Tema 8: Input-Output Management

Apuntes

Índice

[**Entrada y salida (I/O) 2**](#_zblrtikey73y)

[Dispositivos de entrada y salida 2](#_ld1qflqte35l)

[**Entrada por teclado 2**](#_x6r3jxbhcafg)

[**Salida por pantalla y otros dispositivos 2**](#_vmvaix5im2q)

[**Formato de print 3**](#_h8d7sf7apki8)

[**Parámetros en la línea de comandos 4**](#_uxztdj1ieh73)

[**Ficheros 4**](#_akevr4md4kwh)

[De texto 5](#_kusypjkovlbx)

[Binarios 6](#_205wrlwyqip8)

[**Operaciones en los ficheros 6**](#_2bjoat7439jq)

[Lectura de ficheros 8](#_esstjz2i7p0m)

[Escritura de ficheros 10](#_a00iq4zgbxie)

[Actualización de ficheros 11](#_oz06y7pbxoev)

# Entrada y salida (I/O)

Para conseguir la comunicación entre la máquina y el usuario.

Entrada/Input: aportar datos al programa para que los procesen

Salida/Output: muestra información al usuario

LEER: <https://docs.python.org/3/tutorial/inputoutput.html>

## Dispositivos de entrada y salida

* Físicos:
  + Entrada: teclado, ratón, cámara web, control pad…
  + Salida: pantalla, impresora, tarjeta de sonido, cascos…
  + Dual (i/o): pantalla táctil, control remoto con feedback, discos/dispositivos de almacenamiento, cámara digital, proyector, otros (móvil o tablet)...
* Lógicos:
  + Ficheros: colección de datos que se manipula de manera unitaria y que tienen asociado un nombre único en el sistema y se almacena en un dispositivo de memoria permanente
  + Puertos y/o sockets (i/o):
    - Puerto: Vía de comunicación de protocolos
    - Socket: Similar pero es entre programas

# Entrada por teclado

Hasta ahora siempre lo hemos hecho con teclado con la función *input()*.

Con el input() el programa se detiene hasta que se introduzcan los datos y se pulse intro, todo el contenido se guardará en una variable.

Se puede ejecutar sin argumentos o iniciando el prompt

Ejemplo:

myvar = input(“Introduzca un número: “)

# Salida por pantalla y otros dispositivos

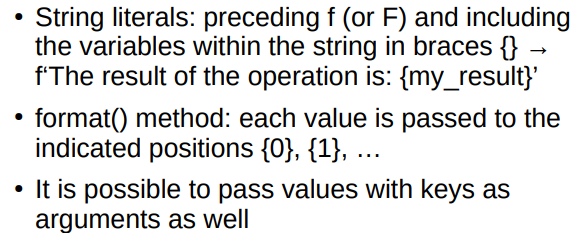
Con la función print().

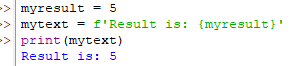
Esta función recoge la información y la muestra como argumentos por pantalla.

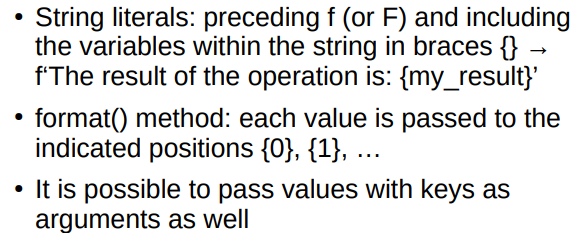
Tiene más parámetros a parte del texto a mostrar:

* file=sys.stdout → por defecto el print pone el resultado en un archivo que será la salida estándar del sistema (pantalla). El print funciona de forma interna con ficheros
* flush=False → ¿?

