



# BASECAMP

Ciencia de Datos

## Obtención y Preparación de Datos

---

### Objetivo de la jornada

---

- Identificar los distintos métodos para la obtención de datos de diversas fuentes utilizando librerías de python
- Recuperar información desde archivos de texto CSV y Excel utilizando librerías de python para su posterior procesamiento.

### Obtención de datos desde archivos

#### Leyendo y escribiendo archivos

La función **open()** retorna un **file object**, y se usa normalmente con dos argumentos: **open(nombre\_de\_archivo, modo)**.

```
f = open('workfile', 'w')
```

El primer argumento es una cadena que contiene el nombre del fichero. El segundo argumento es otra cadena que contiene unos pocos caracteres describiendo la forma en que el fichero será usado, mode puede ser 'r' cuando el fichero solo se leerá, 'w' para sólo escritura (un fichero existente con el mismo nombre se borrará) y 'a' abre el fichero para agregar.; cualquier dato que se escribe en el fichero se añade automáticamente al final. 'r+' abre el fichero tanto para lectura como para escritura. El argumento mode es opcional; se asume que se usará 'r' si se omite.

Normalmente, los ficheros se abren en modo texto, significa que lees y escribes caracteres desde y hacia el fichero, el cual se codifica con una codificación específica. Si no se especifica la codificación el valor por defecto depende de la plataforma. 'b' agregado al modo abre el fichero en modo binario: y los datos se leerán y escribirán en forma de objetos de bytes. Este modo debería usarse en todos los ficheros que no contienen texto.

Cuando se lee en modo texto, por defecto se convierten los fines de líneas que son específicos a las plataformas (`\n` en Unix, `\r\n` en Windows) a solamente `\n`. Cuando se escribe en modo texto, por defecto se convierten los `\n` a los finales de línea específicos de la plataforma. Este cambio automático está bien para archivos de texto, pero corromperá datos binarios como los de archivos **JPEG** o **EXE**. Asegúrate de usar modo binario cuando leas y escribas dichos archivos.

Es una buena práctica usar la declaración `with` cuando manejamos objetos archivo. Tiene la ventaja que el archivo es cerrado apropiadamente luego de que el bloque termina, incluso si se generó una excepción. También es mucho más corto que escribir los equivalentes bloques **try-finally**

```
with open('workfile') as f:
    read_data = f.read()
# Se puede revisar el archivo ha sido cerrado automáticamente
f.closed

True
```

Si no está utilizando la palabra clave `with`, debe llamar a `f.close()` para cerrar el archivo y liberar inmediatamente los recursos del sistema que utiliza.

## Método de los objetos Archivos

Para leer el contenido de un archivo utiliza **`f.read(size)`**, el cual lee alguna cantidad de datos y los retorna como una cadena de (en modo texto) o un objeto de bytes (en modo binario). *size* es un argumento numérico opcional. Cuando se omite *size* o es negativo, el contenido entero del archivo será leído y retornado; es tu problema si el archivo es el doble de grande que la memoria de tu máquina. De otra manera, como

máximo *size* caracteres (en modo texto) o *size* bytes (en modo binario) son leídos y retornados. Si se alcanzó el fin del archivo, **f.read()** retornará una cadena vacía ('').

```
f.read()

'This is the entire file. \n'

f.read()

''
```

**f.readline()** lee una sola línea del archivo; el carácter de fin de línea (\n) se deja al final de la cadena, y sólo se omite en la última línea del archivo si el mismo no termina en un fin de línea. Esto hace que el valor de retorno no sea ambiguo; si f.readline() retorna una cadena vacía, es que se alcanzó el fin del archivo, mientras que una línea en blanco es representada por '\n', una cadena conteniendo sólo un único fin de línea.

```
f.readline()

'This is the first line of the file. \n'

f.readline()

'Second line of the file. \n'

f.readline()

''
```

Para leer líneas de un archivo, podés iterar sobre el objeto archivo. Esto es eficiente en memoria, rápido, y conduce a un código más simple:

```
for line in f:
    print(line, end = '')
```

This is the first line of the file  
Second line of the file

Si quieres leer todas las líneas de un archivo en una lista también puedes usar **list(f)** o **f.readlines()**.

