3.2.1 GESTIÓN DE ARCHIVOS (1)

Los archivos son necesarios para tener información de forma permanente. Cuando termina una aplicación, la información que contiene en la memoria RAM deja de estar accesible y puede ser sobreescrita por otros procesos del ordenador. Al apagar el ordenador se pierde esa información.

Los archivos los gestiona el sistema operativo a través de su sistema de archivos. Python es capaz de comunicarse con (casi) cualquier sistema de archivos, con lo que podemos leer o escribir archivos en cualquier ordenador (si tenemos los permisos adecuados).

Para lo que nos interesa, un archivo es simplemente una secuencia finita de bytes. Para leer y/o guardar archivos se han de abrir indicando qué queremos hacer con ellos y de qué modo queremos abrirlo, como secuencia de caracteres (por defecto en Python) o como secuencia de bytes. Una vez no sea necesario que esté abierto hemos de cerrarlo, para que el sistema operativo lo sepa.

Para leer o guardar archivos binarios (imágenes...) usaremos algún módulo que sepa cómo interpretar sus bytes y proporcione métodos con los que el desarrollador pueda manipularlos. En esta sección sólo usaremos archivos binarios a través del módulo `pickle`, aunque podríamos usar el modo 'b' al abrirlos no es fácil manipular un archivo de este tipo.

Para leer o guardar archivos de texto basta con usar el modo 't' al abrirlo o simplemente no usar el modo 'b'. Si queremos trabajar con formatos específicos de archivo es mejor usar algún módulo especializado en ese tipo, como `csv` para .csv, `configparser` para .cfg o .ini, o `json` para .json, ...

La función `open()` es la encargada de abrir el archivo que le indiquemos con las opciones que necesitemos. Si sólo indicamos el nombre del archivo se abrirá de tipo texto en modo lectura. El método `read()` del objeto file que devuelve al abrirlo, devuelve un string con todo su contenido. Una vez ya no es necesario hemos de cerrarlo con su método `close()`. Si no puede abrir el archivo, bien porque no existe o bien porque no tenemos suficientes permisos, la función `open()` genera la excepción `FileNotFoundError` o `PermissionError`. Conviene, pues, usar `try/except/finally` cuando se trabaja con archivos.

```
f = open('ruta/archivo')
print(f.read())
f.close()
```

El objeto file devuelto por la función open() tiene métodos con los que podemos manipular el contenido del archivo y guardarlo físicamente en disco si lo hemos abierto con ese propósito y tenemos los permisos necesarios.

Los métodos file.seek(posición [, origen]) y file.tell() sitúa el objeto en la posición indicada o devuelve su posición.

Si se ha abierto el archivo en modo 'r' (por defecto) o '+' (lectura y escritura), podemos usar varios métodos para leer su contenido: next(), read([bytes]), readline([bytes]) y readlines([bytes]).

Al abrir un archivo en modo 'w' se eliminia su contenido y se prepara para que escribamos algo, hemos de estar seguros de que es lo que queremos hacer pues no hay modo de recuperar su antiguo contenido. Los modos 'x', 'a' y '+' permiten abrirlo para escritura sin perder su contenido inicial. En todos estos casos podemos usar diferentes métodos para escribir nuevo contenido en el archivo: flush(), write(str) y writelines(secuencia).

```
with open('ruta/archivo') as f:
    print(f.read())
```

La declaración with llama al método close() al terminar, por lo que no tenemos que hacerlo explícitamente.

No profundizo más en la manipulación directa de archivos porque Python tiene clases especializadas en la gestión de un amplio tipo de formatos de archivo, que serán los que usemos a lo largo de esta asignatura.