

# INTRODUCCIÓN A LA PROGRAMACIÓN

**Archivos.**



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Un archivo es un **conjunto de datos identificado con un nombre** que puede ser almacenado de manera **permanente** en el directorio de un **dispositivo**, como puede ser el disco duro de un ordenador, un CD, una memoria USB o un espacio en la nube.



disco duro



DVD



CD



pendrive



tarjeta SD



Memory Stick



disco duro portátil



Disquete

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Existen dos formas básicas de acceder a un archivo:

- **archivo de texto**, que procesaremos **línea por línea**.
- **archivo binario**, que procesaremos **byte por byte**.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

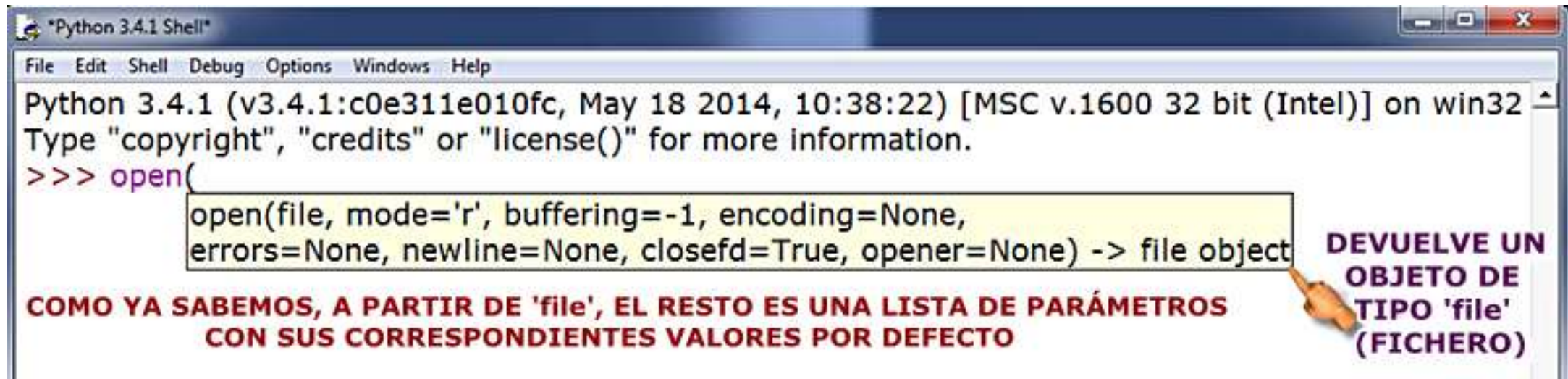
Sobre un archivo podemos realizar las siguientes operaciones: **apertura, cierre, escritura, lectura y desplazamiento**. Para realizarlas, no es necesario importar ninguna biblioteca, puesto que en la biblioteca estándar de Python está incluido todo lo necesario para trabajar con archivos.

El tipo de dato fichero en Python es **file**. Para asignar a una variable un valor de tipo file, solo es necesario recurrir a la función integrada **open()**, la cuál está destinada a la apertura de un archivo.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

La función **open()** cuenta con un argumento obligatorio, file, y siete argumentos opcionales, aunque como la mayoría de ellos ofrecen valores por defecto, no hay que especificarlos siempre.



The screenshot shows a Python 3.4.1 Shell window with the following content:

```
Python 3.4.1 (v3.4.1:c0e311e010fc, May 18 2014, 10:38:22) [MSC v.1600 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> open(
```

A yellow box highlights the function signature: `open(file, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None) -> file object`. A hand icon points from the text "DEVUELVE UN OBJETO DE TIPO 'file' (FICHERO)" to the end of the signature.

