Programación

Tema 5. Manejo de Ficheros

Javier González Villa

David Lázaro Urrutia

Departamento de Matemática Aplicada
y Ciencias de la Computación

Este material se publica bajo la siguiente licencia:
Creative Commons BY-NC-SA 4.0

Contenidos

1. Ficheros
    1. Tipos de ficheros
    2. Lectura de ficheros
    3. Escritura de ficheros
2. Manejo avanzado de ficheros
    1. Contextos
    2. Ramificación e iteración
    3. Formateo
3. Ficheros estructurados y binarios

1. Ficheros

- Un fichero no es más que un conjunto de datos relacionados entre sí y que son almacenados de forma permanente en dispositivos físicos (discos duros, memorias, etc.).
- Los datos que se producen en el contexto de ejecución de código son borrados tras cerrar la aplicación con la que estemos trabajando ya que estos datos se almacenan en memoria.
- Los ficheros son una posible solución para almacenar resultados que queremos guardar para su posterior utilización si nos ha sido muy costoso, computacional o temporalmente, obtenerlos.
- Existen diversas maneras de estructurar los datos almacenados, pero debemos tener en cuenta que hay ciertos estándares que permiten compartir y reutilizar datos producidos por terceros.

1. Ficheros

- Rutas de acceso a ficheros:
  - Ruta absoluta: contiene toda la dirección donde se aloja el fichero.

    C:\Usuarios\NombreUsuario\Documentos\Proyecto\Datos\archivo.txt (Windows)
    /home/nombreusuario/documentos/proyecto/datos/archivo.txt (Linux)

  - Ruta relativa: contiene la dirección desde la ubicación actual.

    datos\archivo.txt (Windows)
    datos/archivo.txt (Linux)

- Tipos de ficheros:
  - Programas: contienen instrucciones.
  - Datos: contienen información sobre el problema concreto y pueden ser números, secuencias de caracteres u otros tipos.

1.1. Tipos de ficheros

- Ficheros binarios (‘b’):
  - No son editables.
  - Contienen una representación idéntica a los objetos almacenados en Python.
  - Pueden almacenar datos para su posterior recuperación.
  - No son fácilmente interpretables.
  - Permiten almacenar objetos creados dentro de Python y recuperarlos exactamente igual.
- Fichero de texto:
  - Los datos representados se guardan mediante caracteres alfanuméricos (ASCII).
  - Pueden ser leídos y modificados por un editor de texto.
  - Pueden ser estructurados de diversas formas.
  - No representan los objetos con los que se trabaja en Python de forma idéntica.

1.1. Tipos de ficheros

- Fichero de texto:
  - Los datos representados se guardan mediante caracteres alfanuméricos (ASCII).

Caracteres ASCII de control

00	NULL	(carácter nulo)
01	SOH	(inicio encabezado)
02	STX	(inicio texto)
03	ETX	(fin de texto)
04	EOT	(fin transmisión)
05	ENQ	(consulta)
06	ACK	(reconocimiento)
07	BEL	(timbre)
08	BS	(retroceso)
09	HT	(tab horizontal)
10	LF	(nueva línea)
11	VT	(tab vertical)
12	FF	(nueva página)
13	CR	(retorno de carro)
14	SO	(desplaza afuera)
15	SI	(desplaza adentro)
16	DLE	(esc.vínculo datos)
17	DC1	(control disp. 1)
18	DC2	(control disp. 2)
19	DC3	(control disp. 3)
20	DC4	(control disp. 4)
21	NAK	(conf. negativa)
22	SYN	(inactividad sínc)
23	ETB	(fin bloque trans)
24	CAN	(cancelar)
25	EM	(fin del medio)
26	SUB	(sustitución)
27	ESC	(escape)
28	FS	(sep. archivos)
29	GS	(sep. grupos)
30	RS	(sep. registros)
31	US	(sep. unidades)
127	DEL	(suprimir)

Caracteres ASCII imprimibles

32	espacio	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

ASCII extendido (Página de código 437)