**f.write(cadena)** escribe el contenido de la cadena al archivo, retornando la cantidad de caracteres escritos.

```
f.write('This is a test \n')
15
```

Otros tipos de objetos necesitan ser convertidos – tanto a una cadena (en modo texto) o a un objeto de bytes (en modo binario) – antes de escribirlos:

```
value = ('The answer', 42)
s = str(value) # convierte la tupla en string
f.write(s)
18
```

**f.tell()** retorna un entero que indica la posición actual en el archivo representada como número de bytes desde el comienzo del archivo en modo binario y un número opaco en modo texto.

Para cambiar la posición del objeto archivo, utiliza **f.seek(offset, whence)**. La posición es calculada agregando el **offset** a un punto de referencia; el punto de referencia se selecciona del argumento **whence**. Un valor **whence** de 0 mide desde el comienzo del archivo, 1 usa la posición actual del archivo, y 2 usa el fin del archivo

como punto de referencia. **whence** puede omitirse, el valor por defecto es 0, usando el comienzo del archivo como punto de referencia.

```
f = open('workfile', 'rb+')
f.write(b'0123456789abcdef')

16

f.seek(5)

5

f.read(1)

b'5'

f.seek(-3, 2)

13

f.read(1)

b'd'
```

En los archivos de texto (aquellos que se abrieron sin una **b** en el modo), se permiten solamente desplazamientos con **seek** relativos al comienzo (con la excepción de ir justo al final con **seek(0, 2)**) y los únicos valores de desplazamiento válidos son aquellos retornados por **f.tell()**, o cero. Cualquier otro valor de desplazamiento produce un comportamiento indefinido.

Los objetos archivo tienen algunos métodos más, como **isatty()** y **truncate()** que son usados menos frecuentemente; consultá la Referencia de la Biblioteca para una guía completa sobre los objetos archivo.

## Manipulación de archivos CSV

Un archivo CSV (valores separados por comas) permite que los datos sean guardados en una estructura tabular con una extensión .csv. Los archivos CSV se usan de manera extensiva en aplicaciones de comercio electrónico porque son considerados muy fáciles de procesar. Algunas de las áreas en donde han sido usados incluyen:

- Importar y exportar datos de clientes
- Importar y exportar productos
- Exportar órdenes
- Exportar reportes analíticos de comercio electrónico

## Módulos de lectura y Escritura

El módulo CSV tiene varias funciones y clases disponibles para leer y escribir CSVs, y estas incluyen:

- Función `csv.reader`
- Función `csv.writer`
- Clase `csv.Dictwriter`
- Clase `csv.DictReader`

*csv.reader*

El módulo **csv.reader** toma los siguientes parámetros:

- **csvfile**: Este es usualmente un objeto el cuál soporta el protocolo iterador y usualmente devuelve una cadena cada vez que su método **\_\_next\_\_()** es llamado.

- **dialect='excel'**: Un parámetro opcional usado para definir un conjunto de parámetros específicos a un dialecto CSV en particular.
- **fmtparams**: Un parámetro opcional que puede ser usado para sobrescribir parámetros de formato existentes.

Aquí está un ejemplo de cómo usar el módulo `csv.reader`.

```
import csv

with open('example.csv', newline='') as File:
    reader = csv.reader(File)
    for row in reader:
        print(row)
```

### *csv.writer*

Este módulo es similar al módulo **csv.reader** y es usado para escribir datos a un CSV.

Este toma tres parámetros:

- **csvfile**: Este puede ser cualquier objeto con un método **write()**.
- **dialect='excel'**: Un parámetro opcional usado para definir un conjunto de parámetros específicos a un CSV en particular.
- **fmtparam**: Un parámetro opcional que puede ser usado para sobrescribir parámetros de formato existentes.



