



**Universidad Tecnológica Nacional**  
**Facultad Regional Córdoba**

# **Algoritmos y Estructuras de Datos**

## **Ficha 22**

### **Archivos**



# Temario

- Archivos
  - Introducción
- Archivos Binarios en Python
  - Operaciones Básicas
    - Abrir. Cerrar.
    - Leer. Escribir.
  - Serialización
- Archivos Binarios: Recorrido Secuencial y Acceso Directo.

# Archivos: Introducción

- Es una estructura de datos lineal almacenada en dispositivos de almacenamiento externos (como discos, pendrive, DVD etc.).
- El almacenamiento de los datos es permanente (no volátil) sin perderlos cada vez que el programa finaliza.
- Permite la gestión de grandes volúmenes de datos.
- Pueden ser utilizados por cualquier otro programa.
- Desde cualquier programa es posible agregar, eliminar y modificar datos.

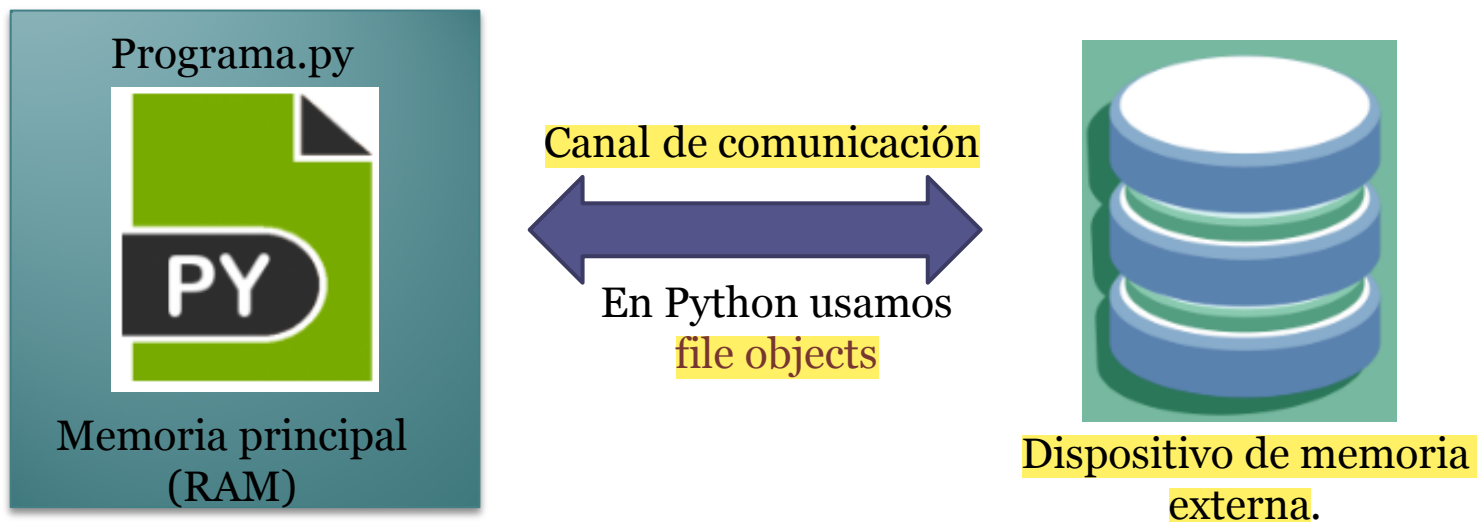


# Archivos: Introducción

- Los **archivos** almacenan sus datos en **sistema binario**.
- Cada dato almacenado utiliza tantos **bytes** como sea necesario para representar ese dato. **Por lo tanto el tamaño de un archivo esta determinado por la “cantidad de bytes” que contiene.**
- **En la práctica, cabe hacer una diferenciación entre:**
  - **Archivos de texto:** todos los bytes del archivo son interpretados y visualizados como caracteres ( por ejemplo de acuerdo a la tabla de código ASCII).
  - **Archivos binarios:** Las secuencias o bloques de bytes que el archivo contiene representan información de cualquier tipo (caracteres, valores booleanos, etc.) y no asume que cada byte representa un carácter.