128	Ç	160	á	192	└	224	Ó
129	ü	161	í	193	┴	225	ß
130	é	162	ó	194	┬	226	Ô
131	â	163	ú	195	├	227	Ò
132	ä	164	ñ	196	─	228	õ
133	à	165	Ñ	197	┼	229	Õ
134	å	166	ª	198	ã	230	µ
135	ç	167	º	199	Ã	231	þ
136	ê	168	¿	200	╚	232	Þ
137	ë	169	®	201	╔	233	Ú
138	è	170	¬	202	╩	234	Û
139	ï	171	½	203	╦	235	Ù
140	î	172	¼	204	╠	236	ý
141	ì	173	¡	205	═	237	Ý
142	Ä	174	«	206	╬	238	¯
143	Å	175	»	207	¤	239	´
144	É	176	░	208	ð	240	≡
145	æ	177	▒	209	Ð	241	±
146	Æ	178	▓	210	Ê	242	‗
147	ô	179	│	211	Ë	243	¾
148	ö	180	┤	212	È	244	¶
149	ò	181	Á	213	ı	245	§
150	û	182	Â	214	Í	246	÷
151	ù	183	À	215	Î	247	¸
152	ÿ	184	©	216	Ï	248	°
153	Ö	185	╣	217	┘	249	¨
154	Ü	186	║	218	┌	250	·
155	ø	187	╗	219	█	251	¹
156	£	188	╝	220	▄	252	³
157	Ø	189	¢	221	¦	253	²
158	×	190	¥	222	Ì	254	■
159	ƒ	191	┐	223	▀	255	nbsp

https://elcodigoascii.com.ar/

1.1. Tipos de ficheros

Tipo de fichero	Extensiones
Texto	.txt, .csv, .json, .xml, .html, .dat…
Imagen	.jpg, .gif, .bmp, .png, …
Video	.avi, .mp4, .mpeg, .mwv, …
Ejecución	.exe, .bat, .dll, .sys, …
…	.mp3, .zip, .iso, .tar, .pdf, …

```
cep, city, neighborhood, service, state, street
31270901, Belo Horizonte, Pampulha, correios, MG, Av. Presidente Antônio Carlos, 6627
```

CSV
Comma Separated Values

JSON
JavaScript
Object Notation

```
{
    "endereco": {
        "cep": "31270901",
        "city": "Belo Horizonte",
        "neighborhood": "Pampulha",
        "service": "correios",
        "state": "MG",
        "street": "Av. Presidente Antônio Carlos, 6627"
    }
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<endereco>
   <cep>31270901</cep>
   <city>Belo Horizonte</city>
   <neighborhood>Pampulha</neighborhood>
   <service>correios</service>
   <state>MG</state>
   <street>Av. Presidente Antônio Carlos, 6627</street>
</endereco>
```

XML
Extensible Markup
Language

1.2. Lectura de ficheros

- Leer un archivo en Python es sencillo y solo requiere los siguientes pasos:

1. Apertura del fichero en modo lectura:

   fichero = open('NombreFichero.ext', 'r')

2. Lectura por fichero completo o líneas (solo una por fichero abierto):

   fichero_completo = fichero.read()

   fichero_líneas = fichero.readlines()

   Linea_siguiente = fichero.readline()

2. Cerrado del fichero:

   fichero.close()

```
[33]: f = open("MiFichero.txt", 'r')
      print(f.readlines())
      f.close()

      ['L1\n', 'L2\n', 'L3\n', 'L4\n', 'L5']

[34]: f = open("MiFichero.txt", 'r')
      print(f.readline())
      f.close()

      L1

[35]: f = open("MiFichero.txt", 'r')
      print(print(f.read()))
      f.close()

      L1
      L2
      L3
      L4
      L5
      None
```

1.2. Lectura de ficheros

- Para procesar las secuencias retornadas por los métodos de lectura anteriores se pueden utilizar funciones de procesado de cadenas de caracteres y de conversión de tipos útiles:
  - cadena.split(caracter): divide la cadena por el carácter indicado.
  - float(cadena): convierte el tipo de la cadena para poder operarlo.
  - list(cadena): convierte la cadena de caracteres en una lista de caracteres.

```
[22]: cadena = "Hola mi nombre es Javier (3)."
      print(cadena)
      palabras = cadena.split(" ")
      print(palabras)
      letras = list(cadena)
      print(letras)
      numero = float(letras[-3])
      print(type(numero), numero)

      Hola mi nombre es Javier (3).
      ['Hola', 'mi', 'nombre', 'es', 'Javier', '(3).']
      ['H', 'o', 'l', 'a', ' ', 'm', 'i', ' ', 'n', 'o', 'm', 'b', 'r', 'e', ' ', 'e', 's', ' ', 'J', 'a', 'v', 'i', 'e', 'r', ' ', '(', '3', ')', '.']
      <class 'float'> 3.0
```

