Programación 1º iMAT

Grado en Ingeniería Matemática e Inteligencia Artificial - Comillas ICAI

por David Contreras Bárcena

Tema 10. Ficheros

10.1 Introducción

Los ficheros son estructuras de información que se almacenan en el disco duro del ordenador y son gestionados por el sistema de ficheros del sistema operativo. En la actualidad convivimos con una gran cantidad de ellos (hojas de cálculo, documentos, imágenes, etc.), pero ¿en qué momento son de utilidad en la programación? Solo recordad un detalle, tanto Word, como Photoshop, Excel, etc. son programas escritos en un lenguaje de programación como los que hacemos nosotros...

Todo ordenador posee unos dispositivos que representan la entrada estándar (**stdin**, teclado) y una salida estándar (**stdout**, consola). Al principio de la asignatura vimos que utilizámos estos dispositivos para interactuar con nuestro programa y realizar programas amigables. Ahora, en este tema, veremos cómo nuestro programa puede obtener la información de entrada o de salida, no por los dispositivos estándar, si no mediante ficheros. Así, la entrada de mi programa podrá venir dada por un fichero, en lugar del teclado, y la salida, por otro fichero, en lugar de la pantalla.

10.1.1 Tipos de ficheros

Existen dos tipos de ficheros básicamente:

- **Ficheros de texto**: los que representa información legible por el usuario al abrirlos como caracteres. Pueden ser manipulados desde cualquier editor de texto o programa Python. Desde Python, estos ficheros se corresponden con objetos del tipo **str**.
    - Los formatos más conocidos de ficheros de texto, son:
        - Ficheros de texto plano (**fichero.txt**):

```
Esto es un ejemplo de fichero
de texto plano. Puede representar información no estructurada
como ésta,
o información semi estructurada como la siguiente:
```

```
Luis 22 Madrid
Ana 18 Ciudad Real
```

- Ficheros CSV separados por comas (**fichero.csv**): mejora la delimitación de los campos y presenta una información estructurada donde la primera línea (en ocasiones, puede aparecer o no) representra su estructura:

```
nombre, edad, ciudad
Luis, 22, Madrid
Ana, 18, Ciudad Real
```

- Ficheros de texto JSON (**fichero.json**): aumenta todavía más la estructura de la información ya que cada línea define dicha estructura.

```
{
    {nombre: "Luis", edad: 22, ciudad: "Madrid"},
    {nombre: "Ana", edad: 18, ciudad: "Ciudad Real"}
}
```

- **Ficheros binarios**: aquellos que no representan información basada en caracteres. Pueden ser manipulados por aplicaciones específicas (editores de imágenes, reproductores de música o vídeo, etc) o programas Python que conocen exactamente su estructura contenido. Desde Python estos ficheros se corresponden con objetos de cualquier tipo **list**, **dict**, etc. Se dice que cuando se guardan objetos se produce un proceso de serialización.

Ejemplo

Si en nuestro programa trabajamos con un diccionario de dnis y nombres, y deseamos guardar su información en un fichero, tendremos dos posibilidades:

- Guardar la información del diccionario como cadenas de caracteres (tipos de datos str), es decir, como un fichero de texto. Esta información será legible por cualquier editor de texto, como se puede ver: fichero **dict.txt**.

```
diccionario = {
    11: "Luis",
    22: "Ana",
    33: "Lucía"
}
```

- Guardar los objetos como variables complejas, no representables mediante texto (tipos de datos avanzados, como los dict). Al guardar, por ejemplo, un diccionario, esta información solo será legible por un programa Python, por lo que no se podrá abrir desde un editor de texto, como se puede ver al mostrar el contenido del fichero **dict.obj**.

```
€•!}”(K Luis”K Ana”K! LucÃa”u.
```

Siempre que podamos, generaremos la información mediante ficheros de texto.

In [9]:
```
!type data\\dict.txt
```

```
11 Luis
```

In [11]:
```
!type data\\dict.obj
```

```
€•!}”(K‚Œ‚Luis”K‚Œ‚Ana”K!Œ‚LucÃa”u.
```

10.2 Ficheros de texto

Hay dos operaciones básicas que se realizan con ficheros:

- Lectura: proceso de lectura de su contenido para ser procesado por el programa.
- Escritura: proceso de escritura en un fichero del resultado de un programa.

