



SEAS
CAMPUS SEAS

7

UNIDAD
DIDACTICA

Python

7. Ficheros

ÍNDICE

| | |
|--|-----|
| OBJETIVOS | 269 |
| INTRODUCCIÓN | 270 |
| 7.1. El sistema de directorios | 271 |
| 7.2. Creación de un fichero de texto usando PyCharm | 273 |
| 7.3. Escritura y lectura de datos en ficheros de texto | 275 |
| 7.3.1. Escritura de ficheros de texto | 275 |
| 7.3.2. Lectura de ficheros de texto | 278 |
| 7.3.3. Acceso directo en ficheros de texto | 285 |
| 7.4. Uso de pandas en ficheros de texto | 286 |
| 7.5. Ficheros CSV | 290 |
| 7.5.1. Lectura de ficheros csv | 291 |
| 7.5.2. Lectura de archivos csv con pandas | 295 |
| 7.6. Ficheros zip | 298 |
| 7.6.1. Escritura y lectura de ficheros zip | 299 |
| 7.7. Formato binario | 301 |
| 7.7.1. Escritura de un fichero binario | 301 |
| 7.7.2. Lectura de un fichero binario | 302 |
| 7.7.3. Ejemplo de lectura | 302 |
| RESUMEN | 305 |

OBJETIVOS

- Conocer algunos de los formatos de ficheros que existen.
- Aprender a utilizar las herramientas de Python para trabajar con los ficheros.
- Escribir datos en ficheros de texto.
- Leer datos almacenados en ficheros de texto.
- Escribir y leer datos en ficheros csv.
- Escribir y leer datos en ficheros de Excel.
- Escribir y leer ficheros binarios.



Cuando termina la ejecución de un programa, los datos almacenados en sus variables desaparecen de la memoria central del ordenador. Sin embargo, la mayor parte de las veces se deben guardar para volver a usarlos posteriormente. Una forma de conservar los datos consiste en almacenarlos en una unidad de memoria permanente tal como el disco duro del ordenador, un *pen-drive*, etc. La forma en la que se guarda la información es en forma de ficheros (también llamados archivos). La gestión de los dispositivos de almacenamiento permanente es del sistema operativo por lo que el programador se ve libre de realizar tan engorrosa tarea.

Existen multitud de formatos para almacenar datos en los ficheros. El formato más simple es en forma de texto (sucesiones de caracteres) legible por los editores

de texto. Los formatos csv, text, HTML y XML forman parte de esta clase de formatos. En los ficheros de texto, la información se almacena en forma matricial usando un separador de columnas y un separador de líneas (filas). Otros formatos distintos de los de texto pueden ser el formato comprimido (por ejemplo **ZIP**), el formato binario en el que la información se maneja byte a byte, etc. Formatos más complejos permiten, por ejemplo, almacenar contenido multimedia tal como imágenes (por ejemplo: JPEG, GIF, etc), sonido (por ejemplo: MIDI, WAV, MP3), vídeo (por ejemplo: AVI, MOV, MPG etc).

En esta unidad vamos a ver los ficheros de texto, binarios y ZIP, y las herramientas que nos permitirán trabajar en Python con este tipo de ficheros.

7.1. El sistema de directorios

Los sistemas de directorios sirven para organizar la información en el disco duro del ordenador o en otras memorias no volátiles, es decir, los ficheros. No resulta adecuado que los ficheros estén todos mezclados ya que puede haber una gran cantidad y no sería realista su uso. Por esto, están organizados dentro de carpetas (subdirectorios) que a su vez tienen una estructura en forma de árbol a partir de una carpeta raíz.

Sabemos que el encargado de esta gestión es el sistema operativo, que gestiona directorios, subdirectorios y ficheros en forma de árbol. Se denominará “directorio actual” o “directorio de trabajo” a aquel que contiene los ficheros (*scripts* y datos) con los que trabajamos en ese momento. El recorrido desde el directorio raíz hasta un determinado fichero se denomina “camino” (*path*) y su representación depende del sistema operativo.

Si no se quiere guardar el fichero con el que trabajamos en el subdirectorio actual es necesario indicar al intérprete de Python el camino del directorio en el que se quiere guardar. Las sentencias que permiten al intérprete dialogar con el sistema operativo no son directamente accesibles en el lenguaje, sino que forman parte de un módulo adicional llamado **os**, que habrá que importar. Este módulo nos permite realizar muchas operaciones que son función del sistema operativo, tales como crear una carpeta, hacer un listado del contenido de una carpeta, etc. Para crear una carpeta se usa *makedirs* pasando como parámetro una cadena de caracteres con el nombre de la carpeta. En la figura 7.1. se muestra el código de creación de una carpeta.

```
import os
os.makedirs("MiCarpeta")
```

Figura 7.1. Creación de una carpeta.

Como se muestra en la figura 7.2. , se ha creado la carpeta “MiCarpeta”.

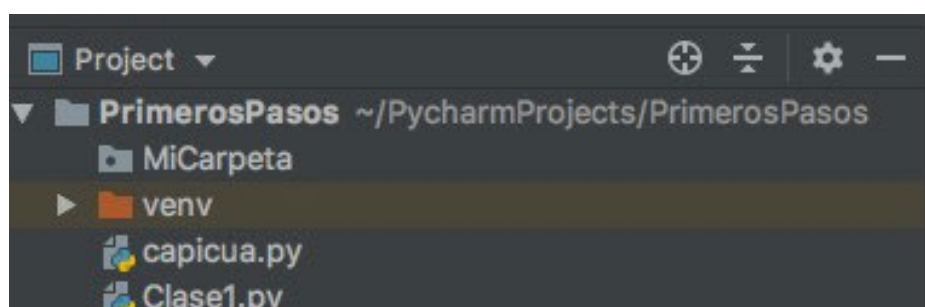


Figura 7.2. Carpeta creada.

La función **listdir** se usa para listar las carpetas y los ficheros existentes. Si se pasa como parámetro *./* devuelve los existentes en la carpeta actual, si se pasa una carpeta, los existentes en esa carpeta. En la figura 7.3. se muestra el código y el listado de carpetas y ficheros del directorio actual y de la carpeta que se acaba de crear, que está vacía.

```
import os
print(os.listdir("./"))
print()
print(os.listdir("MiCarpeta"))

['.idea', 'capicua.py', 'Comienzo.py', 'Datos.csv', 'datos1.csv', 'datos2.csv', 'datosPandas.py', 'ejemplo.xlsx', 'factorial.py', 'fich_matriz.py', 'ficheros.py', 'ficherosExcel.py', 'Gráficos.py', 'mat.txt', 'mat1.txt', 'MiCarpeta', 'MiFichero', 'modulos.py', 'Tenis', 'tenis.py', 'Triangulo', 'Triángulo.py', 'venv']

[]
```

Figura 7.3. Lista de archivos y carpetas.

Un comando interesante de ese módulo es la función `getcwd()`, que devuelve el directorio actual. Por ejemplo:

```
import os
ruta = os.getcwd()
print(ruta)
/Users/Antonio/PycharmProjects/PrimerosPasos
```

Figura 7.4. Uso del comando `getcwd`.

Como ya sabemos, si no se desea importar todo el módulo `os`, se puede importar únicamente esa función en la forma:

```
from os import getcwd
```

Figura 7.5. Importación de la función `getcwd`.

La función `chdir` permite cambiar el directorio actual indicando en el argumento mediante una cadena de caracteres el nuevo directorio deseado.

7.2. Creación de un fichero de texto usando PyCharm

Existen muchos programas para crear directamente ficheros de texto sin necesidad de programar. Vamos a generar un fichero de texto usando PyCharm, para practicar con el código que generemos para trabajar con los ficheros de texto. Vamos a considerar que este fichero es solo para practicar y lo vamos a situar fuera del contexto del proyecto en el que estamos trabajando.

Cuando se necesitan crear notas temporales o escribir código fuera del contexto del proyecto se recurre al uso de ficheros “scratch”. Equivalen a archivos borrador que no se almacenan en el directorio del proyecto pero que se pueden escribir y abrir desde otro proyecto.

Para ello, nos situamos en el menú principal sobre File y seleccionamos el sub-menú desplegable “New Scratch File” como se muestra en la figura 7.6.

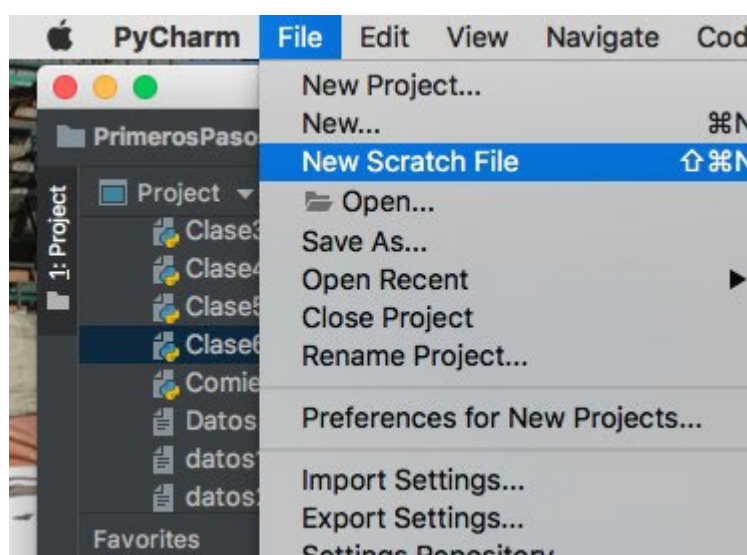


Figura 7.6. Selección del sub-menú New Scratch File.

