Una Ambiciosa Introducción a Python
Parte II

En Python, trabajar con archivos es una tarea común que permite leer y escribir datos de manera persistente. Los archivos son útiles para almacenar información que debe ser recuperada más tarde, como bases de datos, configuraciones, o datos generados por programas.

Python proporciona una interfaz sencilla para manipular archivos mediante funciones integradas y métodos de objeto de archivo.

Tipos de Archivos

1. Archivos de Texto: Son archivos que contienen datos en formato legible por humanos, como .txt, .csv, .json, etc. Python permite leer y escribir en estos archivos fácilmente.
2. Archivos Binarios: Contienen datos en formato binario (no legible por humanos), como imágenes o archivos de audio. Para trabajar con archivos binarios, se utiliza un modo de apertura diferente.

Modos de Apertura de Archivos

Al abrir un archivo en Python, se especifica un modo que determina cómo se interactuará con el archivo

- 'r': Leer (Read) – Valor por defecto. Abre un archivo para lectura. El archivo debe existir, sino genera un error.
- 'w': Escribir (Write) – Crea un archivo nuevo o **sobrescribe** uno existente.
- 'a': Añadir (Append) – Abre un archivo existente y permite agregar contenido al final, sino existe lo creara.
- 'x': Crea (Create) – Crea el archivo especifico, devolverá un error si el archivo existe.
- 't': Texto (Text) – Valor por defecto, modo de texto
- 'r+': Lectura Escritura - Abre un archivo en ambos modos lectura y escritura.
- 'b': Binario (Binary) – Se usa junto con otros modos para indicar que el archivo es binario (ej. 'rb', 'wb').

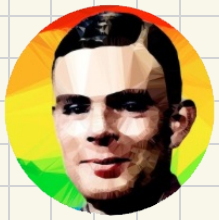
Ver file_handling.py

Un **context manager** en Python es una estructura que permite controlar la inicialización y liberación de recursos de manera automática, asegurando que ciertos procesos, como la apertura y cierre de archivos, se gestionen correctamente sin que el programador tenga que preocuparse explícitamente por ello.

La forma más común de usar un context manager en Python es a través de la instrucción with. Cuando se usa un context manager, se realiza la inicialización del recurso al entrar en el bloque de código bajo el with, y cuando se sale de ese bloque, el recurso se libera automáticamente, **aunque haya ocurrido una excepción.**

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II



Cómo funcionan los context managers

Detrás de la instrucción with, hay métodos que gestionan el contexto de ejecución. Estos son:

- __enter__(): Este método se llama cuando se entra al bloque de código del context manager. Aquí es donde se inicializan los recursos.
- __exit__(): Este método se llama al salir del bloque, y se usa para liberar los recursos. También maneja las excepciones si es necesario.

Ventajas del Uso de with

1. **Simplicidad**: Se evita el código redundante y los errores asociados con olvidar cerrar archivos.
2. **Seguridad**: Se asegura que los archivos se cierren automáticamente, lo que evita fugas de memoria y bloqueos en sistemas de archivos.
3. **Manejo de Excepciones**: Los archivos se cierran automáticamente incluso si se lanza una excepción dentro del bloque with.
4. **Legibilidad**: El código es más limpio y fácil de leer, ya que elimina la necesidad de escribir bloques try-finally o llamadas manuales a close().

Leer un archivo de texto

```
with open('archivo.txt', 'r') as archivo:
    contenido = archivo.read()
    print(contenido)
```

Escribir en un archivo de texto

```
with open('archivo.txt', 'w') as archivo:
    archivo.write("Hola, mundo!\nEste es un archivo de texto.")
```

Ejemplo Completo con Manejo de Excepciones

```
try:
    with open('archivo.txt', 'r') as archivo:
        contenido = archivo.read()
        print(contenido)
except FileNotFoundError:
    print("El archivo no existe.")
except Exception as e:
    print(f"Ocurrió un error: {e}")
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II

```
try:
    with open('archivo.txt', 'r') as archivo:
        contenido = archivo.read()
        print(contenido)
except FileNotFoundError:
    print("El archivo no existe.")
except Exception as e:
    print(f"Ocurrió un error: {e}")
```

read: Realiza la lectura de todo el archivo en un str, si queremos limitar la cantidad de caracteres a leer podemos pasarle un valor en sus argumentos.

readline: Lee la primera línea del archivo.

readlines: Realiza la lectura de todo el texto por línea y retorna una lista

Explorar splitlines()

Los archivos de textos son los mas comunes pero veamos los archivos JSON

```
import json
```

Para transformar un diccionario en un json, primero debemos importar el modulo json.