Toda operación a realizar sobre un fichero requiere los siguientes pasos:

- **Apertura del fichero**: indicación al sistema operativo de la operación a realizar sobre el fichero. Por lo general y en esta asignatura, el fichero se abrirá solo en un solo modo: escrituta ("w"), lectura ("r") o agregación de información al fichero ("a"). En este proceso, se creará un objeto Python que gestionará la referencia (handle) contra el fichero físico.

```
objeto_handle_fichero = open(nombre_fichero, modo_operación)
```

- **Operación**: se escribirá o leerá el fichero.

```
objeto_handle_fichero.write(nueva_linea_a_escribir)
o
contenido_fichero = objeto_handle_fichero.readlines()
```

- **Cierre del fichero**: operación obligatoria para un correcto funcionamiento de la operación. A partir de este momento no se podrá manipular el fichero. Si se desean leer o escribir más datos, se deberá volver a abrir.

```
objeto_handle_fichero.close()
```

10.2.1 Escritura de ficheros

Apertura y cierre del fichero en modo escritura

Si el fichero no existe, se crea. Si el fichero existe (muy importante), se sobreescribe. Ojo, porque se perderá toda la información previamente almacenada.

In [15]:
```
fichero = open("data/prueba.txt", "w")
print(fichero)
print(type(fichero))
fichero.close()
```

```
<_io.TextIOWrapper name='data/prueba.txt' mode='w' encoding='cp1252'>
<class '_io.TextIOWrapper'>
```

Podemos preguntar en cualquier momento sobre el estado del fichero.

In [18]:
```
fichero = open("data/prueba.txt", "w")
print(fichero.closed)
fichero.close()
print(fichero.closed)
```

```
False
True
```

Proceso de escritura

Todo lo anterior está muy bien, pero el objetivo de abrir un fichero es para leer o escribir de él. :)

Para el proceso de escritura podemos utilizar los siguientes métodos:

- **write(str) -> int**: escribe una línea en el fichero. Devuelve, si nos interesa, el número de caracteres escritos.
- **writelines(list) -> None**: escribe una línea en el fichero.

Importante: el método *write* escribe strings de forma concatenada en el fichero sin añadir nuevas líneas ("\n") al final de los strings escritos.

Método write(str)

In [22]:
```
fichero = open("data/prueba1.txt", "w")
fichero.write("Hola")  # Se guardará en la primera línea del fichero
fichero.write("Esto es un ejemplo")  # Idem
fichero.write("de fichero")  # Idem, todo en la misma línea al no existir ningún
fichero.close()
```

El fichero *prueba.txt* contendrá el siguiente contenido:

```
HolaEsto es un ejemplo de fichero
```

Comprobemos este desde el sistema operativo.

Comprobación de la existencia del fichero en el Sistema Operativo

El comando que se muestra en la siguiente celda solo funciona en Windows, ya que se trata de una llamada al sistema operativo, no es un comando Python. En Ubuntu o MacOS habrá que ejecutar lo siguiente:

```
!ls data/prueba1.txt -la
-rw-r--r-- 1 David 197121 33 oct. 28 16:12 prueba1.txt
```

In [27]:
```
!dir data\prueba1.txt
```

```
 El volumen de la unidad C no tiene etiqueta.
 El n£mero de serie del volumen es: 4AC2-5CBC

 Directorio de C:\Users\davidcb\Dropbox\Departamentos\Python\asignatura\temario_n
otebooks\data

07/11/2024  17:09                32 prueba1.txt
               1 archivos             32 bytes
               0 dirs  819.796.549.632 bytes libres
```

Se puede comprobar que si se vuelve a abrir el fichero en modo escritura (existiendo el fichero), se sobreescribirá.

El siguiente comando muestra el contenido del fichero y solo funcionará en Windows. Se trata de una llamada al sistema operativo, no es un comando Python. En Ubuntu o MacOS habrá que ejecutar lo siguiente:

```
!cat data/prueba1.txt
HolaEsto es un ejemplo de fichero
```

In [31]:
```
!type data\prueba1.txt
```

```
HolaEsto es un ejemplode fichero
```

Creemos un nuevo fichero escribiendo el mismo contenido, pero en diferentes líneas:

In [34]:
```
fichero = open("data/prueba2.txt", "w")
fichero.write("Hola\n")  # Primera línea del fichero
fichero.write("Esto es un ejemplo\n")  # Segunda línea
fichero.write("de fichero\n")  # Tercera línea
fichero.close()
```

In [36]:
```
!type data\prueba2.txt
```

```
Hola
Esto es un ejemplo
de fichero
```

Operación append ("a")

Permite conservar el contenido del fichero y agregará al final del mismo el nuevo contenido.