Como se representa en la figura 7.7. (izquierda), aparece un menú desplegable del que se selecciona el comando “Plain Text” tras lo que nos aparece el editor de texto para poder escribir el contenido del fichero. Vamos a escribir, por ejemplo, un extracto de una poesía de Calderón De La Barca, como se muestra en la figura 7.7. derecha.

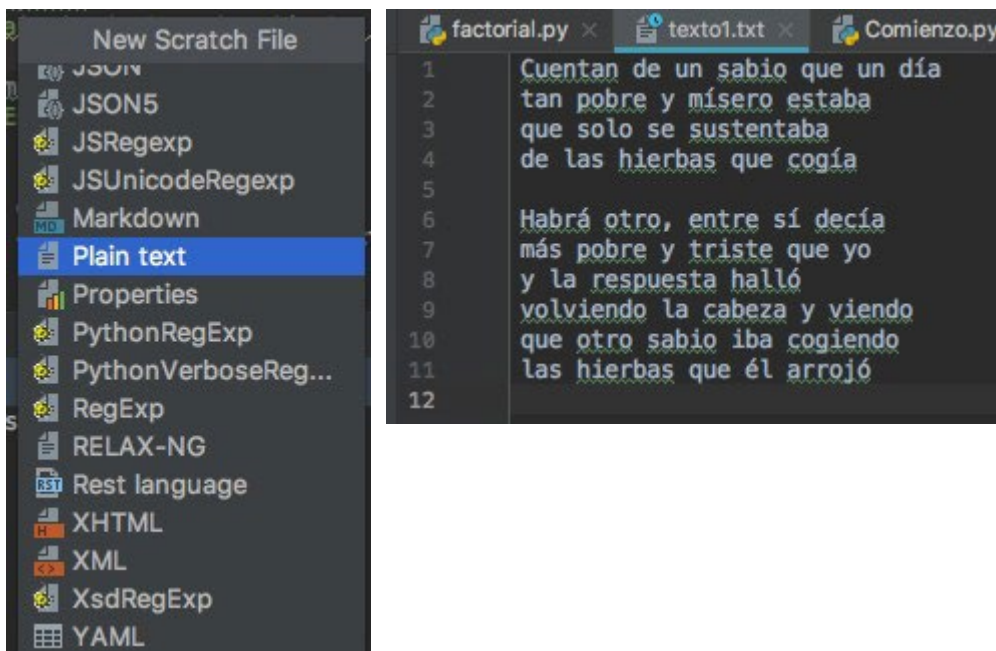


Figura 7.7. Izquierda: menú desplegable "New Scratch File". Derecha: Editor.

Tras escribir el texto, guardamos el contenido del editor y le damos por nombre "texto1.txt", como se muestra en la figura 7.8. La extensión txt aparece por defecto ya que se genera un fichero de extensión txt.

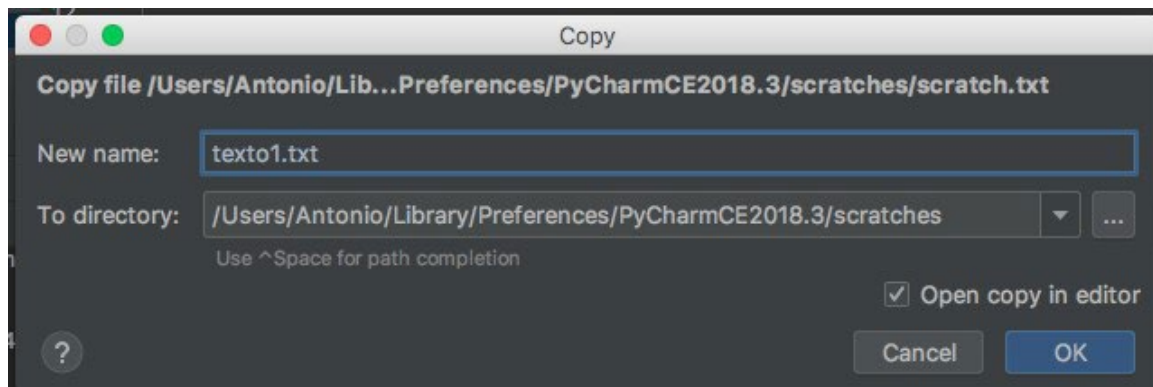


Figura 7.8. Salvando el fichero de texto con el nombre "texto1.txt".

Como se aprecia en la figura 7.9., el directorio en el que se ha creado el fichero de texto anterior es:

```
'/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches'
```

Figura 7.9. Ruta del fichero texto1.txt.

Y el reloj que aparece sobre el icono del fichero indica que se trata de un fichero temporal.

7.3. Escritura y lectura de datos en ficheros de texto

Como ya se ha comentado, el fichero de texto es el más sencillo y ampliamente utilizado en multitud de aplicaciones. En los apartados que siguen se va a ver la escritura de datos en los ficheros de texto, su lectura y su tratamiento posterior en algunas aplicaciones sencillas.

7.3.1. Escritura de ficheros de texto

Los ficheros de texto están compuestos por una secuencia de caracteres. En general, el esquema básico que se sigue para la escritura de los datos, es decir, para el almacenamiento de la información en un fichero es:

- En primer lugar, se crea el fichero si no existe o, si ya existe anteriormente, se abre para escritura.
- A continuación, se procede a la escritura de los datos en el fichero.
- Finalmente, cuando el fichero ya no se va a volver a usar, se cierra el fichero.

Puede extrañar que haya que cerrar el fichero. Para justificarlo vamos a describir brevemente el proceso de escritura. Los datos que se van a escribir en un fichero se almacenan previamente en una memoria llamada “tampón”. Cuando esta memoria se llena, los datos se vuelcan en el disco duro. Si al final de lo que queremos escribir la memoria no se ha llenado, hay que volcar esos datos al disco duro para que no se pierdan. Eso tiene lugar al cerrar el fichero. También permite volver a trabajar con el fichero.

La función que proporciona Python para la apertura de un fichero es:

```
open(fichero, modo)
```

Siendo **fichero** el nombre del fichero (escrito como cadena de caracteres) y **modo** es uno de los caracteres que se muestran en la tabla:

| Carácter | Efecto |
|----------|---|
| 'r' | Apertura para lectura. |
| 'w' | Crea el fichero para escritura. Si el fichero existe, borra el contenido del fichero existente. |
| 'r+' | Apertura para leer y escribir en el fichero. |
| 'a' | Añadir datos. Apertura para escritura tras el contenido actual. |
| 't' | Modo texto (por defecto). |

Figura 7.10. Modos de apertura de ficheros de texto.

Tras la apertura para escritura, si el fichero no existe se crea. Si ya existe y se ha abierto con el modo “w”, se limpia (se borra su contenido). El cierre del fichero se realiza con la función **close** y permite a otros programas leer lo que se ha escrito en él. Sin esta última etapa sería imposible acceder para leer o escribir de nuevo. El esquema de escritura en un fichero es:

```
f = open("nombre del fichero", "w")    # apertura
para escritura
f.write(s) # escritura de la cadena de caracteres s
...
f.close() # cierre del fichero
```

Por ejemplo:

```
fich = open('MiFichero', 'w')
fich.write('Buenos días, fichero1 \n')
fich.write('¿qué tal?')
fich.close()
```

Figura 7.11. Creación, escritura y cierre de un fichero.

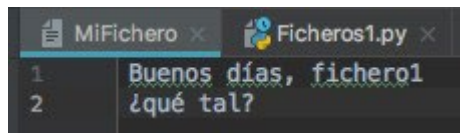


Figura 7.12. Resultado de ejecución del código anterior.

Las etapas de apertura y cierre están siempre presentes y sirven para indicar al sistema operativo que el programa desea acceder a un fichero e impedir que otro programa pueda acceder a él. De esta forma, si otro programa desea crear un fichero con el mismo nombre no podría hacerlo hasta que no se haya cerrado el fichero. Por el contrario, sí que podría leerlo ya que la lectura no interfiere con la escritura.

Cuando se termina un programa, si quedan ficheros abiertos se cierran automáticamente aunque no se haya ejecutado el método `close` pero, como se ha dicho antes, se puede perder la información que todavía no se haya transferido.

En la escritura de los ficheros de texto, se utilizan ciertos caracteres para organizar los datos. El símbolo “;” se usa para separar las columnas de una matriz. También se usa la tabulación o el cambio de línea. Como no son caracteres visibles, se codifican:

- `\n` cambio de línea.
- `\t` inserción de una tabulación. En Excel indica un cambio a la columna siguiente.

La función `open` acepta dos parámetros. El primero es el nombre del fichero y el segundo el modo de apertura, que recordemos que puede ser: “w” para escribir (*write*), “a” para escribir añadiendo al final (*append*) y “r” para leer (*read*).

La primera vez que se abre para escritura en un fichero, se crea un fichero de longitud nula. La escritura de un fichero no es inmediata ya que antes de escribir físicamente sobre el disco duro se necesita tener mucha información acerca de lo que se va a escribir. Como ya se ha dicho, para ello la información que se quiere escribir en el fichero se coloca previamente en una memoria tampón o “buffer interno” utilizado para acelerar el proceso evitando las llamadas al sistema para cada escritura.

■ Escritura en modo “añadir”.