### *DictReader*

Esta crea un objeto el cuál mapea la información leída a un diccionario cuyas llaves están dadas por el parámetro **fieldnames**. Este parámetro es opcional, pero cuando no se especifica en el archivo, la primera fila de datos se vuelve las llaves del diccionario.

```
import csv

with open('name.csv', newline='') as csvfile:
    reader = csv.DictReader(csvfile)
    for row in reader:
        print(row['first_name'], row['last_name'])
```

### *DictWriter*

Esta clase es similar a la clase DictWriter y hace lo contrario, que es escribir datos a un archivo CSV. La clase es definida como **csv.DictWriter(csvfile, fieldnames, restval="", extrasaction='raise', dialect='excel', \*args, \*\*kwds)**

El parámetro **fieldnames** define la secuencia de llaves que identifican el orden en el cuál los valores en el diccionario son escritos al archivo CSV. A diferencia de DictReader, esta llave no es opcional y debe ser definida para evitar errores cuando se escribe a un CSV.

## Dialectos y formatos

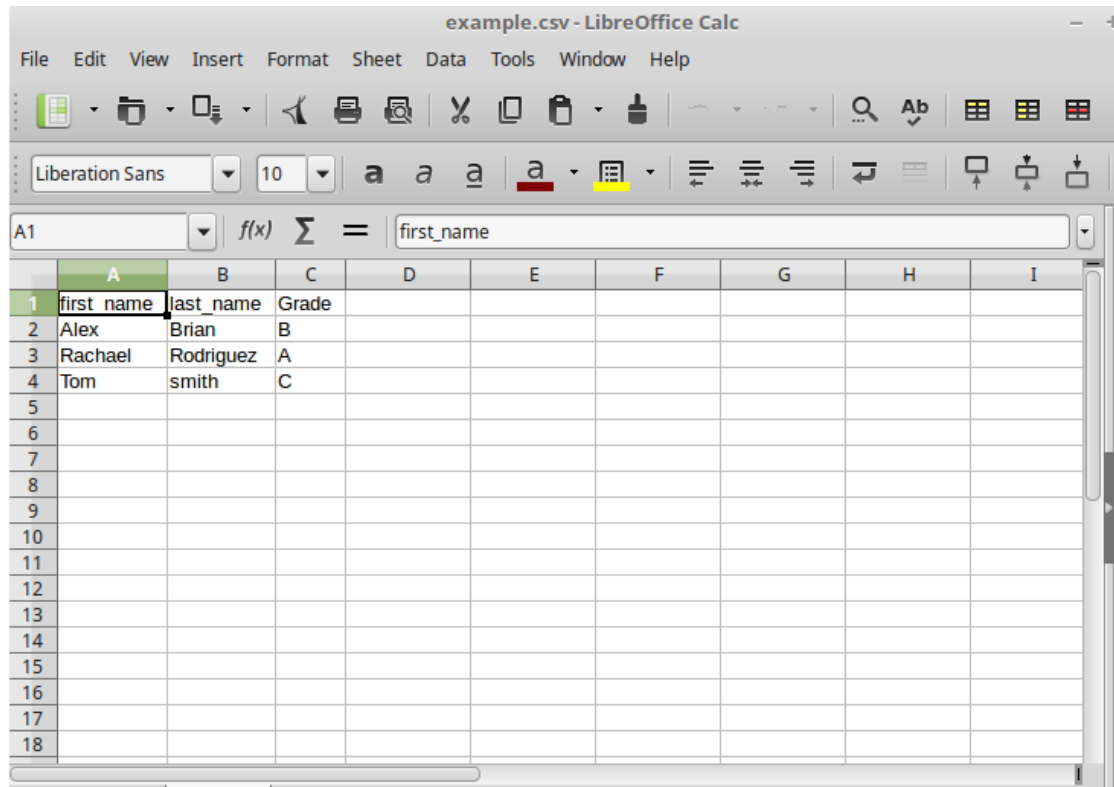
Un dialecto es una clase ayudante usada para definir los parámetros para una instancia **reader** o **writer** específica. Los parámetros de dialecto y formato necesitan ser declarados cuando se realiza una función lectora o writer.