In [39]:
```
fichero = open("data/prueba2.txt", "a")
fichero.write("Adiós\n")  # Se agregará esta línea al final del fichero
fichero.close()
```

In [41]:
```
!type data\prueba2.txt
```

```
Hola
Esto es un ejemplo
de fichero
Adiós
```

Método writelines(list)

In [44]:
```
lista = ["Uno", "Dos", "Tres"]
fichero = open("data/prueba3.txt", "w")
fichero.writelines(lista)
fichero.close()
```

In [46]:
```
!type data\prueba3.txt
```

```
UnoDosTres
```

In [48]:
```
lista = ["Uno\n", "Dos\n", "Tres\n"]
fichero = open("data/prueba4.txt", "w")
fichero.writelines(lista)
fichero.close()
```

In [50]:
```
!type data\prueba4.txt
```

```
Uno
Dos
Tres
```

10.2.2 Lectura de ficheros

En primer habrá que leer el fichero en modo lectura, para después leerlo. Esta lectura se podrá realizar de diferentes formas, como ya ocurriera en la escritura, mediante los métodos que proporciona:

- **read() -> str**: lectura de **todo el fichero** en un objeto string.
- **readline() -> str**: lectura de un línea del fichero. Para leer n líneas, se deberá invocar a este método n veces. Se controla el final de fichero por la lectura de una línea vacía (""). Ver también **for linea in file**.
- **readlines() -> list**: lectura de todo el fichero en una lista. Cada fila del fichero se guardará en una posición de la lista. Nótese cómo la longitud de la lista determinará las líneas del fichero.

Método read() -> str

In [54]:
```
fichero = open("data/prueba1.txt", "r")
contenido = fichero.read()
print(type(contenido))
print(contenido)
fichero.close()
```

```
<class 'str'>
HolaEsto es un ejemplode fichero
```

In [56]:
```
fichero = open("data/prueba2.txt", "r")
contenido = fichero.read()
print(type(contenido))
print(contenido)
fichero.close()
```

```
<class 'str'>
Hola
Esto es un ejemplo
de fichero
Adiós
```

Método readline() -> str

Permite leer línea a línea el fichero. Por cada ejecución del método, lee la línea devolviendo un string y se mueve a la línea siguiente.

In [59]:
```
fichero = open("data/prueba1.txt", "r")
linea = fichero.readline()
print(linea)
fichero.close()
```

```
HolaEsto es un ejemplode fichero
```

In [61]:
```
fichero = open("data/prueba2.txt", "r")
linea = fichero.readline()
print(linea)
linea = fichero.readline()
print(linea)
linea = fichero.readline()
print(linea)
fichero.close()
```

```
Hola

Esto es un ejemplo

de fichero
```

In [63]:
```
fichero = open("data/prueba2.txt", "r")
lineas = []
linea = fichero.readline()

while linea != "":
    lineas.append(linea)
    print(linea)
    linea = fichero.readline()

fichero.close()
lineas
```

```
Hola

Esto es un ejemplo

de fichero

Adiós
```

Out[63]:
```
['Hola\n', 'Esto es un ejemplo\n', 'de fichero\n', 'Adiós\n']
```

In [65]:
```
fichero = open("data/prueba2.txt", "r")
lineas = []
```

```
linea = fichero.readline().rstrip()

while linea != "":
    lineas.append(linea)
    print(linea)
    linea = fichero.readline().rstrip()

fichero.close()
lineas
```

```
Hola
Esto es un ejemplo
de fichero
Adiós
```

Out[65]: `['Hola', 'Esto es un ejemplo', 'de fichero', 'Adiós']`

Estructura for in

Como ya vimos en las estructuras iterativos, el bucle while se puede actualizar a una estructura más cómoda y moderna como el **for in**. En este caso quedaría...

In [68]:
```
fichero = open("data/prueba2.txt", "r")
for linea in fichero:
    print(linea.rstrip())

fichero.close()
```

```
Hola
Esto es un ejemplo
de fichero
Adiós
```

Se pueden combinar ambas formas de trabajar:

In [71]:
```
fichero = open("data/prueba2.txt", "r")
primera_linea = fichero.readline()
print("1ª linea: ", primera_linea.rstrip())
for linea in fichero:
    print("Líneas restantes:", linea.rstrip())

fichero.close()
```

```
1ª linea:  Hola
Líneas restantes: Esto es un ejemplo
Líneas restantes: de fichero
Líneas restantes: Adiós
```