Cuando se escriben datos en un fichero se pueden dar dos casos: en el primero, no se tiene en cuenta su contenido precedente y se borra su contenido previo. El segundo consiste en añadir nuevos datos a los precedentes tras la apertura del fichero. Únicamente cambia el modo de escritura, escribiendo en este segundo caso:

```
f = open ("nombre del fichero", "a") # apertura en modo añadir, modo "a"
```

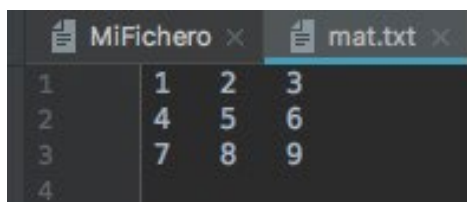
Para almacenar en un fichero de texto el contenido de una lista, tupla o diccionario se debe transformar a líneas de texto. Para ello, se usan separadores, por ejemplo comas, y a la línea de texto resultante se le añade un carácter de fin de línea ‘\n’. Un programa para conservar una matriz en un fichero puede ser como el mostrado en la figura 7.13. Se ve que las columnas están separadas por tabulaciones.

```
mat = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
f = open("mat.txt", 'w')
for i in range(0, len(mat)):
    for j in range(0, len(mat[i])):
        f.write(str(mat[i][j]) + "\t")
    f.write("\n")
f.close()
```

Figura 7.13. Escritura de una matriz en un fichero de texto.

En primer lugar se recorren las filas con el índice *i*. Con *i* igual a 0, se tiene la primera fila y se entra en la iteración interior. Con *j* se recorren las columnas, por tanto, se comienza por la primera columna, luego la segunda y finalmente la tercera. Una vez que se escribe cada columna se añade un salto de línea. Tras eso, *i* pasa a valer 1, con lo que se recorre la segunda fila, y así sucesivamente.

El resultado se muestra en la figura 7.11.b.



| | | | |
|---|---|---|---|
| 1 | 1 | 2 | 3 |
| 2 | 4 | 5 | 6 |
| 3 | 7 | 8 | 9 |
| 4 | | | |

Figura 7.14. Escritura de una matriz en un fichero de texto.

7.3.2. Lectura de ficheros de texto

La lectura de un fichero permite recuperar los datos previamente almacenados en él. Un esquema típico de lectura es:

1. Apertura del fichero en modo de lectura.
2. Lectura de los datos almacenados en el fichero.
3. Cierre del fichero.

Vamos a comenzar por un sencillo ejemplo. Escribamos el código de la figura 7.13, en el que la ruta del fichero puede cambiar según la carpeta en la que se trabaje. Se puede obtener la carpeta situándonos sobre la solapa del fichero y pulsando con el ratón derecho nos aparece un desplegable del que se selecciona “Copy Path” como se ve en la figura 7.15. Esta ruta se puede pegar tal como aparece en la figura 7.16 en color verde.

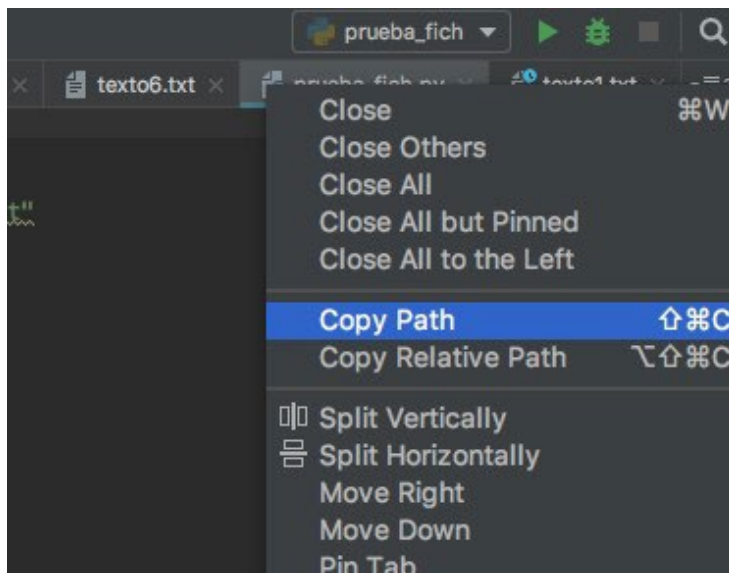


Figura 7.15. Obtención de la ruta del fichero.

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/texto1.txt"
fich = open(fichero, 'r')
print(fich)
<_io.TextIOWrapper name='/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/texto1.txt' mode='r' encoding='UTF-8'>
```

Figura 7.16. Apertura de un fichero de texto en modo de lectura.

La sentencia `open` abre el fichero `texto1.txt` en modo lectura. Con esto, el cursor para la lectura se sitúa sobre el primer carácter. A continuación, la siguiente línea de código hace que se escriba el contenido de la variable `fich`. Observar que todavía no se ha leído el fichero.

Dado que se tiene un cursor apuntando al primer carácter, se podría iterar mediante una sentencia `for` para leer todo el fichero. Si se escribe lo que se ha leído, dado que el fichero contiene caracteres de fin de línea, se escribirá línea a línea el contenido del fichero. Por ejemplo:

```
fichero = "/Users/Antonio/Library/
Preferences/PyCharmCE2018.3/
scratches/texto1.txt"
fich = open(fichero, 'r')
for lin in fich:
    print(lin)
fich.close()
Cuentan de un sabio que un día
tan pobre y mísero estaba,
que solo se sustentaba
de las hierbas que cogía.
¿Habrà otro, entre sí decía,
más pobre y triste que yo?
y cuando el rostro volvió
halló la respuesta, viendo
que otro sabio iba cogiendo
las hierbas que él arrojó.
```

Figura 7.17. Lectura de un fichero de texto.

Como se acaba de ver, a la hora de leer un fichero aparece una diferencia con respecto a la escritura, y es que la lectura se efectúa línea a línea (caracteres `\n`).

■ Método `readlines()`.

Para leer un fichero de texto se pueden utilizar sentencias de lectura. Comenzaremos con el método `readlines` (los métodos se estudian en la siguiente unidad) útil para la lectura de ficheros que no son demasiado grandes (menores de 100.000 líneas), que recupera de una vez todas las líneas de texto del fichero. Esa sentencia hace que se lea todo el fichero y que se devuelva en forma de una lista que contiene todas las líneas del fichero. A continuación, en la figura 7.18 se muestra un ejemplo.

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/text01.txt"
fich = open(fichero, 'r')
lin = fich.readlines()
print(lin)
fich.close()

['Cuentan de un sabio que un día\n', 'tan pobre y misero estaba,\n', 'que solo se sustentaba\n', 'de las hierbas que cogía.\n', '\n', '¿Habrá otro, entre sí decía,\n', 'más pobre y triste que yo?\n', 'y cuando el rostro volvió\n', 'halló la respuesta, viendo\n', 'que otro sabio iba cogiendo\n', 'las hierbas que él arrojó.\n']
```

Figura 7.18. Método `readlines`.

Hemos visto que cuando el programa precedente lee una línea en un fichero, el resultado leído incluye el o los caracteres `\n` `\r` que marcan el fin de línea. Por esta razón, a la lectura le suele seguir una etapa de limpieza que se realiza mediante la función **`strip`**, que devuelve una copia de la cadena de caracteres con los caracteres iniciales y finales eliminados. Su argumento es una cadena de caracteres con los caracteres que se quieren eliminar. En el ejemplo que sigue se eliminan los caracteres `\n` y `\r`. En la figura 7.19, se muestra el contenido del fichero previa y posteriormente a su eliminación.


```
f = open("textol.txt", 'r')
lin = f.readlines()
f.close()
limpieza = [s.strip("\n\r") for s in lin]
print(lin)
print()
print(limpieza)
['Cuentan de un sabio que un día\n', 'tan pobre y
miserio estaba,\n', 'que solo se sustentaba\n', 'de las
hierbas que cogía.\n', '\n', '¿Habr  otro, entre s 
dec a,\n', 'm s pobre y triste que yo?\n', 'y cuando
el rostro volvi \n', 'hall  la respuesta, viendo\n',
'que otro sabio iba cogiendo\n', 'las hierbas que  l
arroj .\n']
['Cuentan de un sabio que un d a', 'tan pobre y miserio
estaba,', 'que solo se sustentaba', 'de las hierbas que
cog a.', '', ' Habr  otro, entre s  dec a,', 'm s pobre
y triste que yo?', 'y cuando el rostro volvi ', 'hall 
la respuesta, viendo', 'que otro sabio iba cogiendo',
'las hierbas que  l arroj .']
```

Figura 7.19. Lectura con y sin limpieza de caracteres de control.

■ Uso de la sentencia *with*.

La sentencia *with* tambi n se usa en Python para abrir y cerrar ficheros de forma muy sencilla. Tiene la ventaja de que si al leer un fichero se produce un error, esta forma de trabajar garantiza el cierre del fichero, cosa que no sucede con los m todos precedentes.

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/textol.txt"
with open(fichero, 'r') as fich:
    lineas = fich.readlines()
    for lin in lineas:
        print(lin)
```

Cuentan de un sabio que un día
tan pobre y mísero estaba,
que solo se sustentaba
de las hierbas que cogía.
¿Habrás otro, entre sí decía,
más pobre y triste que yo?
y cuando el rostro volvió
halló la respuesta, viendo
que otro sabio iba cogiendo
las hierbas que él arrojó.

Figura 7.20. Apertura de un fichero con `with`.

■ Método `read()`.

Se puede usar la sentencia (método) `read` que lee todo el contenido de un fichero y devuelve una cadena de caracteres.

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/textol.txt"
with open(fichero, 'r') as fich:
    f = fich.read()
    print(f)
```

Cuentan de un sabio que un día
tan pobre y mísero estaba,
que solo se sustentaba
de las hierbas que cogía.

¿Habrás otro, entre sí decía,
más pobre y triste que yo?
y cuando el rostro volvió
halló la respuesta, viendo
que otro sabio iba cogiendo
las hierbas que él arrojó.

Figura 7.21. Lectura con la sentencia `read`.

■ Método `readline()`.