Hay varios atributos los cuáles están soportados por un dialecto:

- **delimiter:** Una cadena usada para separar campos. Por defecto es `','`.
- **double quote:** Controla cómo las instancias de *quotechar* que aparecen dentro de un campo deberían ser citadas. Puede ser Verdadero o Falso.
- **escapechar:** Una cadena usada por el escritor para escapar el *delimitador* si *quoting* está establecido a `QUOTE_NONE`.
- **lineterminator:** Una cadena usada para terminar líneas producidas por el writer. Por defecto es `'\r\n'`.
- **quotechar:** Una cadena usada para citar campos conteniendo caracteres especiales. Por defecto es `'"'`.
- **skipinitialspace:** Si se establece a `True`, cualquier espacio en blanco después del *delimitador* es ignorado inmediatamente.
- **strict:** Si se establece a `True`, levanta una excepción `Error` en mala entrada CSV.
- **quoting:** Controla cuando deberían ser generadas las citas cuando se lee o escribe a un CSV.

## Leyendo un archivos CSV - Ejemplo

Crea tu archivo CSV y guárdalo como ejemplo.csv. Asegura que tiene la extensión .csv y llena algunos datos. Aquí tenemos nuestro archivo CSV el cual contiene los nombres de los estudiantes y sus calificaciones.



	A	B	C	D	E	F	G	H	I
1	first_name	last_name	Grade						
2	Alex	Brian	B						
3	Rachael	Rodriguez	A						
4	Tom	smith	C						
5									
6									
7									
8									
9									
10									
11									
12									
13									
14									
15									
16									
17									
18									

Abajo está el código para leer los datos en nuestro CSV usando tanto la función **csv.reader** como la clase **csv.DictReader**.

### *csv.reader*

```
import csv

with open('example.csv') as File:
    reader = csv.reader(File, delimiter = ',', quotechar =
    ',', quoting = csv.QUOTE_MINIMAL)
    for row in reader:
        print(row)
```

En el código de arriba, importamos el módulo CSV y después abrimos nuestro archivo CSV como File. Entonces definimos el objeto lector y usamos el método **csv.reader** para extraer los datos al objeto. Entonces iteramos sobre el objeto **reader** y recuperamos cada fila de nuestros datos.

Mostramos los datos leídos imprimiendo sus contenidos a la consola. También hemos especificado los parámetros requeridos tales como delimiter, quotechar, y quoting.

### **Salida**

```
['first_name', 'last_name', 'Grade']
['Alex', 'Brian', 'A']
['Rachael', 'Rodriguez', 'A']
['Tom', 'smith', 'C']
```

Como mencionamos arriba, **DictWriter** nos permite leer un archivo CSV mapeando datos a un diccionario en vez de cadenas como en el caso del módulo **csv.reader**. Aunque el **fieldname** es un parámetro opcional, es importante siempre tener etiquetadas tus columnas para legibilidad.

Aquí está cómo leer un CSV usando la clase DictWriter.

```
import csv

results = []
with open('example.csv') as File:
    reader = csv.DictReader(File)
    for row in reader:
        results.append(row)
    print(row)
```

Primero importamos el módulo **csv** e inicializamos una lista vacía **results** la cuál usaremos para almacenar los datos recuperados. Después definimos el objeto lector y usamos el método **csv.DictReader** para extraer los datos en el objeto. Entonces iteramos el objeto **reader** y recuperamos cada fila de nuestros datos.

Finalmente, adjuntamos cada fila a la lista de resultados e imprimimos los contenidos a la consola.

### Salida

```
[{'Grade': 'B', 'first_name': 'Alex', 'last_name': 'Brian'},
{'Grade': 'A', 'first_name': 'Rachel',
'last_name': 'Rodriguez'}
{'Grade': 'C', 'first_name': 'Tom', 'last_name': 'smith'}]
```

Como puedes ver arriba, usar la clase DictReader es mejor porque da nuestros datos en un formato de diccionario con el cual es más sencillo trabajar.

### Escribiendo un archivos CSV - Ejemplo

Veamos ahora cómo escribir datos a un archivo CSV usando la función **csv.writer** y la clase **csv.Dictwriter** discutida al inicio de este tutorial.