Método readlines() -> list

In [74]:
```
fichero = open("data/prueba1.txt", "r")
lineas = fichero.readlines()
print(lineas)
print(f"Líneas del fichero: {len(lineas)}")
fichero.close()
```

```
['HolaEsto es un ejemplode fichero']
Líneas del fichero: 1
```

In [76]:
```
fichero = open("data/prueba2.txt", "r")
lineas = fichero.readlines()
print(lineas)
print(f"Líneas del fichero: {len(lineas)}")
fichero.close()
```

```
['Hola\n', 'Esto es un ejemplo\n', 'de fichero\n', 'Adiós\n']
Líneas del fichero: 4
```

10.3 Operaciones avanzadas

10.3.1 Sentencia with

Mediante la sentencia **with** se puede crear un bloque que agrupe las operaciones sobre ficheros, ya sea de lectura o de escritura.

```
with open(fichero) as fichero:
    fichero.read() o fichero.write()
```

Ventajas del bloque **with**:

- Permite agrupar todas las funciones que se realizan con el fichero.
- Simplifica la gestión de erores.
- Al salir el fichero se cerrará automáticamente.

In [80]:
```
with open("data/prueba2.txt", "w") as fichero:
    fichero.write("Hola\n")
    fichero.write("Esto es un ejemplo\n")
    fichero.write("de fichero\n")
```

10.3.2 Gestión del carácter de salto de línea

Como se puede observar, el fichero *prueba2.txt* posee al final de cada línea, el salto de línea que añadimos.

In [83]:
```
with open("data/prueba2.txt", "r") as fichero:
    lineas = fichero.readlines()
print(lineas)

for linea in lineas:
    print(linea)
```

```
['Hola\n', 'Esto es un ejemplo\n', 'de fichero\n']
Hola

Esto es un ejemplo

de fichero
```

rstrip(): eliminación de blancos y secuencias de escape

In [86]:
```
cadena = "Hola\n    \t  "
print(f"#{cadena.rstrip()}#")
```

```
#Hola#
```

In [88]:
```
with open("data/prueba2.txt", "r") as fichero:
    lineas = fichero.readlines()
print("Leído del fichero: ", lineas)

for i in range(len(lineas)):
    lineas[i] = lineas[i].rstrip()

print("Eliminación del \\n: ", lineas)

print("\nLista limpia:")
for linea in lineas:
    print(linea)
```

```
Leído del fichero:  ['Hola\n', 'Esto es un ejemplo\n', 'de fichero\n']
Eliminación del \n:  ['Hola', 'Esto es un ejemplo', 'de fichero']

Lista limpia:
Hola
Esto es un ejemplo
de fichero
```

10.3.3 Lectura de todo un fichero línea a línea

El siguiente ejemplo lee todas las líneas de un fichero una a una.

In [91]:
```
fichero = open("data/prueba2.txt", "r")
linea = fichero.readline()
while linea != "":
    print(linea)
    linea = fichero.readline()
fichero.close()
```

```
Hola

Esto es un ejemplo

de fichero
```

In [93]:
```
fichero = open("data/prueba2.txt", "r")
linea = fichero.readline().rstrip()
while linea != "":
    print(linea)
    linea = fichero.readline().rstrip()
fichero.close()
```

```
Hola
Esto es un ejemplo
de fichero
```

10.3.4 Excepciones lanzadas - Leer después del tema de Excepciones

Cuando veamos excepciones se entenderá este código.

In [96]:
```
try:
    with open("data/prueba_no_existe.txt", "r") as fichero:
```

```
        contenido = fichero.read()
        print(contenido)
except FileNotFoundError as error:
    print(error)
```

```
[Errno 2] No such file or directory: 'data/prueba_no_existe.txt'
```

10.3 Ficheros binarios -serializados- (nivel avanzado) - Leer después del tema de Objetos

Los ficheros binarios que vamos a abordar son los relacionados con el almacenamiento de objetos Python. El único objeto Python que tiene una naturaleza de texto es el **str**, el resto de los objetos a almacenar en disco serán tomados como ficheros binarios: list, dict, Persona, Coche, etc.

PAra este fin utilizaremos la librería **pickle**. Mediante los métodos **dump(objeto, fichero)** y **load(fichero)->obj** llevaremos a cabo las tareas de escritura y lectura respectivamente.

Un detalle importante a tener en cuenta es que en el modo de apertura del fichero, deberemos añadir una "b" al final para indicar esta particularidad (binario). Básicamente con este detalle estaríamos diciendo que no es necesario convertir los bits a caracteres de texto, ya que representan otro tipo de información.