Otra sentencia es *readline* (sin la “s”). Esta sentencia lee el contenido de un fichero de texto y devuelve una cadena de caracteres correspondiente a cada línea. Esto hace que se pueda leer un fichero línea a línea. A continuación se da un ejemplo:

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/texto1.txt"
with open(fichero, 'r') as fich:
    lin = fich.readline()
    while lin != "":
        print(lin)
        lin = fich.readline()
```

```
Cuentan de un sabio que un día
tan pobre y mísero estaba,
que solo se sustentaba
de las hierbas que cogía.
¿Habrà otro, entre sí decía,
más pobre y triste que yo?
Y cuando el rostro volvió
halló la respuesta, viendo
que otro sabio iba cogiendo
las hierbas que él arrojó.
```

Figura 7.22. Uso de `readline` con `with`.

■ Lectura de las tres primeras líneas.

La lectura de las tres primeras líneas será:

```
from itertools import islice
with open(fichero) as fich:
    for lin in islice(fich, 3):
        print(lin)
```

```
Cuentan de un sabio que un día
tan pobre y mísero estaba,
que solo se sustentaba
```

Figura 7.23. Lectura de las tres primeras líneas.

■ Creación de un fichero a partir del contenido de otro fichero.

Veamos cómo se puede abrir un fichero como lectura y crear otro abriéndolo como escritura para escribir en este último un cierto contenido más el contenido del primero línea a línea. Por ejemplo, añadimos un asterisco antes de cada línea. En la figura 7.24 se muestra el código, y en la 7.25, el resultado.

```
fichero = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/texto1.txt"
fichero2 = "/Users/Antonio/Library/Preferences/PyCharmCE2018.3/scratches/texto2.txt"
with open(fichero, 'r') as fich1, open(fichero2, 'w') as fich2:
    for lin1 in fich1:
        fich2.write("* " + lin1)
```

Figura 7.24. Código para añadir contenido a un fichero.

```
* Cuentan de un sabio que un día
* tan pobre y mísero estaba,
* que solo se sustentaba
* de las hierbas que cogía.
*
* ¿Habrà otro, entre sí decía,
* más pobre y triste que yo?
* y cuando el rostro volvió
* halló la respuesta, viendo
* que otro sabio iba cogiendo
* las hierbas que él arrojó.
```

Figura 7.25. Contenido del nuevo fichero.

Ejemplo

Generación de una excepción en la apertura de un fichero inexistente.

Sabemos que si se intenta abrir un fichero que no existe el programa se interrumpe y nos genera un mensaje de error. Veamos la forma de generar una excepción que evite la interrupción del programa en ese caso. En el ejemplo mostrado en la figura 7.25, si el fichero no existiese se ofrecería la opción de crearlo o pedir de nuevo el nombre del fichero que se quiere abrir.

```
while not ok:
    nombre = input("Nombre del fichero: ")
    try:
        ok = True
        fich = open(nombre + '.txt', 'r')
    except FileNotFoundError:
        ok = False
        print('El fichero no existe')
        crearlo = input('Escriba 1 si desea crear el fichero: ')
        if crearlo == '1':
            ok = True
            fich = open(nombre + '.txt', 'w')
        else:
            ok = True
Nombre del fichero: texto10
El fichero no existe
Escriba 1 si desea crear el fichero: 3
Process finished with exit code 0
```

Figura 7.26. Código de generación de una excepción.

7.3.3. Acceso directo en ficheros de texto

Se puede acceder a cualquier carácter de un fichero de texto usando funciones que permiten acceso directo. Son:

- La función **tell()** devuelve la posición actual del cursor en un fichero.
- La función **seek()** que pone el cursor en la posición especificada.
- Para posicionar el cursor se pueden usar dos constantes del módulo **os**: **os.SEEK_CUR** y **os.SEEK_END**. Por ejemplo:
 - `fich.seek(3, os.SEEK_CUR)` avanza la posición del cursor 3 posiciones.
 - `fich.seek(-3, os.SEEK_END)` sitúa el cursor en la cuarta última posición.
 - `fich.seek(30)` pone el cursor en la posición 30 (la primera es cero) relativa al comienzo del fichero.

7.4. Uso de pandas en ficheros de texto.

Los datos se pueden organizar en un fichero de texto en una forma más elaborada, como por ejemplo, en los formatos HTML o XML. Para este formato más complejo se puede utilizar pandas. Vamos a ver métodos de lectura y escritura usando pandas. Como se muestra en el código de la figura 7.27, comenzaremos usando un *DataFrame*. Tras crear el *DataFrame* con los datos de un diccionario, la sentencia `df.to_excel` crea el fichero xlsx (formato de Excel) y almacena en él el contenido del *DataFrame*. En la figura 7.28 aparece ya creado el fichero `ejemplo.xlsx`. Abierto ese fichero con Excel, en la figura 7.29 se muestra su contenido.

```
import pandas as pd

datos = {'nombre': ['Antonio', 'Enrique',
'Fernando'], 'apellidos': ['Hernández',
'Tello', 'Sanz'], 'edad': [43, 62, 37],
'importe': [22, 33, 44]}

df = pd.DataFrame(datos, columns = ['nombre',
'apellidos', 'edad', 'importe'])
df.to_excel('ejemplo.xlsx', sheet_name =
'ejemplo')
```

Figura 7.27. Creación de ficheros de Excel.

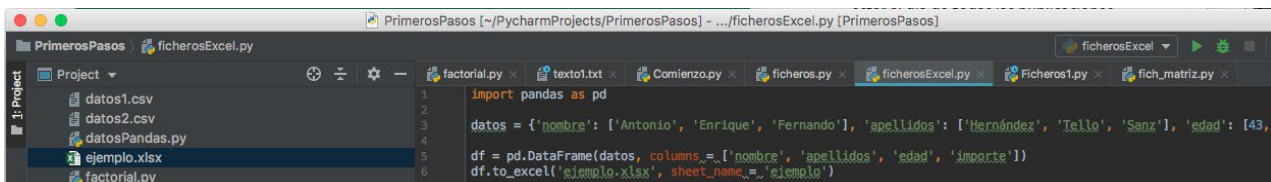


Figura 7.28. Fichero ejemplo.

Si abrimos el fichero utilizando Excel, su contenido se muestra en la figura 7.29.

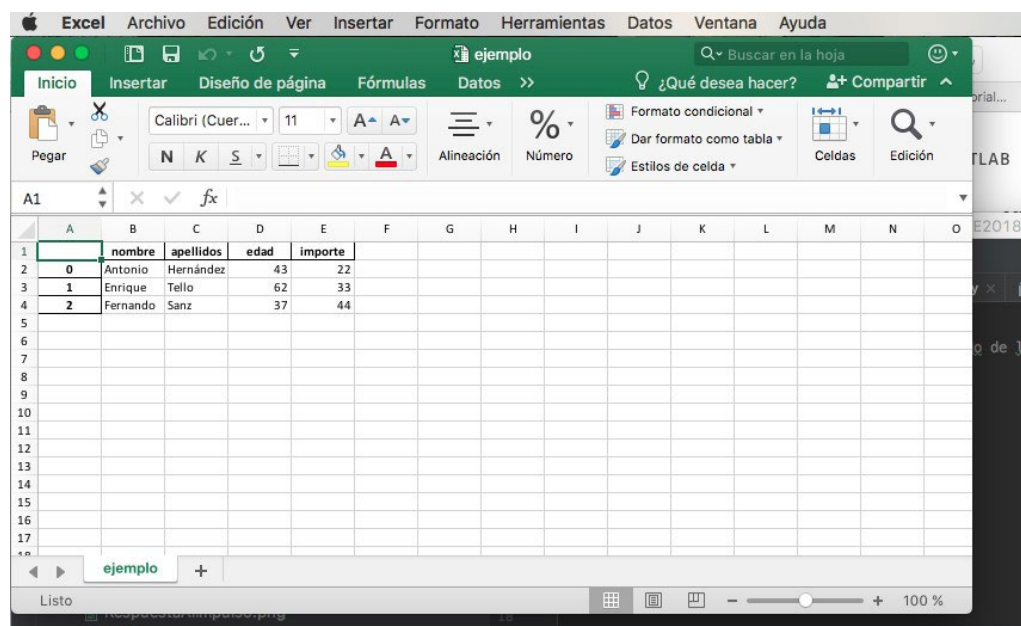


Figura 7.29. Contenido del fichero.

La lectura de archivos de Excel se realiza con el método `read_excel` de pandas. Si contiene más de una hoja se puede pasar el nombre de la hoja usando el método `sheet_name`. Por defecto, el método utiliza la primera línea del fichero como cabecera para asignar nombres a las columnas. Si no dispone de cabecera se debe asignar el valor `None` a `header`. Por ejemplo:

```
df = pd.read_excel('ejemplo.xlsx')
```

El resultado se ve en la figura 7.30.

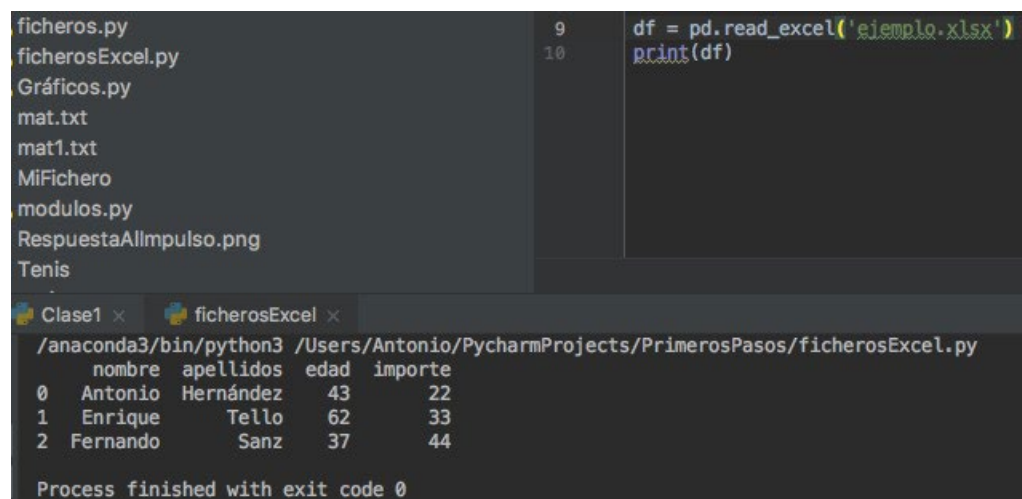
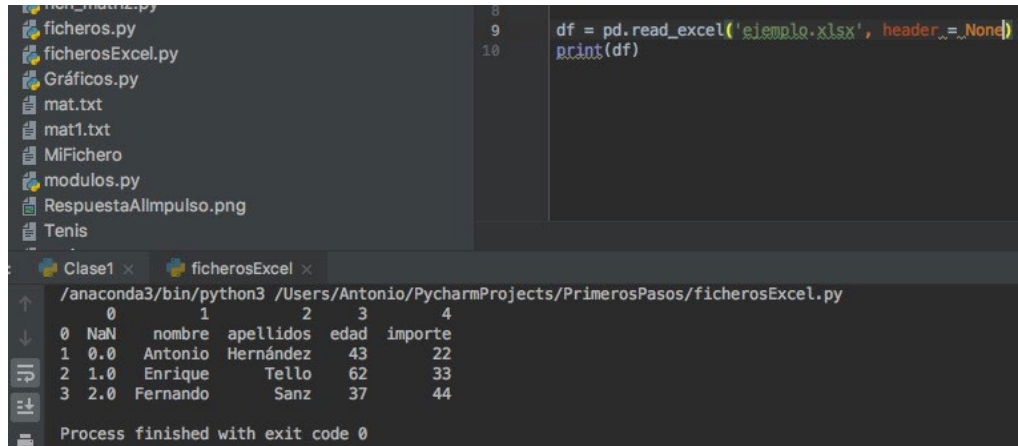


Figura 7.30. Lectura del fichero.

```
df = pd.read_excel('ejemplo.xlsx', header = None)
```

El resultado aparece en la figura 7.31.



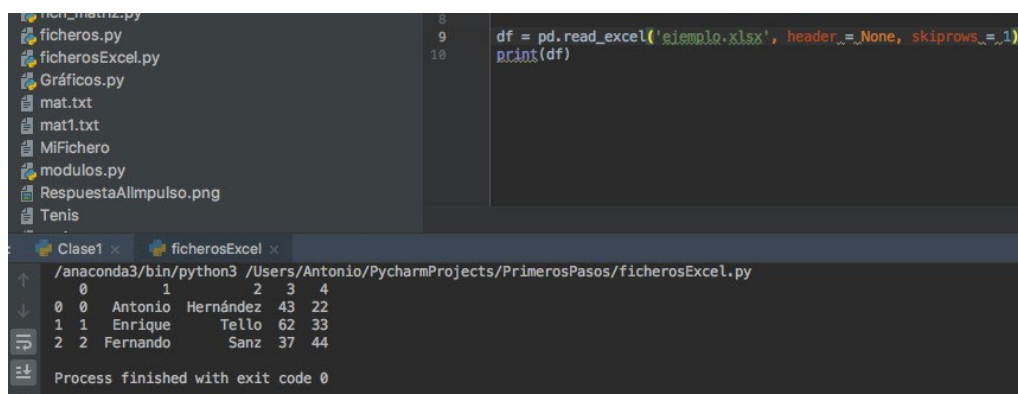
```
df = pd.read_excel('ejemplo.xlsx', header = None)
print(df)
```

| 0 | 1 | 2 | 3 | 4 |
|---|-----|----------|-----------|------|
| 0 | NaN | nombre | apellidos | edad |
| 1 | 0.0 | Antonio | Hernández | 43 |
| 2 | 1.0 | Enrique | Tello | 62 |
| 3 | 2.0 | Fernando | Sanz | 37 |

Figura 7.31. Lectura sin cabecera.

Se puede desechar de la lectura una o más filas utilizando el parámetro `skiprows`. Por ejemplo, para desechar la primera fila (figura 7.32):

```
df = pd.read_excel('ejemplo.xlsx', header = None, skiprows = 1)
```



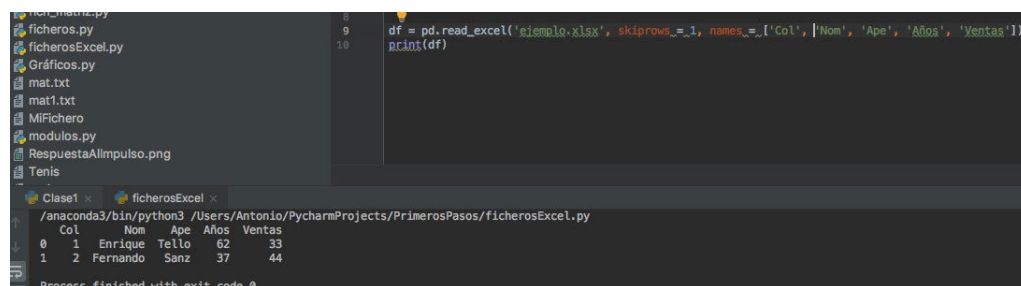
```
df = pd.read_excel('ejemplo.xlsx', header = None, skiprows = 1)
print(df)
```

| 0 | 1 | 2 | 3 | 4 |
|---|---|----------|-----------|----|
| 0 | 0 | Antonio | Hernández | 43 |
| 1 | 1 | Enrique | Tello | 62 |
| 2 | 2 | Fernando | Sanz | 37 |

Figura 7.32. Lectura sin cabecera.

También se puede asignar un nombre a las columnas distinto del que figura en la hoja mediante al parámetro `names` introducido en forma de lista. Por ejemplo (figura 7.30):

```
df = pd.read_excel('ejemplo.xlsx', skiprows = 1, names = ['Nom', 'Ape', 'Años', 'Ventas'])
```

The screenshot shows a PyCharm IDE with a project named 'PrimerosPasos'. The file explorer on the left lists files: ficheros.py, ficherosExcel.py, Gráficos.py, mat.txt, mat1.txt, MIFichero, modulos.py, RespuestaAlImpulso.png, and Tenis. The main editor shows a Python script with the following code:

```
8  
9 df = pd.read_excel('ejemplo.xlsx', skiprows=_1, names=['Col', 'Nom', 'Ape', 'Años', 'Ventas'])  
10 print(df)
```

The output console at the bottom shows the result of the script execution:

```
/anaconda3/bin/python3 /Users/Antonio/PycharmProjects/PrimerosPasos/ficherosExcel.py  
Col Nom Ape Años Ventas  
0 1 Enrique Tello 62 33  
1 2 Fernando Sanz 37 44  
Process finished with exit code 0
```

Figura 7.33. Lectura sin cabecera con supresión de una fila.

Existen otros paquetes que hacen que no sea a veces aconsejable escribir un código de lectura ya que existen librerías como el módulo `HTMLParser` para el formato HTML o el `xml.sax` para el XML que facilitan el trabajo. En este curso no se van a estudiar ya que salen fuera de su ámbito.

7.5. Ficheros CSV

Los ficheros CSV (*Comma Separated Values*) son ficheros de texto muy sencillos de procesar, aunque no son tan eficientes en tamaño y velocidad como los binarios. Son un caso particular de ficheros de texto que contienen una tabla de datos. Es el formato de fichero más utilizado en adquisición de datos, y se pueden leer escribiendo un programa o utilizando la Excel precisando que el formato es CSV.

Estos ficheros almacenan las tablas en un fichero de texto de forma que cada línea de texto del fichero corresponde a una fila de una tabla. Un delimitador tal como una coma, un punto y coma o una tabulación corresponde a la separación entre las columnas. Las porciones de texto separadas por comas corresponden a los contenidos almacenados en la tabla. La primera línea del fichero suele contener los títulos de las columnas. Por ejemplo, vamos a crear un fichero de texto que contenga la siguiente tabla:

```
Temperatura (°C),Presión (Bar)
21.2,1.02
22.11,2.3
23.1,2.12
22.2,1.89
21.2,1,27
21.6,2.02
```

Figura 7.34. Tabla que muestra el contenido de un fichero CSV.

Para ello, se puede utilizar el propio PyCharm. Seleccionamos *File* del menú principal.

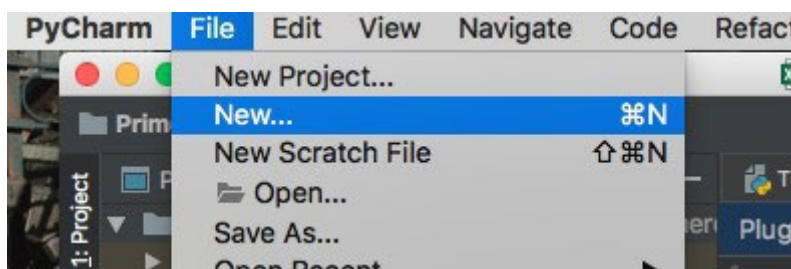


Figura 7.35. Selección de *File* → *New*.

Nos aparece un menú como el mostrado en la figura 7.32, del que se selecciona *New*. A continuación, como se muestra en la figura 7.33, se selecciona *Text* para crear un archivo de texto.

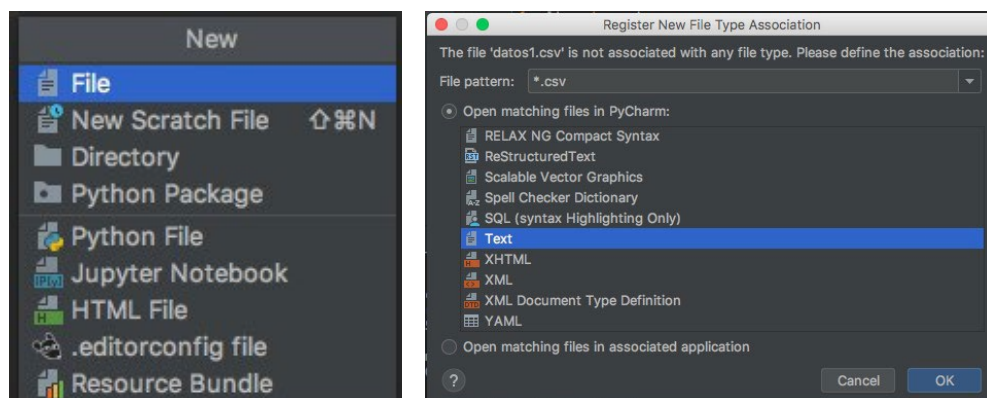


Figura 7.36. Selección de text.

Finalmente se escribe el nombre del archivo con extensión csv. Aunque en la figura pone datos2.csv, le vamos a llamar datos1.csv.

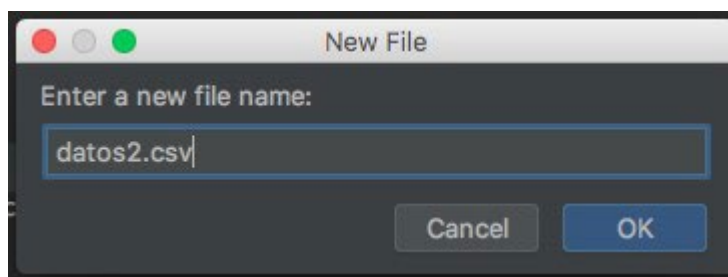


Figura 7.37. Introducción del nombre del archivo.

Tras eso, se escribe el fichero cuyo contenido se ha mostrado en la figura 7.34.

7.5.1. Lectura de ficheros csv

Para leer cualquier fichero, siempre hay que indicar a Python la ruta en la que se encuentra a menos que se haya ubicado en el directorio de trabajo, en cuyo caso no hay que indicarlo. Si hay que obtener la ruta, se puede usar el paquete **os** que permite gobernar el sistema operativo. El método `getcwd()` (`cwd`, Current Working Directory) devuelve la ruta. Por ejemplo:

```
import os
print(os.getcwd())
/Users/Antonio/PycharmProjects/PrimerosPasos
```

Figura 7.38. Obtención de la ruta de trabajo.

Se podría modificar la ruta de trabajo mediante: `os.chdir`.

Numerosos cálculos científicos y simulaciones se presentan en forma de tablas de datos almacenadas en ficheros CSV.

Para leer un fichero CSV con Python se utiliza el paquete CSV. Propone un objeto *reader* que permitirá decodificar un fichero csv. En la figura 7.39, se muestra el código de lectura del fichero `datos1.csv` que se acaba de crear y que encuentra ubicado en el directorio de trabajo.

```
import csv
fichero1 = open("datos1.csv", "r")
reader = csv.reader(fichero1, delimiter = ',')
for línea in reader:
    print(línea)
fichero1.close()
['Temperatura (°C)', 'Presión (Bar)']
['21.2', '1.02']
['22.11', '2.3']
['23.1', '2.12']
['22.2', '1.89']
['21.2', '1', '27']
['21.6', '2.02']
```

Figura 7.39. Lectura de un archivo csv.

Vemos que cada línea se ha escrito como una lista y que cada uno de los elementos de la lista son cadenas de caracteres. Si esas cadenas representan valores numéricos habría que convertirlas a *float*. El código de la figura 7.40 realiza esa operación salvo para la cabecera del fichero que está compuesta por dos cadenas de caracteres no numéricas.

```
import csv
ficherol = open("datos1.csv", "r")
reader = csv.reader(ficherol, delimiter = ',')
i = 0
for línea in reader:
    i += 1
    print(línea)
    if i!=1:
        print(float(línea[0]), float(línea[1]))
ficherol.close()
['Temperatura (°C)', 'Presión (Bar)']
['21.2', '1.02']
21.2 1.02
['22.11', '2.3']
22.11 2.3
['23.1', '2.12']
23.1 2.12
['22.2', '1.89']
22.2 1.89
['21.2', '1', '27']
21.2 1.0
['21.6', '2.02']
21.6 2.02
```

Figura 7.40. Conversión a float.

Para leerlo es necesario transformar cada línea en una lista de cadena de caracteres. Para esto, se puede usar el método *split* que divide una cadena de caracteres en una lista de sub-cadenas. El separador de las sub-cadenas se pasa como parámetro.

```
mat = [] # creación de una lista vacía
fich = open("texto1.txt", 'r') # apertura en modo
de lectura
for lin in fich: # recorrido de todas las líneas
del fichero
    s = lin.strip("\n\r") # limpia caracteres de
fin de línea
    l = s.split(";") # se corta en las columnas
    print('Escribo l:', l) # escribe la línea
    mat.append(l) # añade la línea a la lista
fich.close()
print('Escribo mat: ', mat)
Escribo l: ['Cuentan de un sabio que un día']
Escribo l: ['tan pobre y mísero estaba,']
Escribo l: ['que solo se sustentaba']
Escribo l: ['de las hierbas que cogía.']
Escribo l: ['']
Escribo l: ['¿Habrà otro, entre sí decía,']
Escribo l: ['más pobre y triste que yo?']
Escribo l: ['y cuando el rostro volvió']
Escribo l: ['halló la respuesta, viendo']
Escribo l: ['que otro sabio iba cogiendo']
Escribo l: ['las hierbas que él arrojó.']
Escribo mat: [['Cuentan de un sabio que un día'],
['tan pobre y mísero estaba,'], ['que solo se
sustentaba'], ['de las hierbas que cogía.'], [''],
['¿Habrà otro, entre sí decía,'], ['más pobre y
triste que yo?'], ['y cuando el rostro volvió'],
['halló la respuesta, viendo'], ['que otro sabio
iba cogiendo'], ['las hierbas que él arrojó.']]
```

Figura 7.41. Lectura de un fichero de texto.

Al final, *mat* es una lista de listas cuyos valores numéricos hay que transformarlos mediante *float* o *int*, según sea su tipo.

Este formato de fichero de texto se conoce como CSV. Cuando se usen valores numéricos no hay que olvidar en convertirlos en cadenas de caracteres para la escritura, y en valores numéricos en la lectura.

7.5.2. Lectura de archivos csv con pandas

Para la lectura y escritura de los archivos CSV con Pandas se utiliza `read_csv()` para cargar el contenido del fichero en un *DataFrame* y `df.to_csv()` para almacenar en un fichero el contenido de un *DataFrame* llamado `df`.

```
import os, pandas as pd
path = str(os.getcwd())
df1 = pd.read_csv('datos1.csv')
df1.head()
print(df1)
print()

fich = path + '/' + 'datos1.csv'
df2 = pd.read_csv(fich)
df2.head()
print(df2)
```

| | Temperatura(°C) | Presión(Bar) |
|---|-----------------|--------------|
| 0 | 21.20 | 1.02 |
| 1 | 22.15 | 2.32 |
| 2 | 23.11 | 2.12 |
| 3 | 22.23 | 1.89 |
| 4 | 21.29 | 1.27 |
| 5 | 21.66 | 2.02 |

Figura 7.42. Lectura y escritura con Pandas.

Ahora vamos a crear otro fichero al que llamaremos `datos2.csv` que tendrá un separador distinto. En este caso, como se muestra en la figura 7.43, se usará la coma para separar en los valores numéricos la parte entera de la decimal y el punto y coma para separar las columnas.

```
Temperatura(°C); Presión(Bar)
21,20;1,02
22,15;2,32
23,11;2,12
22,23;1,89
21,29;1,27
21,66;2,02
```

Figura 7.43. Uso del ";" como separador.

La lectura de este archivo se hará:

```
df3 = pd.read_csv('datos2.csv', sep =
';')
df3.head()
print(df3)
```

| | Temperatura(°C) | Presión(Bar) |
|---|-----------------|--------------|
| 0 | 21,20 | 1,02 |
| 1 | 22,15 | 2,32 |
| 2 | 23,11 | 2,12 |
| 3 | 22,23 | 1,89 |
| 4 | 21,29 | 1,27 |
| 5 | 21,66 | 2,02 |

Figura 7.44. Lectura del archivo.

Si el archivo es grande, es mejor ejecutar el código que sigue, que lee las 200 primeras filas.

```
df = pd.read_csv(ruta, nrows = 200)
df.shape
```

Si el archivo es muy grande, se puede leer por trozos:

```
meses = ['mayo', 'julio']
df = pd.DataFrame()
for trozo in pd.read_csv(ruta, sep =
';', chunksize = 1000):
    df = pd.concat([df,
trozo[trozo['mes'].isin(meses)]]
df.mes.value_counts()
```

Con *chunksize* se lee el archivo csv en trozos (chunks) de 1000 en 1000 y nos quedamos sólo con aquellos que cumplen un cierto requisito, en este caso, que el campo mes sea mayo o julio. Incluso en archivos extraordinariamente grandes se podría leer un csv extrayendo una muestra aleatoria.

```
df = pd.DataFrame()
for trozo in pd.read_csv(path + 'bank-additional-
full.csv', sep = ';', chunksize = 1000):
    df = pd.concat([df, trozo.sample(frac =
0.25)])
df2.shape
```


Se pueden leer los datos y obtener un diccionario por línea utilizando DictReader.

