AMPLIAR CONOCIMIENTOS

- Sets in Python

5.5 Ficheros

Aunque los ficheros encajarían más en un apartado de «*entrada/salida*» ya que representan un **medio de almacenamiento persistente**, también podrían ser vistos como *estructuras de datos*, puesto que nos permiten guardar la información y asignarles un cierto formato.[1]

Un **fichero** es un *conjunto de bytes* almacenados en algún *dispositivo*. El *sistema de ficheros* es la estructura lógica que alberga los ficheros y está jerarquizado a través de *directorios* (o carpetas). Cada fichero se identifica unívocamente a través de una *ruta* que nos permite acceder a él.

[1] Foto original de portada por Maksym Kaharlytskyi en Unsplash.

5.5.1 Lectura de un fichero

Python ofrece la función `open()` para «abrir» un fichero. Esta apertura se puede realizar en 3 modos distintos:

- **Lectura** del contenido de un fichero existente.
- **Escritura** del contenido en un fichero nuevo.
- **Añadido** al contenido de un fichero existente.

Veamos un ejemplo para leer el contenido de un fichero en el que se encuentran las temperaturas máximas y mínimas de cada día de la última semana. El fichero está en la subcarpeta (*ruta relativa*) `files/temps.dat` y tiene el siguiente contenido:

```
29 23
31 23
34 26
33 23
29 22
28 22
28 22
```

Lo primero será abrir el fichero:

```
>>> f = open('files/temps.dat')
```

La función `open()` recibe como primer argumento la **ruta al fichero** que queremos manejar (como un «string») y devuelve el manejador del fichero, que en este caso lo estamos asignando a una variable llamada `f` pero le podríamos haber puesto cualquier otro nombre.

Nota: Es importante dominar los conceptos de **ruta relativa** y **ruta absoluta** para el trabajo con ficheros. Véase este artículo de DeNovatoANovato.

Hay que tener en cuenta que la ruta al fichero que abrimos (*en modo lectura*) **debe existir**, ya que de lo contrario obtendremos un error:

```
>>> f = open('foo.txt')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'foo.txt'
```

Una vez abierto el fichero ya podemos proceder a leer su contenido. Para ello Python nos ofrece la posibilidad de leer todo el fichero de una vez o bien leerlo línea a línea.

Lectura completa de un fichero

Siguiendo con nuestro ejemplo de temperaturas, veamos cómo leer todo el contenido del fichero de una sola vez. Para esta operación, Python nos provee, al menos, de dos funciones:

`read()` Devuelve todo el contenido del fichero como una cadena de texto (`str`):

```
>>> f = open('files/temps.dat')

>>> f.read()
'29 23\n31 23\n34 26\n33 23\n29 22\n28 22\n28 22\n'
```

`readlines()` Devuelve todo el contenido del fichero como una lista (`list`) donde cada elemento es una línea:

```
>>> f = open('files/temps.dat')

>>> f.readlines()
['29 23\n', '31 23\n', '34 26\n', '33 23\n', '29 22\n', '28 22\n', '28 22\n']
```

Importante: Nótese que, en ambos casos, los saltos de línea `\n` siguen apareciendo en los datos leídos, por lo que habría que «limpiar» estos caracteres. Para ello se recomienda utilizar *las funciones ya vistas de cadenas de texto*.

Lectura línea a línea

Hay situaciones en las que interesa leer el contenido del fichero línea a línea. Imaginemos un fichero de tamaño considerable (varios GB). Si intentamos leer completamente este fichero de sola una vez podríamos ocupar demasiada RAM y reducir el rendimiento de nuestra máquina.

Es por ello que Python nos ofrece varias aproximaciones a la lectura de ficheros línea a línea. La más usada es **iterar** sobre el propio *manejador* del fichero:

```
>>> f = open('files/temps.dat')

>>> for line in f:    # that easy!
...     print(line)
...
29 23

31 23

34 26
```

(continué en la próxima página)

(proviene de la página anterior)

```
33 23

29 22

28 22

28 22
```

Truco: Igual que pasaba anteriormente, la lectura línea por línea también incluye el **salto de línea** `\n` lo que provoca un «doble espacio» entre cada una de las salidas. Bastaría con aplicar `line.split()` para eliminarlo.

