ESTUDIOS SUPERIORES DE BUENOS AIRES
ESBA

Archivos

Archivos

Un **archivo** en informática es una secuencia de bytes almacenados en un dispositivo. Un archivo es identificado por un nombre, la ruta y el dispositivo.

La importancia del almacenamiento de contenidos es obvia: envío de información a otros usuarios, posponer el trabajo varios días o semanas sin tener que introducir manualmente los datos de nuevo, acceso a información almacenada en sistemas remotos, etc. Incluso para desarrollos de software de relativamente corta longitud resulta relevante la gestión de datos, por ahorrar una cantidad de tiempo considerable.

Podemos pensar en los archivos de forma análoga a lo que sería un archivo físico: un lugar donde hay información almacenada.

Métodos de Acceso

SECUENCIAL

Se leen y escriben los datos de modo que si se quiere acceder a un dato que está hacia la mitad de un archivo, habrá que pasar primero por todos los datos anteriores. Los ficheros de texto son de acceso secuencial.

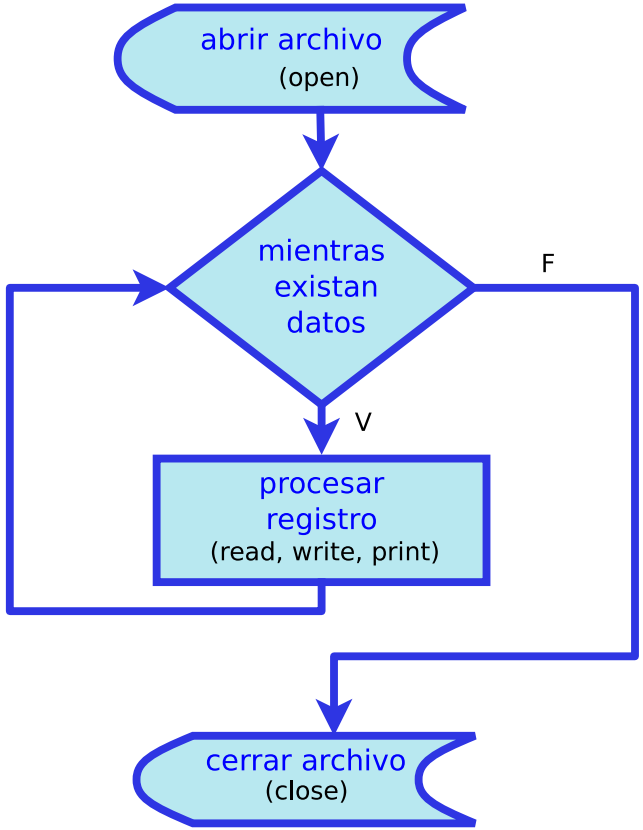
ALEATORIO

Permiten acceder directamente a un dato sin tener que pasar por todos los demás, y pueden acceder a la información en cualquier orden. Tienen la limitación de que los datos están almacenados en bloques que se llaman registros, y que todos los registros que se almacenan en un fichero deben ser del mismo tamaño.

BINARIO

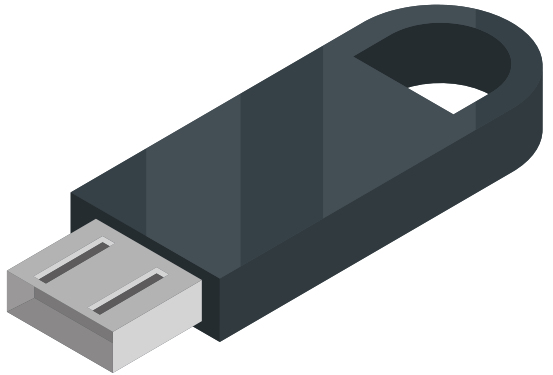
Son como los de acceso aleatorio, pero el acceso no se hace por registros sino por bytes.

Proceso secuencial



Vamos a estudiar únicamente ciertas formas de extraer y guardar datos de un archivo secuencial, a través de algunas funciones disponibles en el lenguaje para este fin. Esto será suficiente para poder crear programas que leen y guardan datos desde archivos de forma simple.