```
import csv
fich = open("fichero4.csv", "rt")
lectorCSV = csv.DictReader(fich,
delimter = ";")
for lin in lectorCSV:
    print(lin)
fich.close()
```

7.6. Ficheros zip

Los ficheros ZIP constituyen un estándar de compresión muy utilizado independientemente del sistema operativo utilizado. El lenguaje Python tiene funciones para comprimir y descomprimir esos ficheros en el módulo **zipfile**. Este formato permite reagrupar varios ficheros en uno solo.

ZIP es un formato que permite comprimir archivos de forma separada, permitiendo recuperar cada uno de los archivos comprimidos en una carpeta sin tener necesidad de leer el resto de los archivos. Un archivo zip contiene uno o varios archivos comprimidos.

Para crear un archivo nuevo, se abre el archivo zip en modo escritura:

```
import zipfile

fichero = "ficheroZip.zip"
arch = zipfile.Zipfile(fichero, mode = "w")
```

Ahora, para añadir archivos a este archivo se usa el método `write()`:

```
arch.write('archivo.txt', 'nombre_en_archivo.txt')
arch.close()
```

Si se desea escribir una cadena de *bytes* en el archivo, se puede usar el método `writestr()`:

```
str_bytes = "buffer cadena de caracteres"
arch.writestr('nombre_archivo.txt', str_bytes)
arch.close()
```

Hay varias formas de examinar el contenido de un archivo zip. Se puede utilizar `printdir`:

```
with zipfile.Zipfile(nombre) as zip:
    zip.printdir()
```

Para extraer todo el contenido de un archivo zip:

```
import zipfile
with zipfile.Zipfile('archivo.zip', 'r') as zfile:
    zfile.extractall('path')
```

El ejemplo siguiente permite obtener la lista de ficheros existentes en un fichero zip llamado `fichero_zip`.

```
import zipfile
fich = zipfile.ZipFile ("fichero_zip.zip", "r")
for info in fich.infolist () :
    print (info.filename, info.date_time,
info.file_size)
fich.close ()
```

Los ficheros comprimidos no son necesariamente ficheros de texto sino que pueden tener cualquier formato. El siguiente programa extrae un fichero de entre los comprimidos y después escribe su contenido (se supone que es un fichero de texto).

```
import zipfile
fich = zipfile.ZipFile ("ejemplozip.zip", "r")
data = fich.read ("informática/testzip.py")
fich.close ()
print (data)
```

En este caso, se encuentran las etapas de apertura y cierre incluso si la primera está incluida implícitamente en el constructor de la clase ZipFile.

7.6.1. Escritura y lectura de ficheros zip

El procedimiento para crear un fichero zip es similar al de cualquier otro fichero. La única diferencia proviene de que es posible almacenar el fichero a comprimir bajo otro nombre en el interior del fichero zip, lo que explica los dos primeros argumentos del método **write**. El tercer parámetro indica si el fichero se debe comprimir (zipfile.ZIP_DEFLATED) o no (zipfile.ZIP_STORED).

```
import zipfile

fich = zipfile.ZipFile("texto_zip1", 'w')    #
crea un fichero zip
fich.write("texto1.txt", "texto_zip1.txt",
zipfile.ZIP_DEFLATED)
fich.close()
```

Figura 7.45. Código para convertir un fichero de texto en un fichero ZIP.

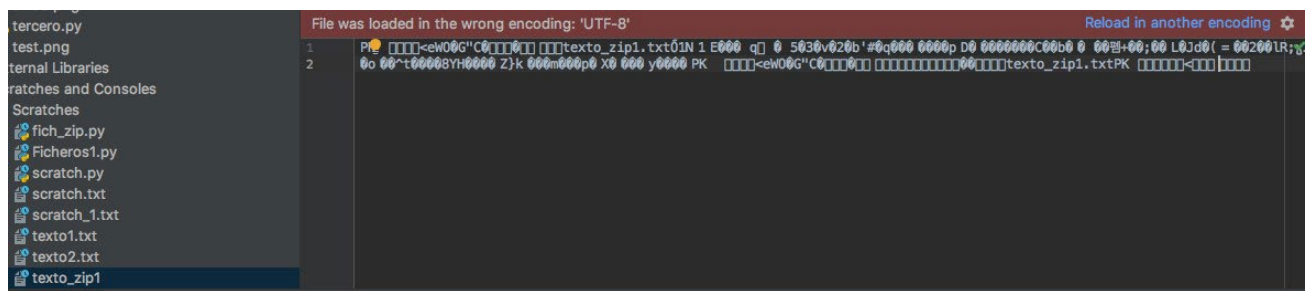


Figura 7.46. Columna izquierda: fichero texto_zip1. Columna derecha: su contenido abriendo el fichero con un editor de texto.

Un ejemplo sería el envío automático de un email al que se ha adjuntado un fichero zip.

7.7. Formato binario

Escribir y leer datos de un fichero de texto lleva a convertirlos en un formato legible por el usuario cualquiera que sea su tipo. Un entero se escribe en forma de caracteres decimales, aunque su representación en memoria sea en forma de unos y ceros (binaria). Esta conversión en un sentido y después en sentido contrario lleva un tiempo de tratamiento y afecta al tamaño de los ficheros incluso si permite releer los datos escritos con no importa qué editor de texto.

Para una gran masa de datos puede resultar más interesante utilizar formato binario, es decir, el formato en que se almacenan los datos en memoria. En este caso:

- En la escritura:
 - Los datos se ponen como una sucesión de bytes (función **pack** del módulo **struct**, que permite empaquetar datos como una sucesión de bytes).
 - Se escriben en el fichero (método *write*).
- En la lectura:
 - Se lee una sucesión de bytes del fichero (método *read*).
 - Esta sucesión de octetos se transforma para recuperar la información en su formato original (función **unpack** del módulo **struct**).

La utilización de ficheros binarios es menos evidente de lo que pueda parecer y hay que hacer llamadas a módulos especializados, mientras que los de texto no tienen ningún problema. El interés de estos ficheros reside en el hecho de que los datos que contienen ocupan menos memoria cuando se almacenan en binario que en formato texto como cadenas de caracteres.

La escritura y la lectura de un fichero binario conlleva los mismos problemas que los de un fichero de texto: hay que organizar los datos antes de registrarlos para saber cómo recuperarlos. Los tipos inmutables (entero, real, carácter) son bastante simples para guardar en este formato. Para objetos complejos, Python dispone del módulo **pickle**.

7.7.1. Escritura de un fichero binario

Al igual que con el resto de ficheros, la escritura de un fichero binario comienza por la apertura del fichero en modo escritura por medio de la instrucción:

```
fich = open("<nombre_fichero>", "wb")
```

El modo "wb" se refiere a escritura (modo de apertura "w") y a formato binario (modo b). El cierre es lo mismo que en el caso de un fichero de texto.

| | |
|-----|---|
| 'b' | Modo binario, para abrir ficheros binarios. |
|-----|---|

El módulo **struct** y la función **pack** permiten convertir las informaciones en forma de cadena de caracteres antes de registrarlas en formato binario. La función *pack* construye una cadena de caracteres igual al contenido de la memoria. Su escritura con la función **print** normalmente produce cosas que no son legibles. La tabla muestra los principales formatos de conversión.

| | |
|----------|-------------------------------|
| c | carácter. |
| B | carácter sin signo (octeto). |
| i | entero (4 octetos). |
| l | entero sin signo (4 octetos). |
| d | doble (8 octetos). |

7.7.2. Lectura de un fichero binario

Se usa la función:

```
fich = open("<nombre_fichero>", "rb")
```

La lectura usa la función **unpack** para efectuar la conversión inversa, de una cadena de caracteres en enteros, reales, etc.

7.7.3. Ejemplo de lectura

En este ejemplo, se crea un fichero "fichero.bin" para escribir datos en él. No se podrá leer su contenido con un editor de texto. El método **pack** acepta un número variable de parámetros, lo que explica la sintaxis `struct.pack("cccc", *s)`.

```
import struct
n = 12
x = 2.223682
rlist = [3.1416, 6.28]
s1 = b'a'
s2 = b'ab'

print(format(n, "b"))    # binario 1100
print(format(n, "x"))    # hexadecimal c
print(format(x, "f"))    # float
print(format(x, "0.2f")) # float con dos decimales
print(format(x, "10.4f")) # float desplazado 4 espacios
                        con cuatro decimales
print(format(x, "e"))    # en notación mantisa y exponente

# creación y escritura
```

```

fich = open("prueba_bin.bin", "wb")
fich.write(struct.pack('n', n))
fich.write(struct.pack('f', x))
fich.write(struct.pack('=2d', *rlist))
fich.write(struct.pack('c', s1))
fich.write(struct.pack('2s', s2))
fich.close()

