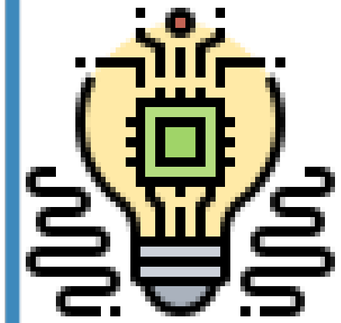
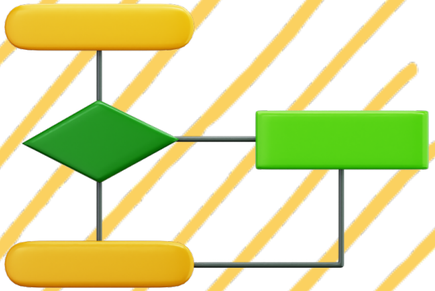


Pensamiento computacional

TEMA: ARCHIVOS



Lic. María Angela Leon

UADE

Definición de Archivo

Un archivo es una secuencia de bytes almacenada en un soporte físico duradero (un disco de estado sólido, un disco rígido o un pendrive). A diferencia de las variables que viven en la memoria RAM y se borran al finalizar el programa (volátiles), los archivos representan la persistencia de los dato

Stream (Flujo de datos)

Cuando nuestro programa interactúa con un archivo, no lo copia entero a la memoria RAM de golpe (lo cual destruiría el rendimiento si el archivo pesa gigabytes). En su lugar, Python abre un Stream (un canal o flujo de datos).

Podemos imaginar el Stream como una tubería: los datos fluyen byte por byte o carácter por carácter desde el disco hacia nuestro programa (lectura) o desde nuestro programa hacia el disco (escritura).

Operaciones Esenciales en Archivos

Cualquier manipulación de archivos en cualquier lenguaje de programación sigue estrictamente un ciclo de vida de 3 pasos. Si falta uno, el programa está incompleto o es propenso a fallas.

[1. APERTURA] → [2. OPERACIÓN (Lectura/Escritura)] → [3. CIERRE]

1. Apertura: El programa le pide permiso al Sistema Operativo para acceder a una ruta del disco. Si el SO lo autoriza, nos devuelve un descriptor de archivo (un objeto que representa ese canal abierto).

Operaciones Esenciales en Archivos

2. Operación:
   - o Lectura (Read): El flujo va del disco a la memoria RAM (variables).
   - o Escritura (Write / Append): El flujo va de la memoria RAM al disco.

3. Cierre: Se rompe el canal. El Sistema Operativo libera el archivo para que otros programas puedan usarlo y se asegura de que no quede información "atrapada" en la memoria intermedia (Buffer)

Acceso Tradicional: La función open() y close()

La forma nativa de iniciar este ciclo es mediante la función integrada open().

Sintaxis Básica

objeto_archivo = open("nombre_del_archivo.txt", "modo")

- nombre_del_archivo.txt: Un string con la ruta (relativa o absoluta).
- modo: Un string de un solo carácter que define qué operación vamos a hacer (por ejemplo, "r" para Read o "w" para Write).

Acceso Tradicional: La función open() y close()

```
# 1. Apertura
archivo = open("notas_parcial.txt", "w", encoding="utf-8")

# 2. Operación de escritura
archivo.write("Estructuras de Datos en Python\n")

# 3. Cierre obligatorio
archivo.close()
```

Acceso Tradicional: Los 3 Modos Básicos de Acceso (Lectura, Escritura y Anexado)

A. Modo Lectura ('r')

Abre el archivo existente para leerlo. Si el archivo no existe, el programa se detiene con un error (FileNotFoundError). El Cursor: Se coloca al principio (posición 0).

Ejemplo de lectura completa:

Abrimos el archivo

archivo = open("alumnos.txt", "r", encoding="utf-8")

Leemos todo el contenido de golpe y lo guardamos en una variable de tipo string

contenido = archivo.read()

¡Siempre cerrar al terminar!

archivo.close()

Acceso Tradicional: Los 3 Modos Básicos de Acceso (Lectura, Escritura y Anexado)

A. Modo Lectura (‘w')

Abre el archivo para escribir. Si el archivo ya existía, borra todo su contenido (lo trunca a 0 bytes). Si no existía, crea un archivo nuevo vacío. El Cursor: Se coloca al principio.