1.3. Escritura de ficheros

- Escribir un archivo en Python es sencillo y solo requiere los siguientes pasos:

1. Apertura del fichero en modo escritura (w / a / x):

   fichero = open('NombreFichero.ext', 'w' / 'a' / 'x')

   w: borra el fichero si existe.
   a: añade el contenido al final.
   x: devuelve error si ya existe el fichero.

2. Escritura en el fichero:

   fichero.write(" contenido que se desea añadir ")
   fichero.writelines(lista_multiples_contenidos)

   \n: carácter que indica salto de línea.

3. Cerrado del fichero:

   fichero.close()

```
[52]: f = open("MiFichero.txt",'x')

---------------------------------------------------------------------------
FileExistsError                           Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_23400\2179375171.py in <module>
----> 1 f = open("MiFichero.txt",'x')

FileExistsError: [Errno 17] File exists: 'MiFichero.txt'
```

```
[53]: f = open("NuevoFichero.txt", 'w')
      f.write("Nuevo contenido\n")
      f.close()
```

```
[54]: f = open("NuevoFichero.txt", 'r')
      print(f.read())
      f.close()

      Nuevo contenido
```

```
[55]: f = open("NuevoFichero.txt", 'a')
      lista = ["L2\n" , "L3\n", "L4\n"]
      f.writelines(lista)
      f.close()
```

```
[56]: f = open("NuevoFichero.txt", 'r')
      print(f.read())
      f.close()

      Nuevo contenido
      L2
      L3
      L4
```

2. Manejo avanzado de ficheros

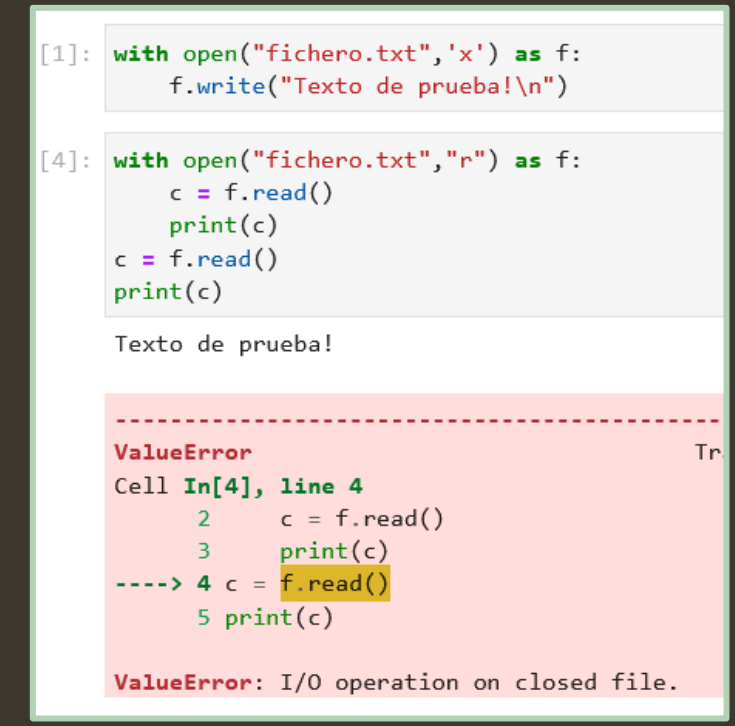
- Permite trabajar de manera eficiente incluso cuando los conjuntos de datos son muy extensos, evitando sobrecargar la memoria del sistema.
- Facilita la personalización del almacenamiento de resultados, adaptando el formato de salida a las necesidades específicas de cada proyecto.
- Brinda la posibilidad de gestionar información en distintos tipos de estructuras, desde tablas hasta datos numéricos complejos o binarios.
- Aporta mayor control y robustez en la manipulación de la información, haciendo que los procesos sean más seguros y confiables en contextos profesionales.