# lectura del fichero
fich = open("prueba_bin.bin", 'rb')
n = struct.unpack('n', fich.read(8))
x = struct.unpack('f', fich.read(4))
d = struct.unpack('=2d', fich.read(16))
c1 = struct.unpack('c', fich.read(1))
c2 = struct.unpack('2s', fich.read(2))
fich.close()

print('n: ', n)
print('x: ', x)
print('d: ', d)
print('c1: ', c1)
print('c2: ', c2)
1100
c
2.223682
2.22
      2.2237
2.223682e+00
n:  (12,)
x:  (2.223681926727295,)
d:  (3.1416, 6.28)
c1:  (b'a',)
c2:  (b'ab',)

```

Figura 7.47. Escritura y lectura de un fichero binario.

RESUMEN

- La forma en la que se guarda la información en memoria externa es en forma de ficheros (también llamados archivos).
- Los sistemas de directorios sirven para organizar los ficheros en el disco duro del ordenador o en otras memorias no volátiles. Sabemos que el encargado de esta gestión es el sistema operativo, que gestiona directorios, subdirectorios y ficheros en forma de árbol.
- El recorrido desde el directorio raíz hasta un determinado fichero se denomina “camino” (path) y su representación depende del sistema operativo.
 - Las sentencias que permiten al intérprete dialogar con el sistema operativo forman parte de un módulo llamado **os**.
 - Para crear una carpeta se usa **makedirs** pasando como parámetro una cadena de caracteres con el nombre de la carpeta.
 - La función **listdir** se usa para listar las carpetas y los ficheros existentes.
 - La función **getcwd()**, devuelve el directorio actual.
- Los ficheros “**scratch**” de PyCharm, equivalen a archivos borrador que no se almacenan en el directorio del proyecto pero que se pueden escribir y abrir desde otro proyecto.

FUNCIONES PARA GESTIÓN DE FICHEROS DE TEXTO

■ Apertura de un fichero. Se usa la función `open(fichero, modo)`. **fichero** es el nombre del fichero, **modo** indica el tipo de operación a realizar.

■ Modos:

- 'w': Crear fichero para escritura. Si el archivo existe previamente lo borra y crea uno vacío.
- 'a': Añadir datos a un fichero al final del mismo. Si el fichero no existe, crea un fichero vacío.
- 'r': Abre un fichero para lectura de datos.
- 'r+': Abre un fichero para lectura y escritura de datos.

■ Escritura de texto. Se usa la función `write(cad)` para escribir una línea de texto. La línea `cad` debe terminar con el carácter de fin de línea `'\n'`.

■ En la escritura, se utilizan ciertos caracteres para organizar los datos:

- `“,”` para separar las columnas de una matriz.
- `\n` cambio de línea.
- `\t`: inserción de una tabulación. En Excel indica un cambio a la columna siguiente.

■ **Lectura de una línea.** almacenada en un fichero de texto. Se usa la función:

```
lin = fich.readline(),
```

que lee una línea hasta que el cursor llega al carácter de fin de línea. Si no quedan líneas, devuelve una línea vacía.

■ **Lectura de todo el fichero.** Se usa la función `readlines`, que devuelve una lista con el contenido de todo el fichero:

```
lst = fich.readlines()
```

■ Dado que la función `readlines` lee todo el fichero y lo asigna a una variable, se debe utilizar para la lectura de ficheros de pequeño tamaño.

■ La función `readlines` devuelve una lista, mientras que `readline` devuelve una cadena de caracteres.

■ La función **`strip`**, devuelve una copia de la cadena de caracteres con los caracteres iniciales y finales eliminados.

■ La sentencia **`with`** también se usa para abrir y cerrar ficheros y garantiza el cierre del fichero.

- **Cierre de un fichero:** se usa la función `close`. Por ejemplo: `fich.close()`
- Obtener la posición actual del cursor de lectura: se usa la función `tell`, que devuelve la posición actual del cursor. La posición inicial es cero. Por ejemplo: `pos = fich.tell()`
- Ubicar el cabezal de lectura en una cierta posición `pos` (la primera es cero). Se usa el método `seek`, en la forma: `fich.seek(pos)`.
- Se puede indicar el número de líneas a leer empleando `islice(fichero, num_lin)`.
- Para formato más complejo se puede utilizar **pandas**.
- La sentencia `df.to_excel` crea el fichero `xlsx` (formato de Excel) y almacena en él el contenido de un DataFrame creado previamente con los datos de un diccionario.
- La lectura de archivos de Excel se realiza con el método `read_excel` de **pandas**. Si contiene más de una hoja se puede pasar el nombre de la hoja usando el método `sheet_name`. Si no dispone de cabecera se debe asignar el valor `None` a `header`. Se puede desechar de la lectura una o más filas utilizando el parámetro `skiprows`. También se puede asignar un nombre a las columnas distinto del que figura en la hoja mediante al parámetro `names`.

```
df = pd.read_excel('ejemplo.xlsx', skiprows = 1, names =  
                  ['nombre1', 'nombre2'])
```

- Los **ficheros csv** (Comma Separated Values) son un caso particular de ficheros de texto que contienen una tabla de datos. Un delimitador tal como una coma, un punto y coma o una tabulación corresponde a la separación entre las columnas.
- Para leerlo se empleará el paquete **csv** que propondrá un objeto **reader** que permitirá decodificar un fichero **csv**.
- Cada línea de la tabla que se incluya se escribe como una lista, donde cada uno de los elementos de la lista son cadenas de caracteres. Si esas cadenas representan valores numéricos habría que convertirlas a `float`.
- **split** divide una cadena de caracteres en una lista de sub-cadenas. El separador de las sub-cadenas se pasa como parámetro.
- Para la lectura y escritura de los archivos CSV con **pandas** se utiliza `read_csv()` para cargar el contenido del fichero en un DataFrame y `df.to_csv()` para almacenar en un fichero el contenido de un DataFrames llamado `df`.
- Con `chunksize` se lee el archivo csv en trozos (chunks) de 1000 en 1000 y nos quedamos sólo con aquellos que cumplen un cierto requisito. Se pueden leer los datos y obtener un diccionario por línea utilizando **DictReader**.

EJEMPLOS DE GESTIÓN DE FICHEROS

- Lectura de un archivo de texto en una lista. Supondremos que cada línea está compuesta por una cadena de caracteres, un entero y un real, en ese orden.

```
fich = open(fichero, 'r')
lista = []
for lin in fich:
    dato = lin.rstrip().split(',')
    info1 = dato[0]
    info2 = int(dato[1])
    info3 = float(dato[2])
    lst = [info1, info2, info3]
    lista = lista + [lst]
fich.close()
```

- Lectura de un archivo de texto en un diccionario. Supondremos que cada línea está compuesta por una clave entera, una cadena de caracteres, un entero y un real.

```
fich = open(fichero, 'r')
dicc. = {}
for lin in fich:
    dato = lin.rstrip().split(',')
    clave = int(dato[0])
    info1 = dato[1]
    info2 = int(dato[2])
    info3 = float(dato[3])
    valor = [info1, info2, info3]
    dicc[clave] = valor
fich.close()
```

- Escritura de una lista en un archivo de texto. Supondremos creada una lista en la que cada línea está compuesta por una cadena de caracteres, un entero y un real, en ese orden.

```
fich = open(fichero, 'w')
for lst in lista:
    info1 = lst[0]
    info2 = lst[1]
    info3 = lst[2]
    info = [info1, str(info2), str(info3)]
    lin = ','.join(info) + '\n'
    fich.write(lin)
fich.close()
```

- Escritura de un diccionario `dicc` en un archivo de texto. Supondremos creado un diccionario con un campo clave entero y un campo valor compuesto por una cadena de caracteres, un entero y un real, en ese orden.

```
fich = open(fichero, 'w')
for clave in dicc:
    info1 = dicc[clave][0]
    info2 = dicc[clave][1]
    info3 = dicc[clave][2]
    campo = [str(clave), info1, str(info2),
str(info3)]
    lin = ','.join(campo) + '\n'
    fich.write(lin)
fich.close()
```

- Creación de una matriz de Numpy leída de un fichero csv.

```
import numpy as np

fich = open(fichero, 'r')
mat = []
for lin in fich:
    fila = lin[:-1].split(';')
    mat = mat + [fila]
matriz = np.array(mat, int)
fich.close()
```

- Escribir una matriz de Numpy en un fichero Excel.

```
fich = open(fichero, 'w')
for fila in matriz:
    lin = ''
    for dato in fila:
        lin = lin + str(dato) + ';'
    lin = lin[:-1] + '\n'
    fich.write(lin)
fich.close()
```

FICHEROS ZIP

- Los ficheros zip constituyen un estándar de compresión muy utilizado. El lenguaje Python tiene funciones para comprimir y descomprimir esos ficheros en el módulo **zipfile**.
- El procedimiento para crear un fichero zip es similar al de cualquier otro fichero. La única diferencia proviene de que es posible almacenar el fichero a comprimir bajo otro nombre en el interior del fichero zip, lo que explica los dos primeros argumentos del método **write**. El tercer parámetro indica si el fichero se debe comprimir (**zipfile.ZIP_DEFLATED**) o no (**zipfile.ZIP_STORED**).

FICHEROS BINARIOS

- Para una gran masa de datos puede resultar más interesante utilizar formato binario, es decir, el formato en que se almacenan los datos en memoria. Los tipos inmutables (entero, real, carácter) son bastante simples para guardar en este formato. Para objetos complejos, Python dispone del módulo **pickle**.
- En la escritura:
 1. Los datos se leen como una sucesión de bytes (función **pack** del módulo **struct**).
 2. Se escriben en el fichero (método **write**).
- En la lectura:
 1. Se lee una sucesión de bytes del fichero (método **read**).
 2. Esta sucesión de octetos se transforma para recuperar la información en su formato original (función **unpack** del módulo **struct**).
- La escritura de un fichero binario comienza por la apertura del fichero en modo escritura por medio de la instrucción: `fich = open("<nombre_fichero>", "wb")`. El código "wb" se refiere a escritura (modo de apertura "w") y a formato binario (b). El cierre es lo mismo que en el caso de un fichero de texto.
- El módulo **struct** y la función **pack** permiten convertir las informaciones en forma de cadena de caracteres antes de registrarlas en formato binario. La función **pack** construye una cadena de caracteres igual al contenido de la memoria.
- En la lectura se usa la función: `fich = open("<nombre_fichero>", "rb")`. La lectura usa la función **unpack** para efectuar la conversión inversa, de una cadena de caracteres en enteros, reales, etc.