# Operaciones básicas: Apertura de archivo

- La función `open()` es la encargada de abrir el canal de comunicación entre el dispositivo que contiene al archivo y la memoria principal.
- El objeto creado y retornado por la función (file object) se aloja en memoria y contiene diversas funciones que permiten manejar el archivo (manejador del archivo).
- La sintaxis es: `open(nombre físico, modo de apertura)`



# Operaciones básicas: Apertura del archivo

Ejemplo de modo de apertura de un archivo:

```
m = open("datos.dat", "wb")
```



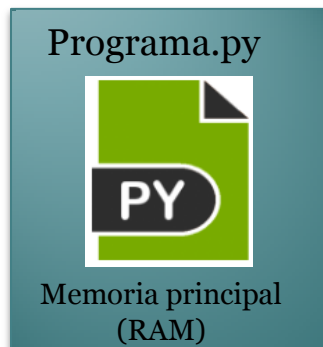
El archivo se busca en la carpeta actual del proyecto o programa que se este ejecutando.

```
# crea una variable m asociada a un file object y
# deja abierto el archivo datos.dat en modo de
# grabación.
```

Otra variante de apertura de archivo válida:

```
m = open("c:\\files\\datos.dat", "wb")
```

Se especifica el directorio de acceso del archivo.



Canal de comunicación



file objects

```
m = open("datos.dat", "wb")
```



Dispositivo de memoria externa.

# Modos Apertura de archivos binarios

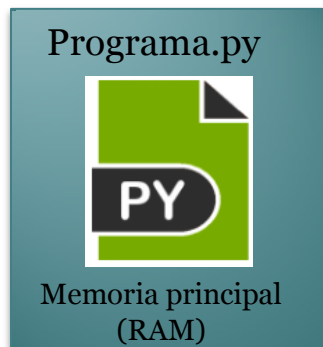
Modo	Significado
<b>rb</b>	El archivo se abre como <i>archivo binario</i> en modo de <i>sólo lectura</i> . No será creado en caso de no existir previamente.
<b>wb</b>	El archivo se abre como <i>archivo binario</i> en modo de <i>sólo grabación</i> . Si ya existía su contenido será eliminado. Si no existía, será creado.
<b>ab</b>	El archivo se abre como <i>archivo binario</i> en modo de sólo <i>append</i> (todas las grabaciones se hacen al final del archivo, preservando su contenido previo si el archivo ya existía). Si no existía, será creado.
<b>r+b</b>	El archivo se abre como <i>archivo binario</i> en modo de <i>lectura y grabación</i> . El archivo debe existir previamente: no será creado en caso de no existir.
<b>w+b</b>	El archivo se abre como <i>archivo binario</i> en modo de <i>grabación y lectura</i> . Si ya existía su contenido será eliminado. Si no existía, será creado.
<b>a+b</b>	El archivo se abre como <i>archivo binario</i> en modo de <i>lectura y de append</i> (todas las grabaciones se hacen al final del archivo, preservando su contenido previo si ya existía). Si no existía, será creado.

# Operaciones básicas: Cierre del archivo

- La forma de **cierre de los archivos** se puede realizar de dos formas:
  - Automático**: se presenta cuando la variable (file object) para acceder al archivo sale del ámbito donde esta definida.
  - Manual**: invocando la función **close()**. Ejemplo:

```
m = open('datos.dat', 'wb')
m.close()
```

- El programador debe estar seguro que el archivo que estuvo manejando se cierre oportunamente, ya sea en forma manual o en forma automática.



Dispositivo de memoria externa.



