

MÓDULO 7

**Ficheros de texto, ficheros y objetos.
App con datos persistentes.
Ficheros CSV y ficheros JSON.**



INTRODUCCIÓN

Prácticamente todo lo que hemos hecho hasta aquí fue crear pequeños scripts que manejan información sobre la marcha, es decir, se definen variables asignando valores, o se ingresa por teclado para luego manipularla, sin embargo imaginemos que debemos trabajar con un gran caudal de datos, por ejemplo llevar un registro de asistencia a clases y notas de alumnos que rindieron exámenes para determinar su condición de “regular” o “libre”, para ello podríamos crear variables en el programa y asignar valores, pero tenemos al menos dos inconvenientes, simular la situación nos consumiría el uso de demasiadas variables y aun así hay situaciones que no podríamos reflejar, también podríamos pedir el ingreso estos valores por teclado pero luego de cerrar el programa los perderíamos.

Lo que necesitamos es que nuestra información sea persistente y nos permita seguir trabajando la próxima vez que abramos el programa. Existen dos maneras de trabajar con datos persistentes, organizados en **ficheros** o almacenados en una **base de datos**.

Este módulo trata sobre la organización de la información en ficheros, donde estudiaremos cómo hacer para que el programa en vez de gestionar datos definidos en el propio código, los obtenga de nuestros ficheros con operaciones de lectura, modificación y escritura.



FICHEROS

Un fichero es un conjunto de bits almacenados en un dispositivo de memoria persistente, normalmente un disco duro. Este conjunto de información se identifica con un nombre (el nombre del fichero) y la dirección de la carpeta o directorio que lo contiene. Todos, absolutamente todos los ficheros se localizan en un directorio determinado que se conoce como la ruta del fichero.

Se conocen también como archivos porque son equivalentes digitales a los archivos escritos por ejemplo en expedientes o libretas alojados en una oficina tradicional.

Otra cosa importante es que los ficheros se suelen identificar también con una extensión (tex, doc, csv, etc.). Una extensión es un código que se escribe después del nombre, con un punto y varios caracteres y que nos permite identificar varios ficheros de un mismo tipo. En realidad esto no deja de ser una formalidad, ya que a nuestros programas no les importa la extensión, sino cómo deben interpretar los datos que hay escritos dentro.

Las operaciones que nos permiten los ficheros son:

- Creación: Proceso por el cual creamos un fichero en el disco.
- Apertura: Proceso por el cual abrimos un fichero para comenzar a trabajar.
- Cierre: Proceso por el cual cerramos un fichero para dejar de trabajar con él.
- Extensión: Proceso por el cual añadimos información al fichero.

Es posible realizar varias operaciones a la vez, como creación y apertura en la misma instrucción. Sin embargo es necesario abrir un fichero para poder extenderlo o cerrarlo.

También debemos conocer el concepto de puntero que es la manera en cómo el ordenador accede y escribe en el fichero correctamente. Hay que imaginar el puntero como si fuera el dedo del ordenador mientras recorre el fichero, igual que nosotros seguimos con el dedo un texto mientras lo leemos y así sabemos por dónde vamos.

El puntero es muy importante para evitar sobrecribir información.

Creación y escritura

Veamos un ejemplo para crear y escribir dentro de un archivo de texto:

```
from io import open

texto = "Una línea con texto\nOtra línea con texto"

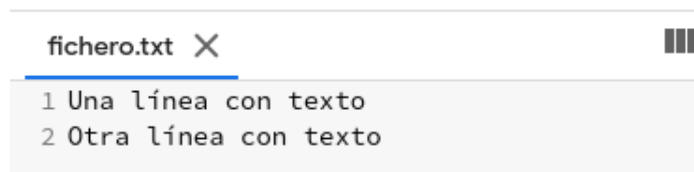
# Ruta donde se crea el fichero, w indica escritura (puntero al
principio)
fichero = open('fichero.txt', 'w')

# Escribimos el texto
fichero.write(texto)

# Cerramos el fichero
fichero.close()

# Este script no tiene salida en pantalla
```

Se genera el siguiente archivo, que puede ser descargado en la computadora personal:



Lectura

En este ejemplo leeremos el contenido de un archivo de texto y los mostraremos en pantalla:

```
from io import open

# Ruta donde se lee el fichero, r indica lectura (por defecto es r)
fichero = open('fichero.txt', 'r')

# Lectura completa
texto = fichero.read()

# Cerramos el fichero
fichero.close()

print(texto)
```