2.1. Contextos

Contexto: *Entorno o marco que garantiza la correcta apertura, uso y cierre de un fichero, asegurando que los recursos se gestionen de forma segura y eficiente.*

- El bloque de código contenido en el contexto, se ha de sangrar de forma que todas las operaciones que utilicen el fichero abierto en dicho contexto queden dentro.
- El contexto gestiona de forma eficiente los ficheros, cerrándolos al terminar el bloque de código, por lo que es recomendable para grandes volúmenes de datos y evitar fallos.

```
with open('NombreFichero.ext', 'w' / 'a' / 'x') as fichero:
        fichero.write("Texto de prueba\n")
        contenido = fichero.read()
```



```
[1]: with open("fichero.txt",'x') as f:
         f.write("Texto de prueba!\n")

[4]: with open("fichero.txt","r") as f:
         c = f.read()
         print(c)
     c = f.read()
     print(c)

     Texto de prueba!

     ---------------------------------------------
     ValueError                                Tr
     Cell In[4], line 4
           2     c = f.read()
           3     print(c)
     ----> 4 c = f.read()
           5 print(c)

     ValueError: I/O operation on closed file.
```

2.2. Ramificación e iteración

- Se puede utilizar de forma conjunta las estructuras de ramificación e iteración junto con las operaciones de ficheros y contextos para operar sobre los datos almacenados de forma operativa.
- Permiten automatizar tareas repetitivas de procesamiento, facilitando la manipulación y transformación sistemática de grandes cantidades de datos sin intervención manual.
- Permiten procesar línea a línea, lo que es eficiente para manejar ficheros grandes sin cargar todo el contenido en memoria.
- Facilitan el procesamiento secuencial y organizado de datos, adaptándose fácilmente a operaciones repetitivas y a la transformación de cada línea.

[14]:
```
with open("fichero.txt", 'w') as f:
    for v in ["L1", "L2", "L3"]:
        f.write(v+"\n")
```

[15]:
```
with open("fichero.txt", 'r') as f:
    i = 0
    for linea in f:
        if i != 0:
            print(linea)
        i = i + 1
```

```
L2

L3
```

2.3. Formateo

- Permite integrar valores dinámicos dentro de cadenas de texto para presentar resultados claros y estructurados.
  - F-strings: Integra directamente las variables con su formato en las cadenas de texto.
  - Método format(): Mapea las variables introducidas al método format() dentro de la cadena de texto.
- Estos formateos no solo pueden ser utilizados en las cadenas de texto que se escribirán en fichero sino también el todas las utilizadas.

```
[20]: valor = 2
      with open("fichero.txt", 'w') as f:
          f.write(f"El valor es {valor:.2f}\n")
          f.write("El valor es {v}\n".format(v = 10))

[21]: with open("fichero.txt", 'r') as f:
          for linea in f:
              print(linea)

      El valor es 2.00

      El valor es 10


[23]: print(f"El valor es {valor:.5f}")

      El valor es 2.00000
```

3. Ficheros binarios

Fichero binario: *Almacena datos en formato binario, no legible por humanos, ideal para guardar información precisa y eficiente como imágenes, programas o datos técnicos.*

- Acceso rápido y eficiente a datos, especialmente en aplicaciones donde el rendimiento es crítico.
- Conservan la exactitud de datos numéricos evitando conversiones y pérdida de precisión.
- Reducen espacio en disco comparado con ficheros de texto equivalentes.
- Útiles para almacenar resultados de simulaciones, imágenes, modelos 3D y otros datos técnicos.
- Existen múltiples formas de trabajar con datos binarios (pickle, struct, encoders, etc).

```
[41]: import pickle

      datos = {"Proyecto": "Obra 1",
               "Cargas": [120.5, 89.75]}

      with open("datos.bin", "wb") as f:
          pickle.dump(datos, f)
```

```
[42]: with open("datos.bin", "rb") as f:
          datos = pickle.load(f)
          print(datos)

      {'Proyecto': 'Obra 1', 'Cargas': [120.5, 89.75]}
```