# Operaciones básicas: Lectura y escritura

- Los objetos de tipo `file` (`file object`) que se crean con `open()` contienen numerosas funciones adicionales para el manejo de archivos.
- En el caso de los `archivos de textos` se dispone:
  - Para la `lectura` de datos: `read()`
  - Para la `escritura` de datos: `write()`
- Ambas son directas y simples de usar cuando se trata de archivos de textos, pero no son tan directas cuando se trata de un archivo de tipo binario.

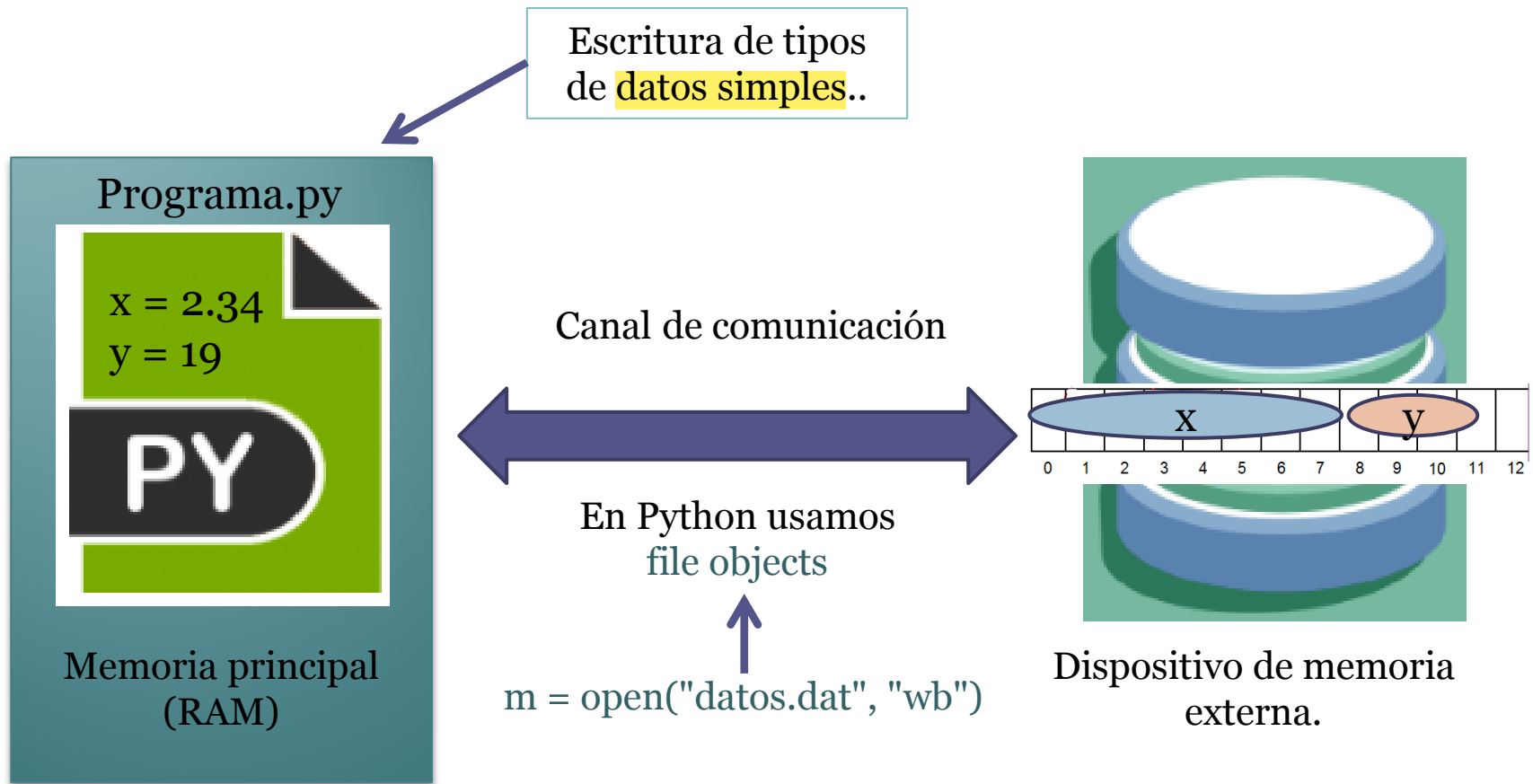


# Operaciones básicas: Lectura y escritura

- En el caso de los **archivos binarios** se puede hacer:
  - Lectura y grabación de **datos simples**: entero, float entre otros.
  - Lectura y grabación de **datos compuestos**: registros, listas etc.



