# Ficheros

## Curso de Python 3 Iniciación

César Husillos Rodríguez

IAA-CSIC

1-5 de Abril de 2019

# Índice

# Índice

# Ficheros

## Introducción

- El trabajo con ficheros es fundamental para cualquier programador.
- Hacen posible
  - Almacenamiento de información.
  - Su distribución (*portabilidad*).
  - su recuperación

Ficheros

César Husillos Rodríguez

# Ficheros

## ¿Para qué sirven?

Sus posibles usos son infinitos. Entre los más habituales están:

- Distribución de información en diferentes formatos (*portabilidad*).
- Establecimiento de parámetros de configuración de aplicaciones, Sistemas Operativos...
- Registro de actividad llevada a cabo en cualquier instrumento/aplicación.
- Identificación de usuarios en la red (certificados digitales).

# Ficheros

## Tipos

Atendiendo a la forma en la que se almacena la información, se distingue entre

- Formato en texto plano (txt, docx, csv, json...)
  - Legibles por cualquier aplicación que permita editar texto.
- Formato en binario (jpg, bmp, exe,...)
  - Son específicos y propietarios de determinadas aplicaciones o Sistemas Operativos.

En este curso trabajaremos sobre un formato de ficheros de texto.

# Ficheros

## Metodología

- El trabajo con ficheros en PYTHON3 (y cualquier otro lenguaje de programación) implica tres fases:
  1. **Apertura** del fichero.
  2. **Lectura/Escritura** del fichero.
  3. **Cierre** del fichero.

# Ficheros

## Apertura

- Para abrir un fichero usamos la función

```
open(rutaFichero[, modoApertura][1])
```

- Como *parámetro obligatorio*, necesita la ubicación del archivo en el sistema de ficheros del ordenador (`rutaFichero`). Esta ubicación puede ser
  - Absoluta (desde el primer directorio del disco, ya sea **C:**, **D:** (Windows) o **/** (LINUX o MAC)).
  - Relativa (desde el directorio actual)
- El *parámetro opcional* es el modo de apertura (`modoApertura`).
  - Modo de lectura (``'r'``, es el **valor por defecto**).
  - Modo de escritura (``'w'``).
  - Modo de escritura para agregar información **al final** del fichero (``'a'``).

[1] Un parámetro de una función entre `[]`, indica que es **opcional**.

# Ficheros

## Apertura

- Si todo ha ido bien, la función `open` devuelve un **objeto de tipo fichero** (`_io.TextIOWrapper`). A través de este objeto se accede a la información.
- Si no existe un fichero que tratamos de abrir en modo lectura, se lanza una excepción del tipo `FileNotFoundError`.

## Error de apertura en lectura

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

# Ficheros

## Lectura

- Una vez tengo acceso a un fichero en modo de lectura, puedo revisar su contenido de tres formas:
  - Leer todo el contenido.
  - Leer todo o parte del contenido.
  - Leer de línea en línea.
- La elección de un procedimiento u otro depende de nuestro algoritmo de interacción con el fichero.

# Ficheros

## Lectura con read()

- Usaremos el método `read()` del `objetoFichero` devuelto por la función `open`, en la forma `objetoFichero.read()`
- Devuelve el contenido del fichero en forma de cadena de texto.

## ejemplo1.txt

```
Primera linea
Segunda linea
```

## Ejemplo de lectura completa

```
>>> fichero = open('ejemplo1.txt')
>>> texto = fichero.read()
>>> texto
'Primera linea\nSegunda linea\n'
```

# Ficheros

## Lectura con `readlines()`

- Usaremos el método

  ```
  readlines(hint = -1)
  ```

  del `objetoFichero` devuelto por la función `open`, en la forma

  ```
  objetoFichero.readlines([numLineas])
  ```

- El parámetro opcional `hint` representa el número de líneas a leer. Tiene por defecto el valor `-1` que significa que las lea todas.
- Si el número de lineas que queremos leer es mayor que las que contiene el fichero, lee todas.
- Devuelve una lista.
  - Cada elemento de la lista es una línea.
  - El número de elementos viene dado por el valor del parámetro `hint` o por el número máximo de líneas del fichero.

# Ficheros

## Lectura con `readlines()`

- Cada línea incluye el carácter retorno de carro (`\n`) al final, como parte de la línea.