Una línea con texto
Otra línea con texto

salida

Podemos usar el método `readlines()` del fichero para generar una lista con las líneas:

```
from io import open
fichero = open('fichero.txt', 'r')

# Leemos creando una lista de líneas
texto = fichero.readlines()

fichero.close()
print(texto)
```

```
['Una línea con texto\n', 'Otra línea con texto']
```

salida

También se puede leer un fichero utilizando la instrucción estándar *with* de la siguiente forma:

```
with open("fichero.txt", "r") as fichero:
    for linea in fichero:
        print(linea)
```

```
Una línea con texto
```

```
Otra línea con texto
```

salida**Extensión (agregar datos)**

Este modo nos permite añadir datos al final de un fichero:

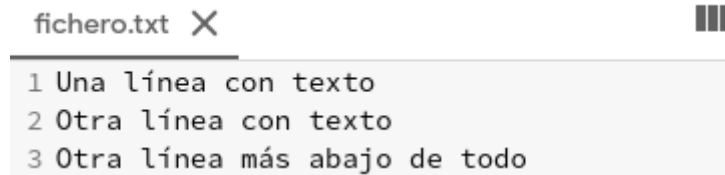
```
from io import open

# Ruta donde leeremos el fichero, a indica extensión (puntero al final)
fichero = open('fichero.txt', 'a')

fichero.write('\nOtra línea más abajo de todo')

fichero.close()
```

Observemos cómo quedó el archivo:



```
fichero.txt X
1 Una línea con texto
2 Otra línea con texto
3 Otra línea más abajo de todo
```

La variante `'a+'` permite crear el fichero si no existe, usaremos otro archivo que no fue creado antes:

```
from io import open

# Ruta donde leeremos el fichero, a indica extensión (puntero al final)
fichero = open('fichero_NUEVO.txt', 'a+')

fichero.write('coloco linea')

fichero.close()
```

El archivo_NUEVO.txt no existía, pero fue creado y además se agregó la línea deseada.



```
fichero_NUEVO.txt X
1 coloco linea
```

Es posible posicionar el puntero en el fichero manualmente usando el método **seek** e indicando un número de caracteres para luego leer una cantidad de caracteres con el método **read**:

```
fichero = open('fichero.txt', 'r')
fichero.seek(0)    # Puntero al principio
fichero.read(10)   # Leemos 10 caracteres
```

Una línea

salida

Introducción a la Programación en Python

Para posicionar el puntero justo al inicio de la segunda línea, podríamos ponerlo justo en la longitud de la primera:

```
fichero = open('fichero.txt', 'r')
fichero.seek(0)

# Leemos la 1ra línea y situamos el puntero al principio de la 2da
fichero.seek( len(fichero.readline()) )

# Leemos todo lo que queda del puntero hasta el final
fichero.read()
```

```
\nOtra línea con texto\nOtra línea más abajo de todo
```

salida

Se puede abrir un fichero en modo lectura con escritura, pero éste debe existir previamente. Además por defecto el puntero estará al principio y si escribimos algo sobre escribiremos el contenido actual.

El siguiente ejemplo muestra detalladamente cómo hacerlo, tengan en cuenta los saltos de línea y los caracteres especiales:

```
# Creamos un fichero de prueba con 4 líneas
fichero = open('fichero2.txt', 'w')
texto = "Línea 1\nLínea 2\nLínea 3\nLínea 4"
fichero.write(texto)
fichero.close()

# Lo abrimos en lectura con escritura y escribimos algo
fichero = open('fichero2.txt', 'r+')
fichero.write("0123456")

# Volvemos a poner el puntero al inicio y leemos hasta el final
fichero.seek(0)
fichero.read()
fichero.close()
```

Archivo generado:

fichero2.txt X



```
1 01234561
2 Línea 2
3 Línea 3
4 Línea 4
```

Modificar una línea

Para lograr este fin lo mejor es leer todas las líneas en una lista, modificar la línea en la lista, posicionar el puntero al principio y reescribir de nuevo todas las líneas:

```
fichero = open('fichero2.txt', 'r+')
texto = fichero.readlines()

# Modificamos la línea que queramos a partir del índice
texto[2] = "Esta es la línea 3 modificada\n"

# Volvemos a poner el puntero al inicio y reescribimos
fichero.seek(0)
fichero.writelines(texto)
fichero.close()

# Leemos el fichero de nuevo
with open("fichero2.txt", "r") as fichero:
    print(fichero.read())
```

```
01234561
Línea 2
Esta es la línea 3 modificada
Línea 4
```

salida

Luego de esta serie de ejemplos para manipular ficheros podemos hacer el siguiente resumen:

La función **open()** requiere dos argumentos de entrada, el **nombre del fichero** y el **modo de apertura del fichero**.