# Operaciones básicas: Lectura y escritura



# Operaciones básicas: Lectura y escritura

- El proceso de **lectura y escritura** se realiza mediante un proceso denominado **serialización**.
- La “**serialización**” es un proceso por el cual el **contenido de una variable** normalmente de estructura compleja se **convierte automáticamente en una secuencia de bytes** listos para ser almacenados en un archivo.
- **Pero de tal forma que luego esa secuencia de bytes puede recuperarse desde el archivo y volver a crear con ella la estructura de datos original.**
- Para el mecanismo de serialización en Python se usará el **módulo pickle**. Este módulo provee ciertas funciones:
  - **Para la escritura: `dump()` y**
  - **Para la lectura: `load()`**

# Operaciones básicas: escritura (tipo simple)

- Para **grabar los datos** en un archivo se hace mediante la función **dump()**.
- Ejemplo como **grabar datos simples en un archivo binario.**

```
import pickle

print('Procediendo a grabar números en el archivo')

m = open('prueba.dat', 'wb')
x, y = 2.345, 19

pickle.dump(x, m)
pickle.dump(y, m)

m.close()
```

**dump(arg1,arg2):**

- 1 arg = el valor a grabar.
- 2 arg = manejador al archivo (file object)

# Operaciones básicas: lectura (tipo simple)

- Para leer los datos en un archivo se hace mediante la función `load()`.
- Ejemplo como leer datos simples en un archivo binario

```
import pickle
```

```
print('Procediendo a leer números en el archivo')
```

```
m = open('prueba.dat', 'wb')
```

```
a = pickle.load(m)
```

```
b = pickle.load(m)
```

```
m.close()
```

```
print('Datos recuperados:', a , '-', b)
```

`load (arg1)`

- 1 arg= manejador al archivo (file object)

# Operaciones básicas: Lectura y escritura (tipo compuesto)

- Lectura y grabación de una variable registro en un archivo binario. Ejemplo: Guardar en el archivo tres variables de tipo registro Libro.

```
import pickle

class Libro:
    def __init__(self, cod, tit, aut):
        self.isbn = cod
        self.titulo = tit
        self.autor = aut

def display(libro):
    print('ISBN:', libro.isb, end='')
    print('Titulo:', libro.titulo, end='')
    print('Autor:', libro.autor)
```

# Operaciones básicas: Lectura y escritura (tipo compuesto)

- Lectura y grabación de una **variable registro** en un archivo binario. Ejemplo: Guardar en el archivo tres variables de tipo registro Libro.

```
def test:
    print('Prueba de grabación de registro')
    lib1 = Libro(123, 'Libro1', 'Autor1')
    lib2 = Libro(234, 'Libro2', 'Autor2')
    fd = 'libros.dat'
    m = open(fd, 'wb')
    pickle.dump(lib1,m)
    pickle.dump(lib2,m)

    lib1 = pickle.load(m)
    lib2 = pickle.load(m)
    m.close()
    display(lib1)
    display(lib2)
```

