Lección 16: Ficheros

Índice

Anterior

Ficheros

El manejo de archivos es una parte importante de la programación que nos permite crear, leer, actualizar y borrar archivos. En Python para manejar datos utilizamos la función incorporada *open()*.

```
# sintaxis
open('fichero', modo) # modos(r, a, w, x, t, b)  could be to read, write, update
```

- "r" - Lectura - Valor por defecto. Abre un archivo para leerlo, devuelve un error si el archivo no existe
- "a" - Append - Abre un fichero para añadir, crea el fichero si no existe.
- "w" - Write - Abre un archivo para escritura, crea el archivo si no existe.
- "x" - Create - Crea el fichero especificado, devuelve un error si el fichero existe
- "t" - Texto - Valor por defecto. Modo texto
- "b" - Binary - Modo binario (por ejemplo, imágenes)

Apertura de ficheros para lectura

El modo por defecto de *open* es lectura y texto, por lo que no es necesario especificar el modo *'r'* o *'rt'*.

```
f = open('./ejemplo.txt', , encoding='utf-8')
print(f) # No imprime el contenido del archivo, sino otra información
print(type(f))
f.close()
```

El archivo abierto tiene diferentes métodos de lectura: *read()*, *readline*, *readlines*. Un archivo abierto tiene que ser cerrado con el método *close()*.

- *read()*: lee todo el texto como cadena. Si queremos limitar el número de caracteres que queremos leer, podemos limitarlo pasando un valor int al método *read(number)*.

```
f = open('./ejemplo.txt', encoding='utf-8')
print(f) # No imprime el contenido del archivo, sino otra información
print(type(f))
texto = f.read()
print(type(texto))
print(texto)
f.close()
```

- *readlines()*: Lee todo el téxto línea a línea, y devuelve un diccionario de líneas.

```
f = open('./ejemplo.txt', encoding='utf-8')
print(f) # No imprime el contenido del archivo, sino otra información
print(type(f))
lineas = f.readlines()
print(type(lineas))
for i,linea in enumerate(lineas):
    print(f'Línea {i}: {linea}')
f.close()
```

Otra posibilidad es:

```
f = open('./ejemplo.txt', encoding='utf-8')
print(f) # No imprime el contenido del archivo, sino otra información
print(type(f))
for linea in f: # Recorrer directamente el archivo como un iterable
    print(linea)
f.close()
```

Después de abrir un archivo, debemos cerrarlo. Hay una gran tendencia a olvidarse de cerrarlos. Con *with* - cierra los archivos por sí mismo. El ejemplo anterior con *with* sería:

```
with open('./ejemplo.txt', encoding='utf-8') as f:
    print(f) # No imprime el contenido del archivo, sino otra información
    print(type(f))
    lineas = f.readlines()
    print(type(lineas))
    for i,linea in enumerate(lineas):
        print(f'Línea {i}: {linea}')
```

Apertura de ficheros para escritura

Para escribir en un archivo hay que indicar el modo *a* o *w* en la función *open()*:

- "a" - append - añadir al final de un archivo, si no exite lo crea.
- "w" - write - sobreescribe un archvo existente, si no existe lo crea.

Ejemplo:

```
with open('./ejemplo.txt','a', encoding='utf-8') as f:
    f.write('\nTexto añadido al final\n')
```

El siguiente ejemplo sobreescribirá el contenido del archivo si existe, si no existe lo creará:

```
with open('./ejemplo2.txt','w', encoding='utf-8') as f:
    f.write('Texto añadido en un archivo nuevo')
```

Borrando ficheros

El módulo *os* proporciona métodos para borrar archivos.

```
import os
os.remove('./ejemplo2.txt')
```

Si el archivo no existe, el módulo *os* dará un error, así que es buena idea comprobar primero si lo que vamos a borrar existe:

```
import os
if os.path.exists('./ejemplo2.txt'):
    os.remove('./ejemplo2.txt')
else:
    print('El fichero no existe')
```

Otra opción:

```
import os
try:
    os.remove('./ejemplo2.txt')
except FileNotFoundError as e:
    print(e)
```

Archivos JSON

JSON es el acrónimo de JavaScript Object Notation. Es el estándar para almacena objetos javascript o (listas de) diccionarios de python serializados (¿stringificados?). En definitiva, es un formato de texto sencillo para el intercambio de datos. En Python el módulo json proporciona herramientas para trabajar con este formato.

Las principales funciones/métodos para trabajar con el formato JSON son:

- **json.dump**(**obj**, **fp**, *, skipkeys=False, **ensure_ascii=True**, check_circular=True, allow_nan=True, cls=None, **indent=None**, separators=None, default=None, sort_keys=False, **kw) → Serializa un objeto (normalmente un diccionario) en un archivo *fp*.
- **json.dumps**(**obj**, *, skipkeys=False, **ensure_ascii=True**, check_circular=True, allow_nan=True, cls=None, **indent=None**, separators=None, default=None, sort_keys=False, **kw) → Serializa un objeto (normalmente un diccionario) lo devuelve en una cadena de texto.
- **json.load**(**fp**, *, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw) → Lee un json de un fichero JSON *fp* y devuelve un objeto (normalmente un diccionario o una lista de diccionarios).
- **json.loads**(**s**, *, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw) → Lee un json de una cadena de texto *s* y devuelve un objeto (normalmente un diccionario).