- 'r': Para leer el fichero.
- 'w': Para escribir en el fichero.
- 'a': Para añadir contenido a un fichero existente.
- 'a+': Para añadir contenido a un fichero existente o que aún no exista.

Introducción a la Programación en Python

Otra cosa que debemos hacer cuando trabajamos con ficheros en Python, es **cerrarlos una vez que ya hemos acabado con ellos**. Aunque es verdad que el fichero normalmente acabará siendo cerrado automáticamente, es importante especificarlo para evitar tener comportamientos inesperados.

Por lo tanto si queremos cerrar un fichero sólo tenemos que usar la función `close()` sobre el mismo. Por lo tanto tenemos tres pasos:

- ***Abrir el fichero que queramos indicando el modo de hacerlo***
- ***Usar el fichero para recopilar o procesar los datos que necesitábamos.***
- ***Cuando hayamos acabado, cerramos el fichero.***

FICHEROS CSV

El formato **CSV (Comma Separated Values)** es uno de los más comunes y sencillos para almacenar una serie de valores como si de una tabla se tratara.

Cada fila se representa por una línea diferente, mientras que los valores que forman una columna aparecen separados por un carácter concreto. El más común de los caracteres empleados para esta separación es la coma, de ahí el nombre del formato.

Sin embargo, es habitual encontrar otros caracteres como el signo del dólar (\$) o el punto y coma (;). Gracias al formato CSV es posible guardar una serie de datos, representados por una tabla, en un simple fichero de texto. Además, software para trabajar con hojas de cálculo, como, por ejemplo, Microsoft Excel o Calc LibreOffice, nos permiten importar y exportar datos en este formato.

Dentro de su librería estándar, Python incorpora un módulo específico para trabajar con ficheros CSV, que nos permite, tanto leer datos, como escribirlos.

Son dos los métodos básicos que nos posibilitan realizar estas operaciones: `reader()` y `writer()`. El primero de ellos sirve para leer los datos contenidos en un fichero CSV, mientras que el segundo nos ayudará a la escritura.

Escritura de listas en CSV

```
import csv

contactos = [
    ("Manuel", "Desarrollador Web", "manuel@ejemplo.com"),
    ("Javier", "Analista de datos", "javier@ejemplo.com"),
    ("Marta", "Experta en Python", "marta@ejemplo.com")
]

with open("contactos.csv", "w", newline="\n") as csvfile:
    writer = csv.writer(csvfile, delimiter=",")
    for contacto in contactos:
        writer.writerow(contacto)
```

Archivo generado:

Manuel	Desarrollador Web	manuel@ejemplo.com
Javier	Analista de datos	javier@ejemplo.com
Marta	Experta en Python	marta@ejemplo.com

Lectura de listas en CSV

```
with open("contactos.csv", newline="\n") as csvfile:
    reader = csv.reader(csvfile, delimiter=",")
    for nombre, empleo, email in reader:
        print(nombre, empleo, email)
```

Manuel Desarrollador Web manuel@ejemplo.com
Javier Analista de datos javier@ejemplo.com
Marta Experta en Python marta@ejemplo.com

salida

Escritura de diccionarios en CSV

```
import csv

contactos = [
    ("Manuel", "Desarrollador Web", "manuel@ejemplo.com"),
    ("Javier", "Analista de datos", "javier@ejemplo.com"),
    ("Marta", "Experta en Python", "marta@ejemplo.com")
]

with open("contactos.csv", "w", newline="\n") as csvfile:
    campos = ["nombre", "empleo", "email"]
    writer = csv.DictWriter(csvfile, fieldnames=campos)
    writer.writeheader()
    for nombre, empleo, email in contactos:
        writer.writerow({
            "nombre": nombre, "empleo": empleo, "email": email
        })
```

Archivo generado:

nombre	empleo	email
Manuel	Desarrollador Web	manuel@ejemplo.com
Javier	Analista de datos	javier@ejemplo.com
Marta	Experta en Python	marta@ejemplo.com