Graba registros Libro

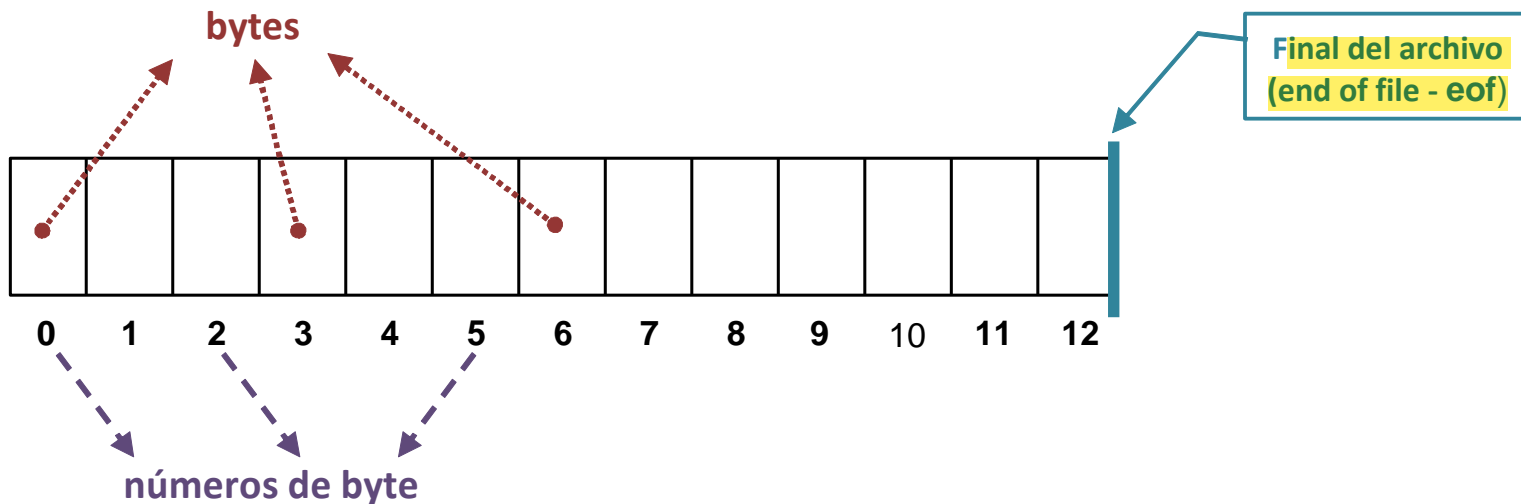
Lectura de registros Libro

```
if __name__ == '__main__':
    test()
```



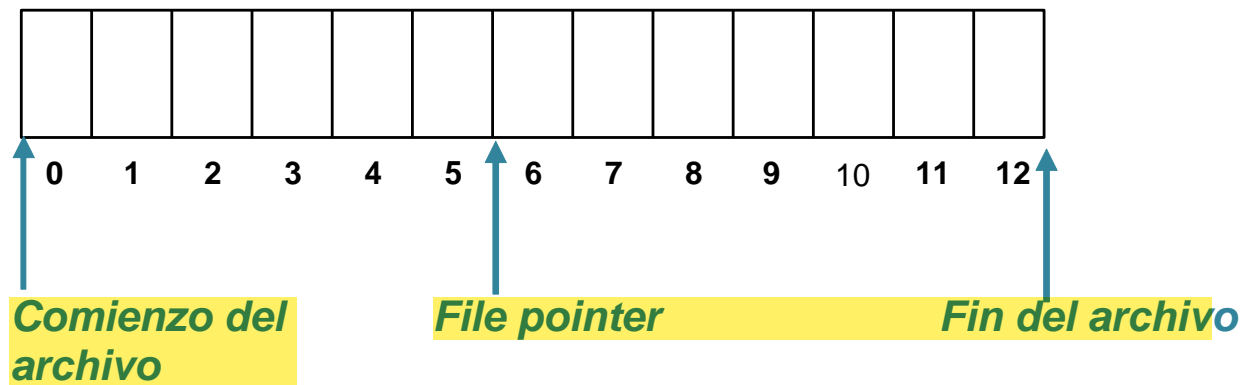
## El objeto File Pointer (i)

- La **estructura de un archivo** puede entenderse con un **arreglo de bytes** en **memoria externa**. Cada componente de este arreglo es uno de los bytes grabados en el archivo, y cada byte tiene a modo de índice un número que lo identifica, correlativo de cero en adelante.