COMO YA SABEMOS, A PARTIR DE 'file', EL RESTO ES UNA LISTA DE PARÁMETROS CON SUS CORRESPONDIENTES VALORES POR DEFECTO

DEVUELVE UN OBJETO DE TIPO 'file' (FICHERO)

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Los que realmente nos interesan, para su uso más básico y habitual, son:

- ✓ **file** donde pasamos una cadena de texto, con la ruta (path) y el nombre del fichero y su extensión correspondiente (normalmente, .txt, para trabajar con textos planos).
- ✓ **mode** que especifica el modo de apertura del fichero. Por defecto se abrirá en modo lectura (r).
- ✓ **encoding** que especifica la codificación del archivo (deberíamos dar siempre el valor "utf-8"). Por defecto se utiliza la codificación del sistema.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Cada vez que abrimos un archivo estamos creando un **puntero**, el cuál se **posicionará** dentro del archivo en un lugar determinado (al comienzo o al final) y este puntero podrá **moverse** dentro de ese archivo, eligiendo su nueva posición, mediante el número de byte correspondiente.

Este puntero, se creará dependiendo del modo de apertura, el cuál será indicado en el segundo parámetro de la función **open()**.



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Entre los modos de apertura posibles, podemos encontrar los siguientes:

Indicador	Modo de apertura	Ubicación del puntero
<b>r</b>	Solo lectura	Al inicio del archivo
<b>rb</b>	Solo lectura en modo binario	Al inicio del archivo
<b>r+</b>	Lectura y escritura	Al inicio del archivo
<b>rb+</b>	Lectura y escritura en modo binario	Al inicio del archivo
<b>w</b>	Solo escritura. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
<b>wb</b>	Solo escritura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

<b>w+</b>	Escritura y lectura. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
<b>wb+</b>	Escritura y lectura en modo binario. Sobreescribe el archivo si existe. Crea el archivo si no existe.	Al inicio del archivo
<b>a</b>	Añadido (agregar contenido). Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.
<b>ab</b>	Añadido en modo binario (agregar contenido). Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.
<b>a+</b>	Añadido (agregar contenido) y lectura. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.
<b>ab+</b>	Añadido (agregar contenido) y lectura en modo binario. Crea el archivo si éste no existe.	Si el archivo existe, al final de éste. Si el archivo no existe, al comienzo.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Si abrimos un fichero y no especificamos su modo de apertura, por defecto se abre en **modo lectura** (r).

```
f = open("archivo.txt")
```

Si el archivo especificado **no existe**, el sistema lanza un error ***FileNotFoundError***.

Si lo que queremos es crear ese archivo para empezar a trabajar con él, debemos especificar el **modo escritura**(w)

```
f = open("archivo.txt", "w")
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Una vez que el **archivo está abierto**, devuelve un objeto de tipo **file** que nos permite obtener **información** sobre él accediendo a sus **propiedades**.

Se pueden acceder a las siguientes propiedades del objeto file:

- **closed**: retorna verdadero si el archivo se ha cerrado. De lo contrario, falso.
- **mode**: retorna el modo de apertura.
- **name**: retorna el nombre del archivo
- **encoding**: retorna la codificación de caracteres de un archivo de texto

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

```
archivo = open("archivo.txt", "r+")
contenido = archivo.read()
nombre = archivo.name
modo = archivo.mode
codificacion = archivo.encoding
archivo.close()
if archivo.closed:
    print "El archivo se ha cerrado correctamente"
else:
    print "El archivo permanece abierto"
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