5.5.2 Escritura en un fichero

Para escribir texto en un fichero hay que abrir dicho fichero en **modo escritura**. Para ello utilizamos un *argumento adicional* en la función `open()` que indica esta operación:

```
>>> f = open('files/canary-iata.dat', 'w')
```

Nota: Si bien el fichero en sí mismo se crea al abrirlo en modo escritura, la **ruta** hasta ese fichero no. Eso quiere decir que debemos asegurarnos que **las carpetas hasta llegar a dicho fichero existen**. En otro caso obtenemos un error de tipo `FileNotFoundError`.

Ahora ya podemos hacer uso de la función `write()` para enviar contenido al fichero abierto.

Supongamos que queremos volcar el contenido de una lista en dicho fichero. En este caso partimos de los *códigos IATA* de aeropuertos de las Islas Canarias[2].

```
1 >>> canary_iata = ("GCFV", "GCHI", "GCLA", "GCLP", "GCGM", "GCRR", "GCTS", "GCXO")
2
3 >>> for code in canary_iata:
4 ...     f.write(code + '\n')
5 ...
6
7 >>> f.close()
```

Nótese:

[2] Fuente: Smart Drone

Línea 4 Escritura de cada código en el fichero. La función `write()` no incluye el salto de línea por defecto, así que lo añadimos de *manera explícita*.

Línea 7 Cierre del fichero con la función `close()`. Especialmente en el caso de la escritura de ficheros, se recomienda encarecidamente cerrar los ficheros para evitar pérdida de datos.

Advertencia: Siempre que se abre un fichero en **modo escritura** utilizando el argumento `'w'`, el fichero se inicializa, borrando cualquier contenido que pudiera tener.

5.5.3 Añadido a un fichero

La única diferencia entre añadir información a un fichero y *escribir información en un fichero* es el modo de apertura del fichero. En este caso utilizamos `'a'` por «append»:

```
>>> f = open('more-data.txt', 'a')
```

En este caso el fichero `more-data.txt` se abrirá en *modo añadir* con lo que las llamadas a la función `write()` hará que aparezcan nueva información al final del contenido ya existente en dicho fichero.

5.5.4 Usandos contextos

Python ofrece gestores de contexto como una solución para establecer reglas de entrada y salida a un determinado bloque de código.

En el caso que nos ocupa, usaremos la sentencia `with` y el contexto creado se ocupará de cerrar adecuadamente el fichero que hemos abierto, liberando así sus recursos:

```
1  >>> with open('files/temps.dat') as f:
2  ...     for line in f:
3  ...         max_temp, min_temp = line.strip().split()
4  ...         print(max_temp, min_temp)
5  ...
6  29 23
7  31 23
8  34 26
9  33 23
10 29 22
11 28 22
12 28 22
```

Línea 1 Apertura del fichero en *modo lectura* utilizando el gestor de contexto definido por la palabra reservada `with`.

Línea 2 Lectura del fichero línea a línea utilizando la iteración sobre el *manejador del fichero*.

Línea 3 Limpieza de saltos de línea con `strip()` encadenando la función `split()` para separar las dos temperaturas por el caracter *espacio*. Ver *limpiar una cadena* y *dividir una cadena*.

Línea 4 Imprimir por pantalla la temperatura mínima y la máxima.

Nota: Es una buena práctica usar `with` cuando se manejan ficheros. La ventaja es que el fichero se cierra adecuadamente en cualquier circunstancia, incluso si se produce cualquier **tipo de error**.

Hay que prestar atención a la hora de escribir valores numéricos en un fichero, ya que el método `write()` por defecto espera ver un «string» como argumento:

```
>>> lottery = [43, 21, 99, 18, 37, 99]

>>> with open('files/lottery.dat', 'w') as f:
...     for number in lottery:
...             f.write(number + '\n')
...
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Importante: Para evitar este tipo de **errores**, se debe convertir a `str` aquellos valores que queramos usar con la función `write()` para escribir información en un fichero de texto.

Ejercicio

Dado el fichero `temperatures.txt` con 12 filas (meses) y 31 columnas (temperaturas de cada día), se pide:

1. Leer el fichero de datos.
2. Calcular la temperatura media de cada mes.
3. Escribir un fichero de salida `avgtemps.txt` con 12 filas (*meses*) y la temperatura media de cada mes.

Guarda el fichero en la misma carpeta en la que vas a escribir tu código. Así evitarás problemas de rutas relativas/absolutas.

AMPLIAR CONOCIMIENTOS

- Reading and Writing Files in Python
- Python Context Managers and the «with» Statement