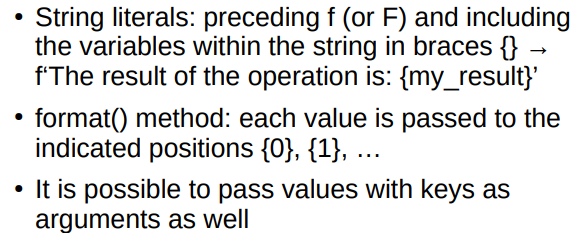
# Formato de print

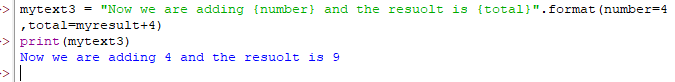




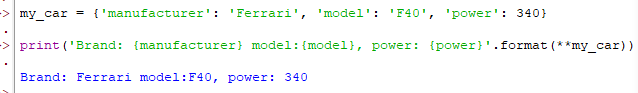


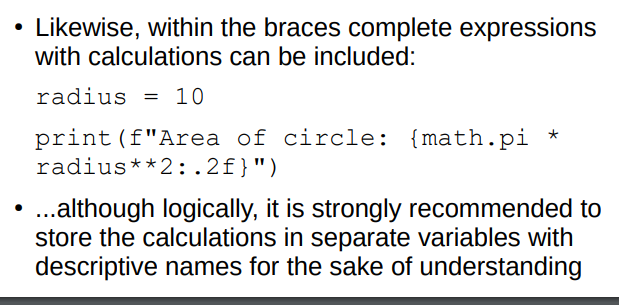






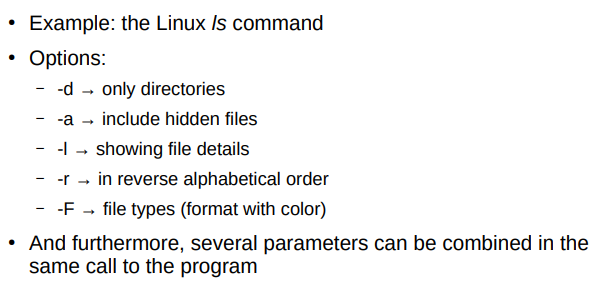
—------------





# Parámetros en la línea de comandos

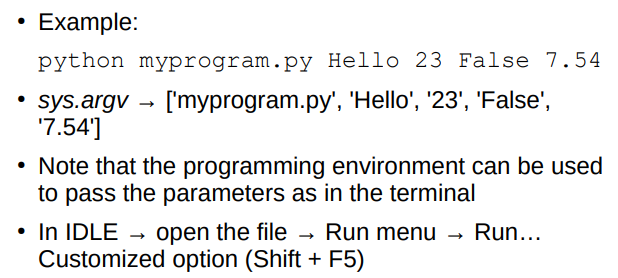
Cualquier programa de python puede lanzarse desde la línea de comandos/terminal.



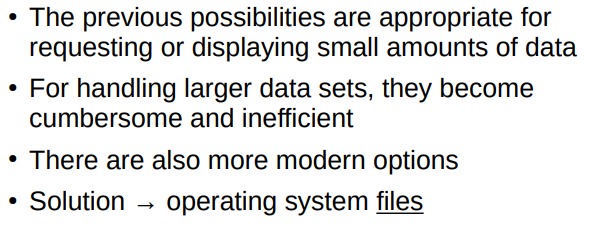
**¿Cómo proporcionar los parámetros en línea de comando?**

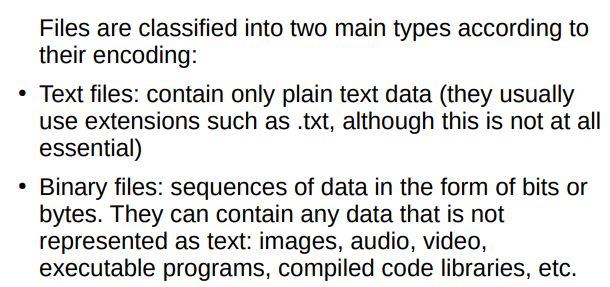
Al ejecutar el script se pondrán los parámetros.

Los parámetros se almacenarán en la variable sys.argv. Esta variable tiene una lista indexada, siendo el primer parámetro (0) el nombre del programa en sí. Se debe importar el módulo sys.



# Ficheros





## De texto

Contienen solo texto plano, usando extensiones como .txt, .html, .css, etc.

Pueden abrirse con un editor de texto normal y es posible leerlo.

**Importante** → los documentos de texto formateado no son ficheros de texto (.rtf, .odt, .doc, …)

Se estructura en líneas y cada una termina con el carácter \n (salto de línea).

El símbolo EOF marca el final de un fichero.

Formatos:

* Actualmente: Unicode (UTF-8)
* Antes: ASCII, ISO-8859-1

¿Para qué se usan los ficheros de texto?

* Código fuente
* Data exchange (XML, JSON, CSV)
* Settings storage
* Instrucciones (read.me), información de usuario…

Ventajas:

* Se entienden (los caracteres están directamente escritos y se puede leer)
* Podemos manipularlos y modificarlos, pudiendo añadir o modificar la información
* Es un formato universal
* Normalmente permiten un gran grado de compresión sin la pérdida de información.