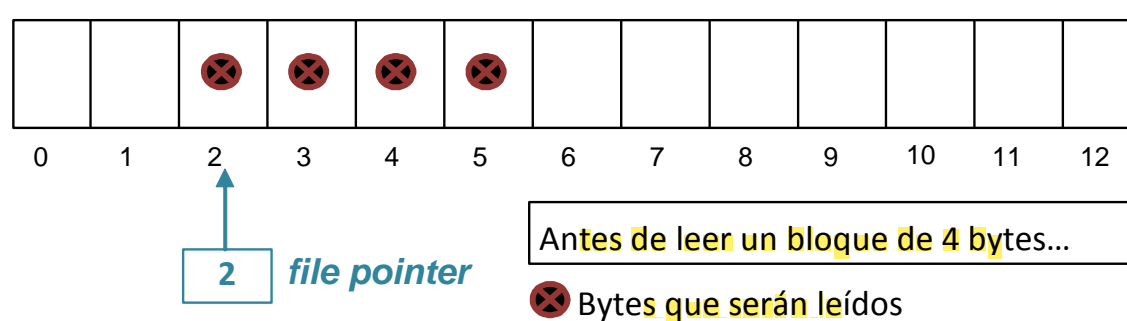
## El objeto File Pointer (ii)

- **File Pointer:** cursor o indicador o marcador interno, es una de tipo entero que contiene el número del byte sobre el cual se realizará la próxima operación de lectura o de grabación.
- **Comienzo del archivo:** Es el byte cero, el primer byte del archivo.
- **Fin del archivo:** marca de final del archivo o end of file (eof).

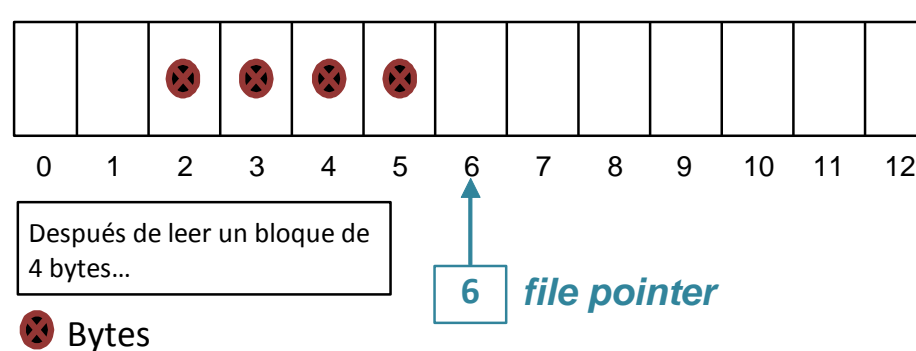


## El objeto File Pointer (iii)

- Esquema del estado del File Pointer en un archivo antes de proceder la lectura.



- Esquema del estado del File Pointer en un archivo después de proceder la lectura.



## El objeto File Pointer (iv)

- Es gestionado automáticamente por las siguientes funciones:
  - `open()` → file pointer queda apuntado al inicio del archivo. Es decir en el byte cero.
  - En un proceso de serialización (lectura-grabación):
    - `dump()` → file pointer queda apuntando en el byte que indica final del registro grabado.
    - `load()` → file pointer queda apuntando en el byte que indica el final del registro leído.