Advertencia: Tenga cuidado al parsear/cargar datos JSON de fuentes no fiables. Una cadena JSON maliciosa puede hacer que el descodificador consuma considerables recursos de CPU y memoria. Se recomienda limitar el tamaño de los datos que se van a analizar.

De objeto python a JSON (Escribir un archivo JSON)

```
import json

persona = {
   'nombre':'Manuel',
   'apellido':'Ejemplar',
   'edad':26,
   'país':'España',
   'casado':True,
   'conocimientos':['JavaScript', 'React', 'Node', 'MongoDB', 'Python'],
   'direccion':{
       'calle':'De la protección de datos',
       'cp':'28000'
   }
   }

persona_json = json.dumps(persona, indent = 4) # Convierte el diccionario en una cadena JSO
print(persona_json)

# Guardar el fichero JSON
with open('./persona.json', mode = 'w', encoding = 'utf-8') as f:
    f.write(persona_json)

# Otra forma más directa
with open('./persona.json', mode = 'w', encoding = 'utf-8') as f:
    json.dump(persona, f, ensure_ascii=False, indent=4)
```

De archivo JSON a objeto python

```
import json

with open('./persona.json', encoding='utf-8') as f:
    persona_json = f.read()

persona = json.loads(persona_json)
print(persona)
```

Otra forma de hacerlo:

```
import json
with open('./persona.json', encoding='utf-8') as f:
```

```
    persona = json.load(f)

print(persona)
```

Archivos CSV

CSV significa valores separados por comas. CSV es un formato de archivo sencillo utilizado para almacenar datos tabulares, como una hoja de cálculo o una base de datos. CSV es un formato de datos muy común en la ciencia de datos.

```
# Ejemplo de CSV:
nombre,apellido,edad,ciudad
Daniel,García,44,Ciudad Real
Ernesto,Sevilla,42,Albacete
...
```

Los archivos CSV pueden leerse como cualquier otro archivo de texto:

```
with open('./ejemplo.csv', encoding='utf-8') as f:
    print(f) # No imprime el contenido del archivo, sino otra información
    print(type(f))
    lineas = f.readlines()
    print(type(lineas))
    for i,linea in enumerate(lineas):
        print(f"Línea {i}: {linea.split(sep = ',')}")
print('Al salir del with se ha cerrado el archivo')
```

Pero además, la biblioteca estándar de python proporciona un módulo csv para leer y escribir archivos CSV más cómodamente.

Principales funciones/métodos:

- **csv.reader**(**csvfile**, dialect='excel', **fmtparams) → Retorna un objeto lector *csvreader* (como si fuera una lista de líneas) que iterará sobre las líneas del csvfile proporcionado.
- **csv.writer**(**csvfile**, dialect='excel', **fmtparams) → Retorna un objeto escritor *csvreader* responsable de convertir los datos del usuario en cadenas delimitadas en el objeto similar a un archivo dado.

Con el objeto escritor *csvwriter* se pueden utilizar: - **csvwriter.writerow(row)** → Escribe una línea en un CSV. *row* debe ser una lista de los elementos de una línea. Ej: ["alberto","garcia","29"] - **csvwriter.writerows(rows)** → Escribe varias líneas en un CSV. *rows* debe ser una lista de líneas. Ej: [["alberto","garcia","29"], ["pedro","moreno","65"]]

Lectura de un archivo CSV

```
import csv
```

```
with open('./ejemplo.csv', encoding='utf-8') as f:
    lineas = csv.reader(f)
    for i,linea in enumerate(lineas):
        if i == 0:
            print(f'Columnas: {", ".join(linea)}')
        elif i < 10:
            print(f'Datos {i}: {", ".join(linea)}')
```

Escritura de un archivo CSV

```
import csv

nombres_columnas = ["nombre", "apellido", "edad"]
linea1 = ["pedro", "moreno", "65"]
linea2 =["alberto", "garcia", "29"]
resto_de_lineas =[["nombre3", "apellido4", "edad4"],
                  ["nombre4", "apellido5", "edad5"],
                  ["nombre5", "apellido6", "edad6"]]

with open('./nuevocsv.csv', mode='w', encoding='utf-8') as f:
    csv_writer = csv.writer(f, delimiter=',')
    csv_writer.writerow(nombres_columnas)
    csv_writer.writerow(linea1)
    csv_writer.writerow(linea2)
    csv_writer.writerows(resto_de_lineas)
```

Ficheros XML

Python proporciona soporte a fichero XML en su biblioteca estándar con el paquete xml.

Siguiente