Desventajas:

* Tienen limitaciones a la hora de almacenar otros tipos de datos.
* Sensibles a posibles errores al manejarlos.

## Binarios

Secuencias de datos en forma de bits o bytes. Solo contienen datos que no se representan como texto, es decir, imágenes, audio, video, programas ejecutables, librerías de código compilado, fichero de texto enriquecido (word, office), pdf.

Los ficheros binarios se usan cuando los datos no permiten o no sea conveniente representación textual. Este formato es usado con información más delicada y sensible.

Los ficheros binarios incluyen una cabecera con metadatos sobre el propio fichero.

Para manipular los ficheros binarios se requiere de un programa concreto que conozca el formato concreto.

Ventajas:

* Eficientes
* Versátiles (cualquier tipo de contenido y propósito)
* Su estructura y organización se puede elegir por el programador

Desventajas:

* No es legible para las personas
* Se necesita un software o aplicación específica.
* Pueden ser específicos de un sistema operativo o software en concreto

# Operaciones en los ficheros

En un lenguaje de programación es posible trabajar con ambos tipos de ficheros.

En general son operaciones de bajo nivel que pueden ser delicadas

Una parte de estas operaciones se definen en el propio lenguaje

Operaciones:

* Abrir → se debe indicar el nombre del archivo, el modo de uso que se le va a dar (lectura, escritura..) y el tipo de archivo.
* Cerrar
* Lectura → los datos se llevan del archivo a la memoria (variables)
* Escritura → espacios de la memoria se almacenan en un archivo
* Borrar
* Actualizar/modificar (no es lo más frecuente)

Concepto importante: **Puntero de posición** → es a partir de donde se recorrerá el archivo, apunta un lugar

**Abrir ficheros:**

* Se usa la función open()
* Devuelve un objeto llamado file handler (estructura de datos que se debe guardar en una variable y te permitirá trabajar con el fichero)
* Parámetros de open()
  + Nombre del fichero o ruta (requerido)
  + Modo de trabajo (opcional). Por defecto tomará r y t, es decir, se abrirá para lectura y será en formato texto.
    - 'r', 'w', 'a' → lectura, escritura (borrando todo lo que ya había previamente), append (escritura por el final, respetando lo que ya hay)
    - 'b', 't', '+' → binario, de texto, ¿+? (que también se puede leer, se usa para lectura y escritura)
    - Se pueden poner una opcion de cada grupo, por ejemplo wt para escribir un fichero de texto

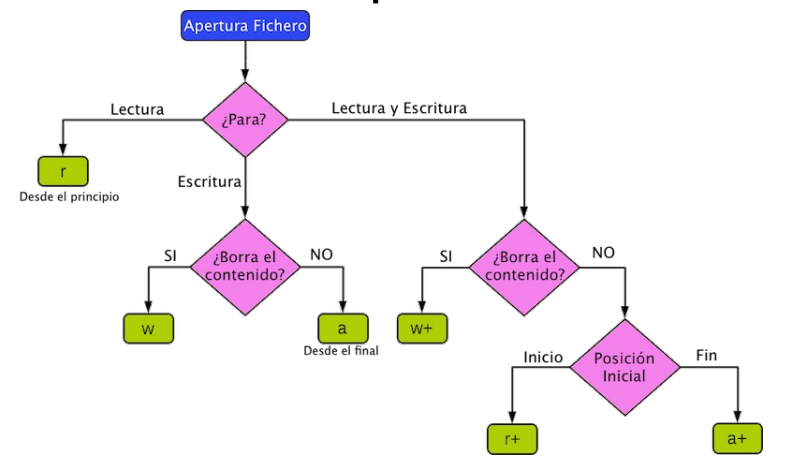
**Cerrar ficheros:**

* Se usa el método close(). Requiere poner archivo.close()

**Objeto file handler/ manejador de fichero?:**

* Contiene metadatos sobre el fichero
  + Cerrado (bool) → indica si se ha cerrado (True) o no (False)
  + Modo (str) → indica el modo de trabajo con el que se ha abierto el fichero (r,w,a)
  + Nombre (str) → del fichero
* Recuerda dónde está localizado el puntero del fichero, avanza conforme se va leyendo el fichero, si añadimos algo estará al final.