Ejemplo:

```
# Abrimos el archivo
archivo = open("alumnos.txt", “w", encoding="utf-8")
# Escribimos líneas. Atención!!!  Python NO agrega saltos de línea solos, hay que usar \n
archivo.write("Primer parcial: 24 de Mayo\n")
archivo.write("Segundo parcial: 28 de Junio\n")
archivo.close()
```

Acceso Tradicional: Los 3 Modos Básicos de Acceso (Lectura, Escritura y Anexado)

A. Modo Lectura (‘a')

Abre el archivo para agregar datos al final. No borra nada de lo que ya existía. Si el archivo no existe, lo crea.. El Cursor: Se coloca al final del archivo.

Ejemplo:

```
# Abrimos el archivo
archivo = open(“salida.txt", “w", encoding="utf-8")
# Escribimos líneas. Atención!!!  Python NO agrega saltos de línea solos, hay que usar \n
# Esto se escribirá justo debajo de lo que ya existía
archivo.write("Recuperatorio: 5 de Julio\n")
archivo.close()
```

Variantes de Lectura: ¿Cómo procesar la información?

Supongamos que el archivo notas.txt contiene:

Gomez,8

Perez,7

Lopez,10

Variante 1: read() — Ideal si el archivo es chico y solo queremos ver o procesar el texto completo como un único gran String.

```
archivo = open("notas.txt", "r", encoding="utf-8")
todo = archivo.read()
print(todo) # Devuelve el string completo con sus saltos de línea
archivo.close()
```

Variantes de Lectura: ¿Cómo procesar la información?

Variante 2: readlines() — Almacenar en una Lista

Lee todo el archivo y crea una lista de Python, donde cada elemento de la lista es una línea del archivo (un String).

```
archivo = open("notas.txt", "r", encoding="utf-8")
lista_lineas = archivo.readlines()
archivo.close()
# Ahora podemos usar un bucle tradicional para recorrer la lista
for linea in lista_lineas:
    print("Línea procesada:", linea.strip()) # .strip() quita el \n invisible del final
```

Variantes de Lectura: ¿Cómo procesar la información?

Variante 2.5: El bucle for directo sobre el archivo

En Python, el objeto archivo se puede recorrer directamente con un for línea por línea, sin necesidad de cargar todo en una lista intermedia.

```
archivo = open("notas.txt", "r", encoding="utf-8")


for linea in archivo:
    print("Estudiante:", linea.strip())


archivo.close()
```

Acceso Tradicional: Posibles dificultades

¿Qué pasa si entre la línea del write y la línea del close ocurre un error inesperado (por ejemplo, una división por cero o un dato nulo)?

```
archivo = open("notas_parcial.txt", "w", encoding="utf-8")
archivo.write("Estructuras de Datos en Python\n")
resultado = 10 / 0  # ¡El programa explota aquí!
archivo.close()  #  Esta línea NUNCA se ejecuta
```

¿Cómo hacemos para controlar este error?

Acceso Seguro: with open(...)

La forma estándar y profesional de trabajar con archivos en Python es utilizando la sentencia with. A esto se le conoce técnicamente como Administrador de Contexto (Context Manager).

Sintaxis con with

```
with open("notas_parcial.txt", "w", encoding="utf-8") as archivo:
    archivo.write("Estructuras de Datos en Python\n")
    # Podemos hacer más operaciones aquí dentro...

# Al llegar acá (fuera de la indentación), el archivo YA SE CERRÓ automáticamente.
```

Acceso Seguro: with open(...)

El bloque with encapsula la apertura y el cierre de forma segura.

1. with open(...) as archivo: abre el archivo y crea la variable archivo.
2. Se ejecuta todo el código que esté indentado (dentro del bloque).
3. Liberación Automática: En el momento exacto en que el código vuelve a la alineación de la izquierda (fuera del with), Python ejecuta internamente el método de cierre.

La ventaja clave: Incluso si el código de adentro falla catastróficamente o el programa lanza una excepción en medio del bloque, Python garantiza que el archivo se cerrará correctamente antes de que el programa muera. No requiere un .close() explícito.

EJEMPLO

Tenemos un archivo llamado temperaturas.txt con registros de un sensor (un valor por línea).

Necesitamos leer esos datos, calcular el promedio de las temperaturas y guardar el resultado resumido en un nuevo archivo llamado reporte.txt.