*csv.writer*

El código de abajo escribe los datos definidos al archivo example2.csv.

```
import csv

myData = [['first_name', 'second_name', 'Grade'],
          ['Alex', 'Brian', 'A'],
          ['Tom', 'Smith', 'B']]

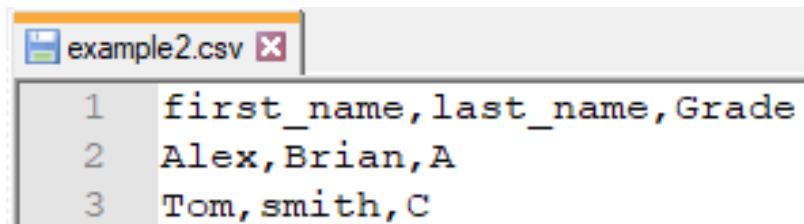
myFile = open('example2.csv', 'W')

with myFile:
    write = csv.writer(myFile)
    write = writerows(myData)

print("Writing complete")
```

Primero importamos el módulo csv, y la función **writer()** creará un objeto apto para escritura. Para iterar los datos sobre las filas, necesitaremos usar la función **writerows()**.

Aquí está nuestro CSV con los datos que le hemos escrito.



1	first_name,last_name,Grade
2	Alex,Brian,A
3	Tom,smith,C

*csv.Dictwriter*

Escribamos los siguientes datos a un CSV.

```
data = [{'Grade': 'B', 'first_name': 'Alex', 'last_name': 'Brian'},  
{ 'Grade': 'A', 'first_name': 'Rachael', 'last_name': 'Rodriguez'},  
{ 'Grade': 'C', 'first_name': 'Tom', 'last_name': 'smith'},  
{ 'Grade': 'B', 'first_name': 'Jane', 'last_name': 'Oscar'},  
{ 'Grade': 'A', 'first_name': 'Kennzy', 'last_name': 'Tim'}]
```

El código es como se muestra abajo.

```
import csv  
  
with open('example4.csv', 'W') as csvfile  
    fieldnames = ['first_name', 'last_name', 'Grade']  
    write = csv.DictWrite(csvfile, fieldnames = fieldnames)  
  
    writer.writeheader()  
    writer.writerow({'Grade': 'B', 'first_name': 'Alex',  
'last_name': 'Brian'})  
    writer.writerow({'Grade': 'A', 'first_name': 'Rachel',  
'last_name': 'Rodriguez'})  
    writer.writerow({'Grade': 'C', 'first_name': 'Tom',  
'last_name': 'smith'})  
    writer.writerow({'Grade': 'B', 'first_name': 'Jane',  
'last_name': 'Oscar'})  
    writer.writerow({'Grade': 'A', 'first_name': 'Kennzy',  
'last_name': 'Tim'})
```

```
print("Writing complete")
```

Primero definimos los **fieldnames**, los cuales representarán los encabezados de cada columna en el archivo CSV. El método **writerow()** escribirá a una fila a la vez. Si quieres escribir todos los datos de una vez, usarás el método **writerows()**.

Aquí está cómo escribir a todas las filas de una vez.

```
import csv

with open('example5.csv', 'W') as csvfile:
    fieldnames = ['first_name', 'last_name', 'Grade']
    writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

    writer.writeheader()
    writer.writerows([
        {'Grade': 'B', 'first_name': 'Alex', 'last_name': 'Brian'},
        {'Grade': 'A', 'first_name': 'Rachel', 'last_name': 'Rodriguez'},
        {'Grade': 'C', 'first_name': 'Tom', 'last_name': 'Smith'},
        {'Grade': 'B', 'first_name': 'Jane', 'last_name': 'Oscar'},
        {'Grade': 'A', 'first_name': 'Kennzy', 'last_name': 'Tim'}
    ])

print("Writing complete")
```

## Manipulación de archivos Excel

**Microsoft Excel** es uno de los programas más utilizados para la **visualización y análisis de datos en la empresa**. La omnipresencia de este programa hace que muchos **usuarios se decanten por el formato xlsx** (o xls) para exportar sus conjuntos de datos. Por esto, saber leer y escribir archivos Excel en Python es clave para trabajar de forma óptima en muchos entornos. A pesar de que este no sea el formato favorito de



muchos científicos de datos. Ya que estos generalmente prefieren trabajar con archivos en formato CSV.