La mayoría de lenguajes propone un proceso de doble lectura. Una lectura antes del mientras y todas las demás lecturas como última instrucción del ciclo ya que la última lectura no traerá datos para procesar.



Apertura

La función **open**() nos devuelve un objeto **file**() que es un enlace hacia el **archivo físico** indicado en el primer argumento mediante el nombre (y la ruta) donde están almacenados o se quieren almacenar los datos, si no se especifica la ruta completa del archivo, por defecto python asumirá que el directorio (o carpeta) es el mismo donde se halla el archivo .py

El segundo argumento indica el modo de apertura que indica para qué se utilizará el archivo. Los modos más importantes son los siguientes:

Modo	Acción
r	Lectura únicamente.
w	Escritura únicamente, reemplazando el contenido actual del archivo o bien creándolo si es inexistente.
a	Escritura únicamente, manteniendo el contenido actual y añadiendo los datos al final del archivo.

En el caso de que el modo sea de lectura, la variable *archivoLogico* contendrá todas las líneas que compongan al archivo y las podremos recorrer con un ciclo for de forma individual.

```
archivoLogico = open (archivoFisico, modo)
```

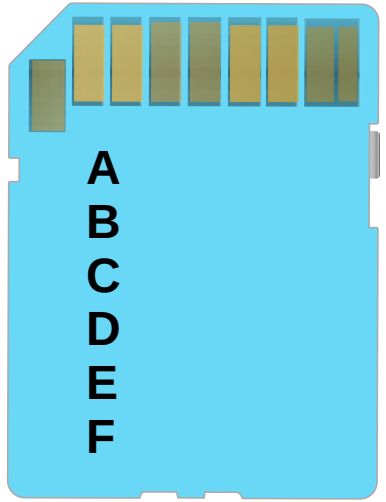
Lectura de archivos

Si necesitamos abrir un archivo que mantenga las acentuaciones y caracteres especiales, podemos agregarle adicionalmente como parámetro la codificación del mismo:

```
contenido = open ("archivo.txt", "r", encoding="UTF-8")
```

Veamos cómo utilizar **open**() para abrir un archivo y recorrer línea por línea su contenido para mostrarlo en pantalla:

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/letras.txt", "r")
    #Recorremos y mostramos cada línea
    for linea in contenido:
        print(linea)
    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```