3. Ficheros binarios

Fichero binario: *Almacena datos en formato binario, no legible por humanos, ideal para guardar información precisa y eficiente como imágenes, programas o datos técnicos.*

[55]:
```python
import struct

datos = struct.pack('fi', 3.1, 7)

with open("datos.bin", "wb") as f:
    f.write(datos)
```

[56]:
```python
with open("datos.bin", "rb") as f:
    contenido = f.read()
    valores = struct.unpack('fi', contenido)
    print(valores)
```

```
(3.0999999046325684, 7)
```

[57]:
```python
texto = "Ingeniería Civil"
numero = 12345

with open("datos.bin", "wb") as f:
    f.write(texto.encode('utf-8'))
    f.write(b'\n')
    f.write(numero.to_bytes(4, byteorder='little'))
```

[58]:
```python
with open("datos.bin", "rb") as f:
    texto_bytes = bytearray()
    while True:
        b = f.read(1)
        if b == b'\n' or b == b'':
            break
        texto_bytes += b
    texto_leido = texto_bytes.decode('utf-8')
    int_bytes = f.read(4)
    numero_leido = int.from_bytes(int_bytes, byteorder='little')

print("Texto leído:", texto_leido)
print("Número leído:", numero_leido)
```

```
Texto leído: Ingeniería Civil
Número leído: 12345
```

3. Ficheros estructurados (CSV)

Fichero estructurado: *Organiza datos en formatos definidos y ordenados, como tablas o registros, facilitando su manipulación y análisis en aplicaciones específicas.*

- Formato texto simple donde los datos están organizados en filas y columnas separadas por comas.
- Uso con módulo estándar csv para leer y escribir datos tabulares básicos.
- Con pandas, permite manejo eficiente de grandes conjuntos de datos estructurados y análisis avanzado.
- Ideal para datos tabulares simples como registros de ensayos, topografía o inventarios.

```
[28]: import csv

datos = [["Etiqueta", "Valor"], ["V1", 10], ["V2", 20]]
with open("valores.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerows(datos)
```

```
[29]: with open("valores.csv", "r") as f:
    reader = csv.reader(f)
    for fila in reader:
        print(fila)
```

```
['Etiqueta', 'Valor']
['V1', '10']
['V2', '20']
```

3. Ficheros estructurados (JSON)

Fichero estructurado: *Organiza datos en formatos definidos y ordenados, como tablas o registros, facilitando su manipulación y análisis en aplicaciones específicas.*

- Formato texto para almacenar estructuras de datos anidadas, como objetos y listas.
- Uso con módulo estándar json para intercambiar datos entre programas y configuraciones.
- Con pandas, facilita la conversión a tablas para análisis estadístico o datos heterogéneos.
- Perfecto para formatos flexibles, por ejemplo, datos BIM o configuraciones de proyectos.

```
[34]: import json

datos = {"Proyecto": "Obra 1",
         "Datos": {"Metros cuadrados":180, "Plantas":2}}

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

```
[35]: with open("datos.json", "r") as f:
    datos_leidos = json.load(f)
    print(datos_leidos)

{'Proyecto': 'Obra 1', 'Datos': {'Metros cuadrados': 180, 'Plantas': 2}}
```

3. Ficheros estructurados (Excel)

Fichero estructurado: *Organiza datos en formatos definidos y ordenados, como tablas o registros, facilitando su manipulación y análisis en aplicaciones específicas.*

- Formato binario o XML, muy usado en ingeniería para reportes y ensayos.
- Python usa librerías externas, pandas incluye funciones para leer/escribir Excel como read_Excel() o to_Excel().
- Excel maneja hojas, fórmulas y formatos, lo que permite informes profesionales desde datos sin procesar.
- Perfecto para intercambio con otros softwares y análisis con herramientas office.

```
[36]: import pandas as pd

datos = {"Material": ["Hormigón", "Acero"],
         "Resistencia": [25.6, 450]}

df = pd.DataFrame(datos)

df.to_excel("materiales.xlsx", index=False)

df_leido = pd.read_excel("materiales.xlsx")
print(df_leido)
```

```
   Material  Resistencia
0  Hormigón         25.6
1     Acero        450.0
```