Una vez terminemos de trabajar con el archivo debemos cerrarlo utilizando el método **file.close()** para que se guarden correctamente las modificaciones realizadas y para liberar los recursos del sistema utilizados para trabajar con él (como los búfer de lectura y escritura).

```
archivo.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

El objeto file, entre sus **métodos** más frecuentes, dispone de los siguientes:

Método	Descripción	Uso
<code>seek(byte)</code>	Mueve el puntero hacia el byte indicado	<pre>archivo = open("remeras.txt", "r") contenido = archivo.read() # el puntero queda # al final del documento archivo.seek(0)</pre>
<code>read([bytes])</code>	Lee todo el contenido de un archivo. Si se le pasa la longitud de bytes, leerá solo el contenido hasta la longitud indicada.	<pre>archivo = open("remeras.txt", "r") contenido = archivo.read() print contenido</pre>
<code>readline([bytes])</code>	Lee una línea del archivo.	<pre>archivo = open("remeras.txt", "r") linea1 = archivo.readline() print linea1</pre>
<code>readlines()</code>	Lee todas las líneas de un archivo	<pre>archivo = open("remeras.txt", "r") for linea in archivo.readlines():     print linea</pre>
<code>tell()</code>	Retorna la posición actual del puntero	<pre>archivo = open("remeras.txt", "r") linea1 = archivo.readline() mas = archivo.read(archivo.tell() * 2) if archivo.tell() &gt; 50:     archivo.seek(50)</pre>

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

<b>write(cadena)</b>	Escribe <i>cadena</i> dentro del archivo	<pre>archivo = open("remeras.txt", "r+") contenido = archivo.read() final_de_archivo = archivo.tell() <b>archivo.write('Nueva línea')</b> archivo.seek(final_de_archivo) nuevo_contenido = archivo.read() print nuevo_contenido # Nueva línea</pre>
<b>writelines(secuencia)</b>	Secuencia será cualquier iterable cuyos elementos serán escritos uno por línea	<pre>archivo = open("remeras.txt", "r+") contenido = archivo.read() final_de_archivo = archivo.tell() <b>lista = ['Línea 1\n', 'Línea 2']</b> <b>archivo.writelines(lista)</b> archivo.seek(final_de_archivo) print archivo.readline() # Línea 1 print archivo.readline() # Línea 2</pre>
<b>close()</b>	Cierra un archivo	<pre>archivo = open("remeras.txt", "r") contenido = archivo.read() <b>archivo.close()</b> print contenido</pre>



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Escritura de archivos

Para la escritura de archivos se utilizan los métodos **write** y **writelines**.

**file.write(*cad*)** escribe la cadena *cad* en el archivo y devuelve el número de caracteres escritos. El argumento *cad* debe ser una **cadena de caracteres**, si se va a escribir en un **archivo de texto** o una **cadena de bytes**, si se va a escribir en un **archivo binario**. Este método no añade el carácter de nueva línea (`\n`) al final de la cadena, por lo que, si fuera necesario, habría que añadirlo en la propia cadena.

```
archivo.write(cadena)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Escritura de archivos

cadena1 = 'Datos'

cadena2 = 'Secretos'

```
# Abre archivo para escribir  
archivo = open('datos1.txt','w')
```

```
# Escribe cadena1 añadiendo salto de línea  
archivo.write(cadena1 + '\n')
```

```
# Escribe cadena2 en archivo  
archivo.write(cadena2)
```

```
# cierra archivo  
archivo.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Escritura de archivos

**file.writelines(iterable\_cad)** Escribe la secuencias de cadenas contenidas en el iterable. Tampoco añade el carácter de nueva línea(\n) al final de cada cadena del iterable, por lo que, si fuera necesario, habría que especificarlo en cada instrucción de escritura.

```
archivo.writelines(lista_de_cadenas)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Escritura de archivos

```
lista = ['lunes', 'martes', 'miercoles', 'jueves', 'viernes']
```

```
# Abre archivo en modo escritura
```

```
archivo = open('datos2.txt','w')
```

```
# Escribe toda la lista en el archivo
```

```
archivo.writelines(lista)
```

```
# Cierra archivo
```

```
archivo.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Lectura de archivos

Para la lectura de archivos se utilizan los métodos **read**, **readline** y **realines**.

**file.read(*tamaño*)** devuelve una cadena con el contenido del archivo o bien el contenido de los primeros n bytes, si se especifica el tamaño máximo a leer. El argumento *tamaño* es opcional. Se trata de un entero que especifica el número de bytes a leer. Si no se indica nada, o si se especifica un valor negativo o None, devuelve el contenido entero del archivo. En caso de que se haya alcanzado el final del archivo devuelve una cadena vacía.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Lectura de archivos

```
# Abre archivo en modo lectura  
archivo = open('archivo.txt','r')
```

```
# Lee los 9 primeros bytes  
cadena1 = archivo.read(9)
```

```
# Lee la información restante  
cadena2 = archivo.read()
```

```
#Muestra la primera lectura  
print(cadena1)
```

```
# Muestra la segunda lectura  
print(cadena2)
```

```
# Cierra el archivo  
archivo.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Lectura de archivos