### Librería xlrd

xlrd es una biblioteca para leer datos y formatear información de archivos de Excel en **.xls** formato histórico .

```
import xlrd

book = xlrd.open_workbook("myfile.xls")
print("El número de la hoja de trabajo es
{0}".format(book.nsheets))
print("Nombre(s) de hoja(s) de trabajo(s): {0}
{0}".format(book.nsheets))

sh = book.sheet_by_index(0)

print("{0} {1} {2}".format(sh.cell_value(rowx = 29, colx =
3)))
for rx in range(sh.nrows):
    print(sh.row(rx))
```

Desde la línea de comando, esto mostrará la primera, segunda y última fila de cada hoja en cada archivo:

```
python PYDIR/scripts/runxlrd.py 3rows *blah*.xls
```

## Lectura de Excel con xlrd

Es posible que necesitemos realizar tareas más complejas a la hora de leer archivos Excel y podemos usar xlrd. Vemos algunas de las posibilidades:

```
import xlrd

archivo = 'C:Users/vaquerizo/Documents/ejemplo.xlsx'
wb = xlrd.open_workbook(archivo)

hoja = wb.sheet_by_index(0)
print(hoja.nrows)
print(hoja.ncols)
print(hoja.cell_value(0, 0))
```

**open\_workbook** nos abre el Excel para trabajar con él. Seleccionamos hojas por índice (empezando por el 0) y con la hoja seleccionada podemos ver el número de filas (**nrows**) o columnas (**ncols**). Seleccionar una celda lo hacemos con **cell\_value** mediante índices (empezando por el 0). Otras posibilidades:

```
archivo = 'C:Users/rvaquerizo/Documents/ejemplo.xlsx'
wb = xlrd.open_workbook(archivo)

hoja = wb.sheet_by_name('Hoja1')

for i in range(0, hoja.nrows):
    print(hoja.cell_value(i, 1))
```

Si por ejemplo deseamos saber las cabeceras, los nombres de las columnas:

```
archivo = 'C:Users/rvaquerizo/Documents/ejemplo.xlsx'
wb = xlrd.open_workbook(archivo)

hoja = wb.sheet_by_index(0)
nombres = hoja.row(0)
print(nombres)
```

Y mediante xlrd podemos crear dataframes de pandas con lo que es posible realizar lecturas de rangos:

```
archivo = 'C:Users/rvaquerizo/Documents/ejemplo.xlsx'
wb = xlrd.open_workbook(archivo)

hoja = wb.sheet_by_index(0)

# Creamos listas
filas = []
for fila in range(1, hoja.nrows):
    columnas = []
    for columna in range(0, 2):
        columnas.append(hoja.cell_value(fila, columna))
    filas.append(columnas)

import pandas as pd

df = pd.DataFrame(filas)
df.head()
```

## Escritura de Excel con xlwt

Use xlwt para escribir datos en Excel. **add\_sheet**, puede agregar un formulario en él, puede establecer el formato de la celda, como la fuente y el color.

```
import xlwt from datetime import datetime

style0 = xlwt.easyxf('font: name Times New Roman, color-index red, bold on', num_format_str='#,##0.00')
style1 = xlwt.easyxf(num_format_str='D-MM-YY')

wb = xlwt.WorkBook()
ws = wb.add_sheet('A Test Sheet')

ws.write(0, 0, 1234.56, style0)
ws.write(1, 0, datetime.now(), style1)
ws.write(2, 0, 1)
ws.write(2, 1, 1)
ws.write(2, 2, xlwt.Formula("A3+B3"))
wb.save('example.xls')
```

## Referencias

[1] Lectura de CSV con Python

<https://www.youtube.com/watch?v=qlgFO-fLXck>

[2] Leer CSV con Pandas

<https://pharos.sh/leer-y-escribir-archivos-csv-en-python-con-pandas/>

[3] Lectura y escritura de archivos Excel.

<https://programmerclick.com/article/9740120496/>