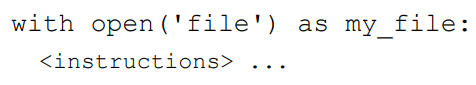
Modos de trabajo:



**Ejemplos:**

* myfile1 = open('myfile.txt') → default
  + Abre myfile.txt en modo texto y lectura, ambos por defecto
* myfile2 = open('myfile2.txt', 'r') → similar to 'rt'
  + Abre myfile2.txt en modo texto (defecto) y lectura ®
* myfile3 = open('anotherfile.bin', 'rb')
  + Abre anotherfile.bin en modo binario (b) y lectura (r)
* myfile4 = open('myfile3.txt', 'w')
  + Abre myfile3.txt en modo texto (defecto) y escritura (w)
  + **OJO**: si tiene contenido se sobreescribirá y borrará todo lo que tenga previamente (w)
* myfile1 = open('myfile4.txt', 'r+') → read + write
  + Abre myfile4.txt en modo texto (defecto) y lectura y escritura (r+)
  + r+ no se cargará el contenido, lo respeta y se podrán añadir cosas, a diferencia de w+ que es lo mismo pero cargando todo
* myfile5 = open('anotherfile2.bin', 'ab')
  + Abre anotherfile2.bin en modo binario (b) y escritura desde el final (a)
* myfile1.close()
  + Cerramos el archivo guardando los cambios. Hay que cerrar los archivos SIEMPRE aunque solo sea para lectura. Es un agujero de seguridad y si se queda abierto nos exponemos a una posible pérdida/robo de información.

**Forma alternativa de abrir y cerrar un fichero:**



Ventajas: No requiere el close por lo que no se te olvidará ponerlo.

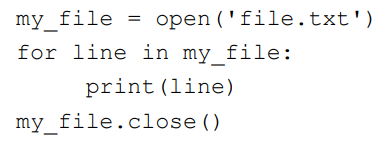
Desventajas: requiere que todas las instrucciones tengan que estar tabuladas, lo que dificulta la comprensión si hay muchas.

## Lectura de ficheros

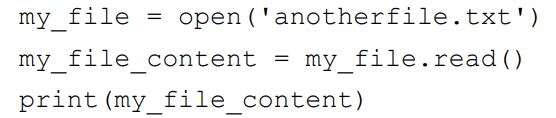
Usando los métodos read() y readline() en un fichero previamente abierto, se guardan en una variable.

* **readline()** → sobre ficheros de texto solamente.
* **read()** → lee datos de tanto ficheros binarios como de texto. Con ficheros binarios devuelve una lista de bytes.
  + Se especifica la cantidad que se quiere leer con un parámetro (opcional).
    - Para archivos binarios se ponen potencias los bytes de 2 (64, 128, 256, 512, 1024, …)
    - Si no se especifica lo lee entero
    - En ambos casos el puntero avanza tanto como se haya leído

Se puede iterar directamente sobre un fichero de texto abierto y mostrar línea a línea el contenido

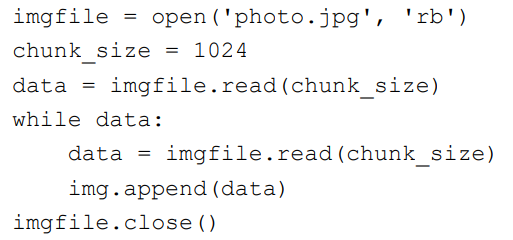


Todo el contenido de un fichero se puede guardar en una variable:



Hay que tener cuidado con la cantidad de memoria que ocupamos¡! puede ralentizar la máquina.

**Ejemplo de lectura de datos binarios:**



imgfile = open('photo.jpg', 'rb') → modo bin y lectura

chunk\_size = 1024 → tamaño de los trozos

data = imgfile.read(chunk\_size) → guarda en data los datos con el tamaño establecido

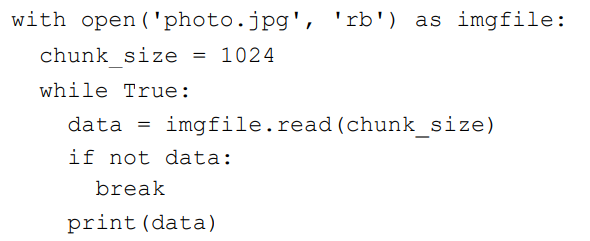
while data:

data = imgfile.read(chunk\_size)→ la operación de lectura devuelve un none al quedarse sin datos, así finaliza el bucle

img.append(data)

imgfile.close()

**Alternativa:**



with open('photo.jpg', 'rb') as imgfile: → no necesita el close

chunk\_size = 1024 → tamaño de los trozos

while True: → será true siempre a menos que se ponga el break del if cuando no se encuentran más datos en la lectura del fichero.