**file.readline()** Permite leer una sola línea del archivo, sin eliminar el carácter de nueva línea (\n) final, y la devuelve como una cadena. Si se ha alcanzado el final del archivo devuelve una cadena vacía.

```
# Abre archivo en modo lectura
```

```
archivo = open('archivo.txt','r')
```

```
linea = archivo.readline() # lee línea
```

```
print(linea) # Muestra la línea leída
```

```
# Cierra el archivo
```

```
archivo.close()
```



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Lectura de archivos

**file.readlines()** Lee el contenido completo del archivo y lo devuelve como una lista formada por todas las líneas del mismo. No elimina el carácter de nueva línea (\n) al final de cada cadena.

```
# Abre archivo en modo lectura
```

```
archivo = open('archivo.txt','r')
```

```
# Lee todas la líneas y asigna a lista
```

```
lista = archivo.readlines()
```

```
# Cierra el archivo
```

```
archivo.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Mover el puntero de lectura/escritura

Para saber en qué posición se encuentra el puntero de un archivo, o para desplazarse a una posición determinada. Python proporciona los métodos **tell** y **seek**.

**file.tell()** devuelve un entero que, en bytes, representa la posición en la que se encuentra el puntero del fichero. Este valor se calcula tomando como referencia el principio del fichero.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Mover el puntero de lectura/escritura

**`file.seek(desplazamiento, inicio)`** Permite desplazar el puntero del archivo los bytes indicados en el argumento *desplazamiento* que toma como parámetro un número positivo o negativo. También es posible utilizar un segundo parámetro para indicar desde dónde queremos que se haga el desplazamiento: 0 indicará que el desplazamiento se refiere al principio del fichero (comportamiento por defecto), 1 se refiere a la posición actual, y 2, al final del fichero. Si se indica un valor diferente, lanza una excepción **ValueError**. Además de desplazar el puntero, devuelve un entero asociado al byte al que se ha desplazado.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Mover el puntero de lectura/escritura

```
# Abre archivo en modo lectura  
archivo = open('datos2.txt','r')
```

```
# Muestra posición del puntero  
print(archivo.tell())
```

```
# Mueve puntero al quinto byte  
archivo.seek(5)
```

```
# Cierra archivo  
archivo.close()
```

```
# lee los siguientes 5 bytes  
cadena1 = archivo.read(5)
```

```
# Muestra cadena  
print(cadena1)
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

### Cerrando archivos de forma automática

Desde la versión 2.5, Python incorpora una manera “elegante” de trabajar con archivos de forma tal, que se cierran de forma automática sin necesidad de invocar al método **close()**.

Se trata de un bloque with:

**with** EXPRESIÓN **as** VARIABLE:  
BLOQUE DE INSTRUCCIONES

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

En el caso de los ficheros, la expresión es una llamada a la función **open()** y la variable es la **conexión con el fichero**:

```
with open("FICHERO") as fichero:  
    BLOQUE DE INSTRUCCIONES
```

La función open puede tener varios argumentos. Los más importantes son

```
with open("FICHERO", mode="MOD0", encoding="CODIFICACIÓN") as fichero:  
    BLOQUE DE INSTRUCCIONES
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

```
with open('mi_fichero', 'w') as f:  
    f.write('Hola mundo\n')
```



# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

El ejemplo siguiente abre un fichero y lo copia en otro:

```
f = open("origen.txt")
```

```
g = open("destino.txt", "w")
```

```
for linea in f:
```

```
    g.write(linea)
```

```
g.close()
```

```
f.close()
```

El bucle "**for linea in f**" permite recorrer un fichero de texto, obteniendo una línea cada vez.

# INTRODUCCIÓN A LA PROGRAMACIÓN

## Archivos.

El siguiente ejemplo hace la copia del fichero leyendo con **readline()** en un bucle hasta fin de fichero. Cuando lleguemos a final de fichero nos devolverá una línea vacía.

```
f = open("origen.txt")
g = open("destino.txt", "w")
linea = f.readline()
while linea != "":
    g.write(linea)
    linea = f.readline()
g.close()
f.close()
```

# INTRODUCCIÓN A LA PROGRAMACIÓN

Archivos.

*“Fin del tema”*