# Funciones útiles para recorrer un archivo de manera secuencial

- Para obtener el tamaño en bytes de un archivo, se utiliza la función `getsize()` incluida en el módulo `os.path`:

```
import os.path
```

```
t = os.path.getsize('libros.dat')  
print(t)
```

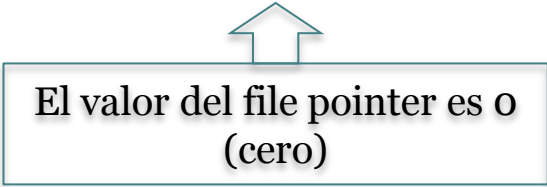
- La función `os.path.getsize()` toma como parámetro el nombre físico del archivo y retorna su tamaño en bytes.
- No es necesario que el archivo este abierto con la función `open()` para usar dicha función.

# Funciones útiles para recorrer un archivo de manera secuencial

- La función `tell()` retorna la ubicación actual del “file pointer” en forma de un número entero (número de byte).

Ejemplo:

```
m = open('prueba.dat', 'wb')  
pos = m.tell()  
print('El valor del file pointer: ', pos)
```



El valor del file pointer es 0  
(cero)

- Es muy útil cuando por algún motivo se quiere conocer en qué byte está posicionado un archivo en un momento dado.

# Archivos binarios: recorrido secuencial

- Como ya analizamos....
- Para grabar un **registro** en un archivo usamos **dump()**

```
fd = 'libros.dat'
m = open(fd, 'ab')
pickle.dump(lib1, m)
pickle.dump(lib2, m)
pickle.dump(lib3, m)
pickle.dump(lib4, m)
pickle.dump(lib5, m)
m.close()
print('Se grabaron varios registros en el archivo', fd)
```

# Archivos binarios: recorrido secuencial

```
m = open(fd, 'rb')
t = os.path.getsize(fd)

print('Se recuperaron estos registros desde el archivo', fd, ':')
while m.tell() < t:
    lib = pickle.load(m)
    display(lib)
m.close()
```

← Chequea que el valor actual del puntero sea menor que el tamaño total.



# Archivos binarios: acceso directo

- La función `seek()` permite **cambiar** el valor del **file pointer**, y elegir cuál será el próximo byte que será leído o grabado.
- Para poder ser usado debe incluirse en el programa: `import io.`
- Y se supone que el **archivo** debe estar previamente **abierto**.

Forma general:

```
seek(offset, from_what)
```

- El **primer parámetro indica cuántos bytes debe moverse** el file pointer.
- El **segundo parámetro indica desde que lugar (número de bytes) se hace ese salto.**

# Archivos binarios: acceso directo

- Constantes predefinidas para el método `seek()`

Constante	Valor	Significado
<code>io.SEEK_SET</code>	0	Reposicionar comenzando desde el principio del archivo. El valor del primer parámetro (offset) puede ser 0 o positivo (pero no negativo).
<code>io.SEEK_CUR</code>	1	Reposicionar comenzando desde la posición actual del puntero de registro activo. El valor de offset puede ser entonces negativo, 0 o positivo.
<code>io.SEEK_END</code>	2	Reposicionar comenzando desde el final del archivo. El valor de offset típicamente es negativo, aunque puede ser 0 o positivo.

# Archivos binarios: acceso directo

- Permiten mover el file pointer para leer o modificar bytes en una posición definida.
- Ejemplos para cambiar el valor de file pointer (puntero). En este caso consideramos que la variable manejadora del archivo se llama “m”.

- Parados al comienzo del archivo (byte 0) y saltar al valor 10:

```
m.seek(10, io.SEEK_SET)
```

- Si el valor del file pointer es 7 y se quiere saltar al byte 4:

```
m.seek(-3, io.SEEK_CUR)
```

# Bibliografía

- [1] V. Frittelli, Algoritmos y Estructuras de Datos, Córdoba: Universitas, 2001.
- [2] Python Software Foundation, "Python Documentation," 2015. [Online]. Available: <https://docs.python.org/3/>. [Accessed 24 February 2015].
- [3] M. Pilgrim, "Dive Into Python - Python from novice to pro," 2004. [Online]. Available: <http://www.diveintopython.net/toc/index.html>. [Accessed 6 March 2015].