Lectura de diccionarios en CSV

```
with open("contactos.csv", newline="\n") as csvfile:
    reader = csv.DictReader(csvfile)
    for contacto in reader:
        print(contacto["nombre"], contacto["empleo"], contacto["email"])
```

Manuel Desarrollador Web manuel@ejemplo.com
Javier Analista de datos javier@ejemplo.com
Marta Experta en Python marta@ejemplo.com

salida



Ministerio de Educación
Cultura, Ciencia y Tecnología
Gobierno de Salta



Ministerio de Economía
y Servicios Públicos
Gobierno de Salta



**Universidad
Nacional de Salta**

FICHEROS JSON

El formato JSON se ha convertido en uno de los más populares para la serialización e intercambio de datos, sobre todo en aplicaciones web que utilizan AJAX. Recordemos que esta técnica permite intercambiar datos entre el navegador cliente y el servidor sin necesidad de recargar la página.

JSON son las siglas en inglés de **JavaScript Object Notation** y fue definido dentro de uno de los estándares (ECMA- 262) en los que está basado el lenguaje JavaScript. Es en este lenguaje donde, a través de una simple función (eval()), podemos crear un objeto directamente a través de una cadena de texto en formato JSON. Este factor ha contribuido significativamente a que sean muchos los servicios web que utilizan JSON para intercambiar datos a través de AJAX, ya que el análisis y procesamiento de datos es muy rápido. Incluso, son muchas las que han sustituido el XML por el JSON a la hora de intercambiar datos entre cliente y servidor.

Para estructurar la información que contiene el formato, se utilizan las llaves ({}) y se definen pares de nombres y valor separados entre ellos por dos puntos. De esta forma, un sencillo ejemplo en formato JSON, para almacenar la información de un empleado, sería el siguiente:

```
{"apellidos": "Fernández Rojas", "nombre": "José Luis", "departamento": "Finanzas",  
"ciudad": "Madrid"}
```

JSON puede trabajar con varios tipos de datos como valores; admite cadenas de caracteres, números, booleanos, listas y null. La mayoría de los lenguajes modernos de programación incluyen API y/o librerías que permiten trabajar con datos en formato JSON.

En Python disponemos de un módulo llamado json que forma parte de la librería estándar del lenguaje. Básicamente, este método cuenta con dos funciones principales, una para codificar y otra para decodificar. El objetivo de ambas funciones es traducir entre una cadena de texto con formato JSON y objetos de Python.

Obviamente, utilizando las funciones de ficheros de Python podemos leer y escribir ficheros que contengan datos en este formato. No obstante, debemos tener en cuenta que las funciones contenidas en json no permiten trabajar directamente con ficheros, sino con cadenas de texto.

Escritura de datos en JSON

```
import json

contactos = [
    ("Manuel", "Desarrollador Web", "manuel@ejemplo.com"),
    ("Lorena", "Gestora de proyectos", "lorena@ejemplo.com"),
    ("Javier", "Analista de datos", "javier@ejemplo.com"),
    ("Marta", "Experta en Python", "marta@ejemplo.com")
]

datos = []

for nombre, empleo, email in contactos:
    datos.append({"nombre": nombre, "empleo": empleo, "email": email})

with open("contactos.json", "w") as jsonfile:
    json.dump(datos, jsonfile)
```

Contenido del archivo JSON

```
[{"nombre": "Manuel", "empleo": "Desarrollador Web", "email": "manuel@ejemplo.com"}, {"nombre": "Marta", "empleo": "Experta en Python", "email": "marta@ejemplo.com"}]
```

Lectura de datos en JSON

```
with open("contactos.json") as jsonfile:
    datos = json.load(jsonfile)
    for contacto in datos:
        print(contacto["nombre"], contacto["empleo"], contacto["email"])
```

Manuel Desarrollador Web manuel@ejemplo.com
Marta Experta en Python marta@ejemplo.com

salida

Introducción a la Programación en Python

BIBLIOGRAFÍA

[1] Python para todos. R. Gonzalez Duque. Licencia Creative Commons Reconocimiento 2.5 España.

Descarga gratuita de versión más reciente en <http://mundogeek.net/tutorial-python/>

[2] Python 3 al descubierto 2da Edición. A. Fernández Montoro. Alfaomega. Año 2013

[3] Tutoriales Web varios.