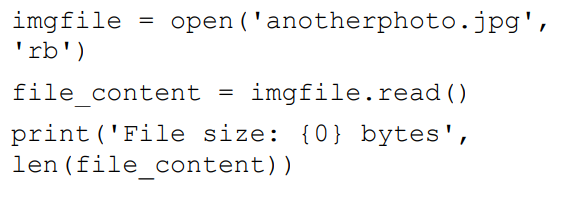
data = imgfile.read(chunk\_size)

if not data:

break

print(data)

**Otra alternativa para leer una foto:**



imgfile = open('anotherphoto.jpg', 'rb')

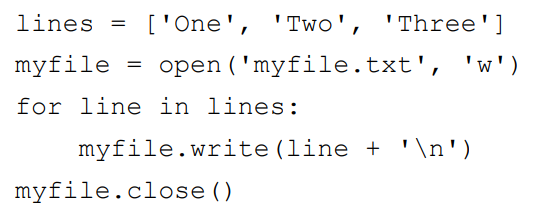
file\_content = imgfile.read() → trae la imagen entera a una variable

print('File size: {0} bytes', len(file\_content)) → el len(file\_content) devuelve el tamaño de la variable

## Escritura de ficheros

Se usa el método write(). Requiere un parámetro con lo que se quiere escribir. Si se hace un write sobre un fichero ya cerrado nos devolverá una excepción de entrada/salida, también pasa al intentar leerlo si está cerrado.

Ejemplo:



lines = ['One', 'Two', 'Three'] → lista con tres valores, son los que escribiremos en el fichero

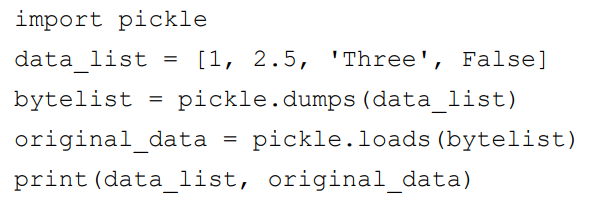
myfile = open('myfile.txt', 'w') → abro el archivo en modo texto y escritura (borrando lo que tenía si ya existía, si no existe lo crea como si fuera nuevo)

for line in lines:

myfile.write(line + '\n') → ponemos el valor line (cada valor de la lista) como parámetro y concatenamos \n para que haya un salto de línea después de cada valor

myfile.close() → cerrar archivo

**Extra: en el módulo sys hay alguna herramienta que comprueba si existe el fichero. Buscarla**

Ejemplo con datos binarios (conversión):  


import pickle → módulo que convierte los datos a binario y viceversa

data\_list = [1, 2.5, 'Three', False] → lista de valores a introducir

bytelist = pickle.**dumps**(data\_list) → convierte a binario

original\_data = pickle.**loads**(bytelist) → convierte a datos legibles

print(data\_list, original\_data)

## Actualización de datos en ficheros

(No lo vamos a usar mucho, solo tiene que sonarnos y saber usarlo)

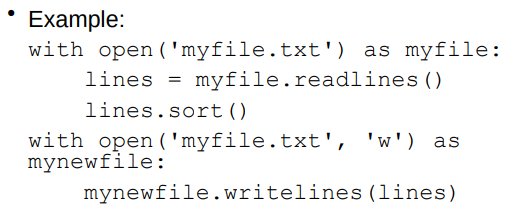
Se combinan operaciones de borrar y escribir, tanto para líneas como para bytes

Adicionalmente, se pueden usar métodos sobre el manejador de archivo:

* tell() → indica la posición actual en el fichero
* seek(offset,whence) → posiciona el puntero donde se indique (whence takes the values 0, 1, 2 = start, current, end)
* truncate(size) → corta el fichero donde se indique

Para actualizar archivos pequeños su contenido puede ser leído, después manipulado en memoria y finalmente escribirlo de nuevo.

**Ejemplo**: leemos las líneas y las guardamos en una lista, ordenamos las líneas alfabéticamente con un método y por último abrimos de nuevo el fichero pero en modo escritura, desechando el contenido que tuviese y escribimos el contenido ordenado.



**Ejemplo:** Borramos la 5a y 8a línea del fichero. Abrimos solo una vez, en modo lectura y escritura con la r para que no se borre el contenido. Leemos el contenido y lo guardamos en una variable. Usamos el método seek(0) sobre el manejador de fichero para mover el puntero al principio del fichero (...)