## Ejemplo de lectura con `readlines()`

```
>>> fichero = open('ejemplo1.txt')
>>> fichero.readlines()
['Primera linea\n', 'Segunda linea\n']
>>> fichero = open('ejemplo1.txt')
>>> fichero.readlines(100)
['Primera linea\n', 'Segunda linea\n']
>>> fichero = open('ejemplo1.txt')
>>> fichero.readlines(1)
['Primera linea\n']
>>> fichero.readlines(1)
['Segunda linea\n']
>>> fichero.readlines(1) # Ya no hay mas lineas
[]
```

# Ficheros

## Lectura con `readline()`

- Para leer una sóla línea cada vez (o hasta el final del fichero si no existe salto de línea) usaremos

  `readline()`

  Es un método del `objetoFichero` devuelto por la función `open`. Se invoca en la forma

  `objetoFichero.readline()`

- Devuelve la línea o una línea vacía si se ha encontrado el fín de fichero (*EOF (End Of File)*).

## Ejemplo de lectura con `readline()`

```
>>> fichero = open("ejemplo1.txt")
>>> fichero.readline()
'Primera_linea\n'
>>> fichero.readline()
'Segunda_linea\n'
>>> fichero.readline()
''
```

# Ficheros

## Elección del modo de lectura

- Cuando el fichero tiene poca información (**tamaño del fichero pequeño**),
  - No hay problema en usar cualquier función de lectura.
  - Es posible procesar todo el contenido a la vez.
- Si el **fichero** es **muy grande** debemos tener cuidado.
  - La operación de lectura de un fichero implica su carga en memoria.
  - Si la RAM de nuestro equipo es inferior al tamaño del fichero más lo que ocupan el sistema operativo y el resto de aplicaciones activas, seguro que se ralentizará[2] o incluso 'colgará'.

[2] Nuestro ordenador comenzará a *paginar*, es decir a usar el disco duro como ampliación de la memoria RAM.

# Ficheros

## Elección del modo de lectura

- Debe tenerse en cuenta que cada vez que se ejecuta una instrucción de lectura, el sistema interrumpe el programa PYTHON, accede al disco, trae la información y la carga en memoria.
  - Muchas operaciones de lectura pueden ralentizar el programa.
  - Las lecturas de disco son mucho más lentas que la memoria RAM (incluso con disco duros de estado sólido (*SSD*, Solid-State Drive)).
- Por último, interviene el diseño de nuestro algoritmo para resolver el problema planteado. Puede necesitar cargar toda la información a la vez.

# Ficheros

## `objetoFichero` como `iterator`

- El `objetoFichero` devuelto por la función `open(ruta)` puede funcionar como un **iterador**, es decir, como una entidad que puede ser usada en bucles.
- Es muy eficiente porque no carga los contenidos a los que apunta hasta que se piden explícitamente.

## Ejemplo de lectura iterativa

```
>>> objFichero = open('fichero.txt')
>>> for linea in objFichero: # iteramos
...       linea
...
'Primera linea\n'
'Segunda linea\n'
```

# Ficheros

## Escritura

- Para poder escribir en un fichero es necesario abrirlo en modo escritura (`'w'`).

```
objFichero = open('ruta', 'w')
```

- Si el fichero no existe, se crea en la `ruta` establecida como parámetro.
- Si existe, se abre en modo escritura, ¡¨borrando todo el contenido previo!

# Ficheros

## Escritura

- Una vez abierto el fichero para escritura, podemos guardar información.
- Se usará el método `write(cadena)` del `objetoFichero` para escribir, en la forma

```
objFichero.write('texto a escribir')
```

- El método `write` devuelve el número de caracteres escritos en el fichero.

## Ejemplo de escritura

```
>>> objFichero2 = open('ejemplo2.txt', 'w')
>>> objFichero2.write("Hola␣Mundo!")
11
>>> objFichero2.write("Esta␣es␣la␣primera␣vez␣que
escribo␣en␣un␣fichero␣con␣PYTHON3!!")
62
>>> objFichero.close() # cerramos el fichero
```

# Ficheros

## Escritura

### Resultado de escritura

```
Hola Mundo!Esta es la primera vez que escribo en un fichero con PYTHON3!!
```

- ¿Ve algo raro?
- Claro que sí. Ha escrito usando dos llamadas al método `write()` y el contenido se ha escrito en una sola línea.
- Para escribir varias líneas debe que establecerlo manualmente, insertando el carácter de nueva línea `\n` en la cadena de texto que le pasa al método `write`.