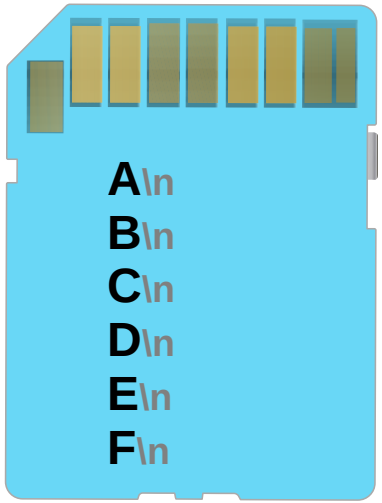


archivos/letras.txt

Lectura completa

Es posible, además, obtener todas las líneas del archivo utilizando una sola llamada con el método **read**(), por ejemplo:

```
try:
    #Abrimos en modo solo lectura
    contenido = open("archivos/letras.txt", "r")
    lineas = contenido.read()
    print(lineas)
    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```



archivos/letras.txt

Siempre que una instrucción cargue un archivo completo en memoria se debe tener el cuidado de utilizarla sólo con archivos pequeños, ya que podría agotarse la memoria de la computadora o causar problemas de rendimiento.

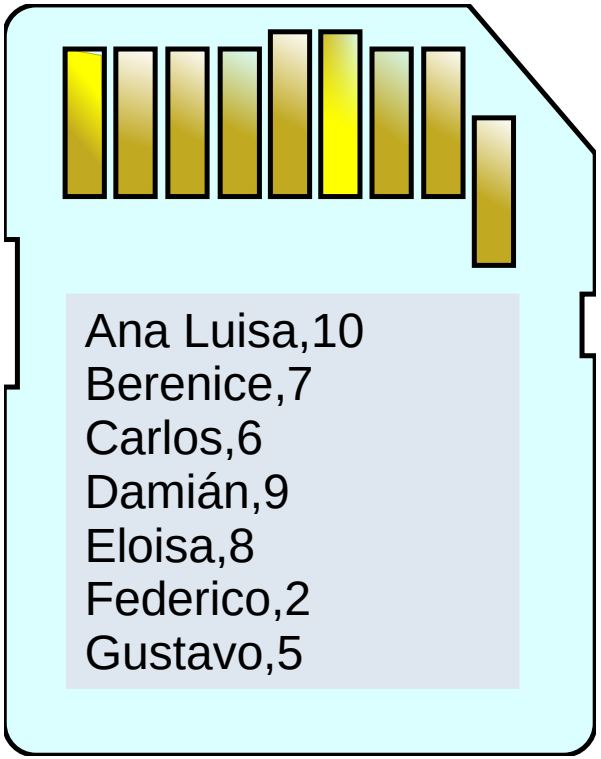
Cada línea o renglón dentro del archivo, constituye un registro. La longitud es variable y para reconocer el fin de cada una de las líneas se utiliza un delimitador (\n) que representa un salto de línea:

A	\n	B	\n	C	\n	D	\n	E	\n	F	\n
BYTE	1	2	3	4	5	6	7	8	9	10	11	12

Lectura de CSV

También, podríamos procesar un documento con un formato preestablecido, tal como un CSV (valores separados por coma), tal como se muestra en el siguiente ejemplo:

```
try:
    #Abrimos en modo solo lectura
    archivo = open("archivos/notas.csv", "r")
    contenido = archivo.read()
    filas = contenido.split("\n")
    #Recorremos y mostramos cada línea
    for fila in filas:
        nombre, nota = fila.split(",")
        print(f"Nombre: {nombre} - Nota: {nota}")
    contenido.close()
except FileNotFoundError:
    print("No existe el archivo")
```



Los archivos CSV son un estándar para el intercambio de datos, se pueden leer/escribir desde muchas aplicaciones como por ejemplo planillas de cálculos o bases de datos.

No siempre el separador de campos es una coma, puede ser punto y coma, tabulado (‘\t’), u otro símbolo.

Escribir datos

El método .**write**() se puede utilizar para escribir en un archivo ya abierto. El único argumento es el texto que vayamos a agregar. Veamos un ejemplo:

```
#Lista de nombres a escribir
nombres = ["Armando", "Beatriz", "Claudia"]
#Abrimos en modo escritura
archivo = open("archivos/personas.txt", "w")
#Recorremos la lista y agregamos cada nombre al archivo
for nombre in nombres:
    archivo.write(nombre + "\n")
#Cerramos el archivo
archivo.close()
```

Para cerrar el archivo debemos utilizar el método .**close**(). Eso permitirá que el contenido sea finalmente escrito en el archivo. El método .**write**() no agrega saltos de línea, así que debemos ponerlos con "\n" para cada línea que deseamos escribir en el archivo.

Escribir datos

Otro modo de escribir en un archivo usando la función print(). Podemos utilizarla para imprimir valores en un archivo con el parámetro file y sacando ventaja de su funcionamiento habitual (saltos de línea, espacios y demás).

```
#Lista de nombres a escribir
nombres = ["Armando", "Beatriz", "Claudia"]
#Abrimos en modo escritura
archivo = open("archivos/personas.txt", "w")
#Recorremos la lista y agregamos cada nombre al archivo
for nombre in nombres:
    print(nombre, file=archivo)
#Cerramos el archivo
archivo.close()
```

Al usar la función print() nos evitamos poner el salto de línea "\n", ya que print() lo hace automáticamente en cada llamado.

Añadir datos

Modificando el modo de escritura (segundo parámetro con la letra a), agregaremos nuevo contenido hacia el final del archivo que estamos haciendo referencia:

```
#Abrimos en modo escritura
archivo = open("archivos/personas.txt", "a")
nombre = " "
while(nombre != ""):
    nombre = input("Ingrese un nombre: ")
    if (nombre != ""):
        print(nombre, file=archivo)
#Cerramos el archivo
archivo.close()
```

En caso de que el archivo no exista, lo crea.