar error
```

Usar `os.path.exists()` previene errores cuando intentas leer archivos que podrían no existir en el sistema.

Ejercicio práctico: Registro de personas

Objetivo: Desarrollar un programa que registre personas en un archivo CSV llamado "personas.csv".

Datos por persona: nombre, edad y nacionalidad

Funcionalidades: Crear archivo inicial, agregar personas una por vez, mostrar todos los registros al final.

Estructura del ejercicio

Crear archivo inicial

El programa debe crear "personas.csv" y escribir el primer registro con datos ingresados por el usuario.

Agregar más personas

Permitir al usuario agregar más personas de a una por vez hasta que decida terminar.

Mostrar resultados

Al finalizar, leer y mostrar todos los datos almacenados en el archivo con formato legible.

Formato: nombre,edad,nacionalidad (separado por comas)

Resumen y próximos pasos

Conceptos clave aprendidos

- Apertura segura con `with`
- Modos de apertura: 'r', 'w', 'a'
- Lectura línea por línea
- Procesamiento CSV con `split()`
- Buenas prácticas de validación

Siguiente nivel: Explorar el módulo `csv` de Python para manejo avanzado de archivos CSV y el módulo `json` para datos estructurados complejos.