Así, hablaremos de lectura ("rb"), escritura ("wb") y agregación ("ab").

https://docs.python.org/3/library/pickle.html

10.3.1 Lectura/escritura de objetos Python

Supongamos que queremos almacenar el resultado de un programa que tenemos en un diccionario:

In [1]:
```
diccionario = {
    11: "Luis",
    22: "Ana",
    33: "Lucía"
}
```

Para la escritura utilizaremos la librería pickle con su método **dump**.

In [2]:
```
import pickle

with open("data/dict.obj", "wb") as fichero:
    pickle.dump(diccionario, fichero)
```

Analizando el resultado en el disco duro mediante el sistema operativo, vemos que efectivamente se trata de un fichero binario, no se entiende nada. :)

In [3]:
```
!type data\\dict.obj
```

```
€•!}”(K€Luis”K€Ana”K!€LucÃa”u.
```

Para leer el fichero, utilizaremos el método **load**.

In [4]:
```
with open("data/dict.obj", "rb") as fichero:
    diccionario_leido = pickle.load(fichero)

print(diccionario_leido)
```

```
{11: 'Luis', 22: 'Ana', 33: 'Lucía'}
```

Almacenando nuestros propios objetos

Cuando deseemos almacenar nuestros propios objetos, lo haremos de la misma forma.

In [5]:
```
class Persona:
    def __init__(self, nombre:str, edad:int):
        self.nombre = nombre
        self.edad = edad

    def __str__(self):
        return f"{self.nombre} {self.edad}"

persona1 = Persona("Luis", 22)
persona2 = Persona("Ana", 18)
```

In [6]:
```
import pickle

with open("data/personas.obj", "wb") as fichero:
    pickle.dump(persona1, fichero)
    pickle.dump(persona2, fichero)
```

Y para la operación opuesta de lectura, procederemos de la misma manera que cuando operábamos con el diccionario.

In [7]:
```
with open("data/personas.obj", "rb") as fichero:
    persona = pickle.load(fichero)
    print(persona)
    persona = pickle.load(fichero)
    print(persona)
```

```
Luis 22
Ana 18
```

En el ejemplo anterior hemos leído dos personas, porque sabíamos de antemano el número de objetos almacenados que teníamos en el fichero. Pero, ¿y si no sabemos el número de objetos que existen en el fichero? Que será lo normal...

En ese caso trabaremos con la marca de final de fichero (End Of File - EOF) que poseen todos los ficheros. La forma de detectar esta marca se gestiona en Python (como en otros lenguajes) mediante la excepción **EOFError**. La diferencia de esta excepción respecto al resto, es que en este caso no se corresponderá con un error, significará que todo ha funcionado correctamente.

In [186…
```
try:
    with open("data/personas.obj", "rb") as fichero:
        try:
            personas = []
```

```
            while True:
                persona = pickle.load(fichero)
                personas.append(persona)
        except EOFError:  # No es un error en realidad
            for persona in personas:
                print(persona)
except FileNotFoundError as error:
    print(error)
```

```
Luis 22
Ana 18
```

10.4 Serializar o no serializar, esa es la cuestión

Hay que tener en cuenta que siempre seremos capaces de almacenar la información de nuestros objetos en forma de texto utilizando ficheros planos, CSV o JSON.

Texto plano

In [208…
```
fichero = open("data/personas.txt", "w")
fichero.write(str(persona1))
fichero.write("\n")
fichero.write(str(persona2))
fichero.write("\n")
fichero.close()
```

In [209…
```
!type data\personas.txt
```

```
Luis 22
Ana 18
```

CSV

In [210…
```
fichero = open("data/personas.csv", "w")
fichero.write(f"{persona1.nombre}, {persona1.edad}")
fichero.write("\n")
fichero.write(f"{persona2.nombre}, {persona2.edad}")
fichero.write("\n")
fichero.close()
```

In [211…
```
!type data\personas.csv
```

```
Luis, 22
Ana, 18
```

JSON

In [204…
```
fichero = open("data/personas.json", "w")
fichero.write(f"{{nombre: \"{persona1.nombre}\", edad: {persona1.edad}}}")
fichero.write("\n")
fichero.write(f"{{nombre: \"{persona2.nombre}\", edad: {persona2.edad}}}")
fichero.write("\n")
fichero.close()
```

In [205…
```
!type data\personas.json
```

```
{nombre: "Luis", edad: 22}
{nombre: "Ana", edad: 18}
```

Programación 1º iMAT

Grado en Ingeniería Matemática e Inteligencia Artificial - Comillas ICAI

por David Contreras Bárcena