```
datos = {
    'nombre': 'Juan',
    'edad': 30,
    'ciudad': 'Madrid',
    'habilidades': ['Python', 'Java', 'SQL']
}
```

Definición del diccionario.

```
with open('datos.json', 'w') as archivo_json:
    json.dump(datos, archivo_json)

print("Diccionario guardado en 'datos.json'.")

with open('datos.json', 'r') as archivo_json:
    datos_cargados = json.load(archivo_json)

print("Contenido cargado desde 'datos.json':")
print(datos_cargados)
```

La función dump convierte el diccionario a un archivo json y luego lo guardamos usando el stream archivo_json

buscar dumps

La función load convierte el archivo json en un diccionario que estamos leyendo desde el stream archivo_json.

buscar loads

Ver example_json.py

Lic. Franco Herrera

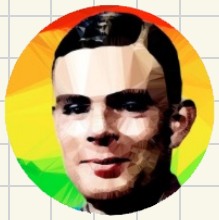
Una Ambiciosa Introducción a Python
Parte II

Ejemplo

```
try:
    with open('archivo_ejemplo.txt', 'w') as archivo:
        archivo.write("Este es el contenido inicial del archivo.\n")
        archivo.write("Incluye varias líneas de texto.\n")
    print("Información inicial escrita en 'archivo_ejemplo.txt'.")
except Exception as e:
    print(f"Ocurrió un error al escribir en el archivo: {e}")

try:
    with open('archivo_ejemplo.txt', 'r') as archivo:
        contenido = archivo.read()
    print("Contenido del archivo después de la escritura inicial:")
    print(contenido)
except FileNotFoundError:
    print("El archivo no fue encontrado.")
except Exception as e:
    print(f"Ocurrió un error al leer el archivo: {e}")

try:
    with open('archivo_ejemplo.txt', 'a') as archivo:
        archivo.write("Esta es una línea añadida posteriormente.\n")
        archivo.write("Esta es otra línea añadida.\n")
    print("Nueva información añadida a 'archivo_ejemplo.txt'.")
except Exception as e:
    print(f"Ocurrió un error al agregar información al archivo: {e}")

try:
    with open('archivo_ejemplo.txt', 'r') as archivo:
        contenido_actualizado = archivo.read()
    print("Contenido del archivo después de agregar información:")
    print(contenido_actualizado)
except FileNotFoundError:
    print("El archivo no fue encontrado.")
except Exception as e:
    print(f"Ocurrió un error al leer el archivo actualizado: {e}")
```

Lic. Franco Herrera

Una Ambiciosa Introducción a Python
Parte II



Función seek ()

La función seek() en Python es utilizada para mover el **puntero del archivo** a una posición específica dentro del archivo abierto. Este puntero indica en qué lugar del archivo se encuentra la próxima lectura o escritura.

Definición

seek(offset, whence=0): Mueve el puntero del archivo a la posición especificada por el offset (desplazamiento) desde una referencia dada. El argumento whence indica desde dónde se cuenta el desplazamiento.

Parámetros:

1. offset: Número de bytes a mover desde la referencia inicial.
2. whence (opcional): Define el punto de referencia para mover el puntero. Los valores posibles son:

- **0 (por defecto)**: El inicio del archivo.
- **1**: La posición actual del puntero.
- **2**: El final del archivo.

Función tell

La función tell() en Python se utiliza para obtener la **posición actual del puntero** dentro de un archivo abierto. Este puntero es el lugar desde donde se realizarán las próximas operaciones de lectura o escritura en el archivo.

```
def navigate_file():
    print("\n# Navegar Archivo: ")
    with open(TEXT_FILE, 'rb') as f:
        print("f.seek(3)")
        f.seek(3)
        print(f"f.read() : {f.read() }")
        print("f.seek(-2,2)")
        f.seek(-2,2)
        print(f"f.read() : {f.read() }")
        print("f.seek(0)")
        f.seek(0)
        print(f"f.read(4): {f.read(4)}")
        print(f"f.tell(): {f.tell()}")
```