# Ficheros

## Escritura

- Si lo que necesitamos es escribir más información al final de un fichero existente, abrimos el fichero en modo `'a'` (de *append*).
  - Si el fichero no existe, lo crea.
  - Si el fichero existe, lo abre para **escritura al final de su contenido**.
- Se usará el método `write(cadena)` del `objetoFichero` para escribir, en la forma

```
objFichero.write('texto a escribir')
```

- El método `write` devuelve el número de caracteres escritos en el fichero.

# Ficheros

## Escritura

Ficheros

César Husillos Rodríguez



## Ejemplo de escritura al final

```
>>> objFichero2 = open('ejemplo2.txt', 'a')
>>> objFichero2.write('\nAhora␣agrego␣una␣linea␣mas␣al␣final.')
37
>>> objFichero2.write('\nY␣finalizo␣con␣esta␣linea.')
27
>>> objFichero2.close()
```

## Resultado

```
Hola Mundo!Esta es la primera vez que escribo en un fichero con PYTHON3!!
Ahora agrego una linea mas al final.
Y finalizo con esta linea.
```

# Ficheros

## Cierre

- Tras finalizar las operaciones de lectura/escritura sobre un fichero, debe cerrarse.
- Motivos:
  - Porque **los cambios realizados se escriben en disco**.
  - Porque **se libera la memoria** ocupada por el `objetoFichero` en el programa PYTHON.
  - Porque **se evitan problemas**, como abrir un fichero dos o más veces en el mismo programa (**concurrencia**).
  - Porque sólo se puede tener **un número finito de ficheros abiertos** simultáneamente[3]
- La forma de cerrarlo es usar el método `close()` del `objetofichero` devuelto por la función `open`.

```
objetoFichero.close()
```

- Las lecturas/escrituras fallan en un fichero cerrado.

[3] El número de ficheros abiertos simultáneamente depende del sistema operativo.

# Ficheros

## Modo alternativo

- Existe una alternativa a la forma de trabajo con ficheros vista hasta ahora. Se trata el bloque `with`.

## Sintaxis `with`

```
with open('rutaFichero', 'modo') as alias:
    # operaciones con (el objetoFichero) alias
```

- El `objetoFichero` devuelto por la función `open` y referenciado por `alias`, **vive dentro del bloque `with`**. Cuando éste termina, el fichero se cierra automáticamente.

# Ficheros

## Ejemplos del modo alternativo

### Escritura al final

```
>>> with open('ejemplo2.txt', 'a') as objFichero2:
...     objFichero2.write('\nLinea escrita con el bloque with.')
...
34
```

### Lectura

```
>>> with open('ejemplo2.txt') as objFichero2:
...     objFichero2.read()
...
'Hola Mundo!Esta es la primera vez que escribo en un
fichero con PYTHON3!!\nAhora agrego una linea mas
al final.\nY finalizo con esta linea.\nLinea escrita
con el bloque with.'
```

# Ejercicios propuestos

## Ficheros

1. Descargue el fichero `ejemplo1.txt`. Escriba un programa al que lea ese fichero e imprima por pantalla el número de caracteres que contiene (también cuentan los espacios en blanco, retornos de carro y líneas de puntuación).
2. Descargue el fichero `listado.csv`. Genere un programa que lo lea y muestre por pantalla la lista de nombres que contiene.
3. Escriba un programa que lea el fichero `listado.csv` e imprima por pantalla el valor máximo, el mínimo y la media de las Calificaciones.
4. Escriba un programa que lea el contenido del fichero `listado.csv` y genere otro fichero llamado `mejorAlumno.csv` donde se guarde el nombre y la calificación más elevada. El formato del nuevo fichero será el siguiente:
   - La cabecera será: `nombre,calificacion`
   - La siguiente línea corresponderá a los datos del alumno/a con mayor calificación. El formato será el mismo que el de la cabecera. El nombre deberá estar entre comillas dobles (").

# Ejercicios propuestos

## Ficheros

5 Escriba un programa que pida información del usuario por pantalla. Almacene todo lo que escriba hasta que pulse el salto de línea (`return` o `enter`) en el fichero `registro.txt`. Vuelva a ejecutar su programa. Agregue en otra línea todo lo escrito por segunda vez en el mismo fichero. Asegúrese de no perder la información anterior.

6 Cree un nuevo fichero `listadoFinal.csv` agregando a `listado.csv` la información contenida en `listado2.csv`.

Asegúrese de que no hay registros duplicados en `listadoFinal.csv`.