Librería csv

Los archivos CSV del inglés Comma Separated Values (Valores Separados por Coma), es un formato de archivo muy utilizado para la importación y exportación de hojas de cálculo y bases de datos.
Es un formato de archivo muy liviano (ya que es texto plano) y sencillo de trabajar. Es muy fácil representar tablas en este formato. Cada línea del archivo es una nueva fila de la tabla y cada valor separado por la coma es una nueva columna.

Cabe destacar que el formato CSV tiene diversos dialectos, por ejemplo, valores separados por comas, valores separados por punto y comas y valores separados por tabs.
Además también se define el tipo de comillas que se utilizan para las cadenas de texto.

Por esto es muy útil tener una librería que nos permita leer archivos CSV con diversos dialectos sin tener que ocuparnos de la implementación.

La librería csv forma parte de la biblioteca estándar de Python. Esta librería permite leer y escribir archivos CSV. Por ejemplo, permite al programador decir “Escribe estos datos en el formato preferido por Excel” o “Lee los datos desde este archivo que fue generado por Excel”, sin conocer los detalles precisos del formato CSV utilizado por Excel. Además el programador puede describir los formatos CSV soportados por otras aplicaciones o crearse sus propios formatos CSV para propósitos especiales.

Dialectos

Como mencionamos anteriormente el formato de archivo CSV soporta muchas variantes. Estas variantes se conocen como dialectos.

A continuación se presentan los dialectos predefinidos en la librería:

Nombre	Descripción
excel	La propiedades usuales de los archivos CSV generados por Excel
excel-tab	La propiedades usuales de los archivos delimitados por TAB generados por Excel.
unix	La propiedades usuales de los archivos CSV generados por los sistemas unix

Además la librería define las siguientes constantes:

Nombre	**Descripción**
QUOTE_ALL	Entrecomilla todos los campos.
QUOTE_MINIMAL	Entrecomilla solo los campos que tienen algún carácter especial, como por ejemplo, el delimitador, el carácter usado para las comillas o los usados para delimitar la línea.
QUOTE_NONNUMERIC	Entrecomilla todo campo no numérico.
QUOTE_NONE	No entrecomilla ningún campo.

El dialecto soporta los siguientes atributos, que hacen que el archivo se interprete de una manera u otra:

Nombre	**Descripción**
delimiter	Un carácter utilizado para separar los campos. Por defecto la coma (,)
doublequote	Controla cuando se ceben entrecomillar las instancias de quotechar en un campo. Cuando está en True el carácter se duplica. Cuando está en False se utiliza el *escapechar* como prefijo del *quotechar.* Por defecto esta en True.
escapechar	Un carácter utilizado para escapar el delimitador si *quoting* está definido en QUOTE_NONE y el *quotechar* si doublequote esta en False.
lineterminator	El string utilizado para terminar una línea. Por defecto '\r\n'
quotechar	Un carácter utilizado para entrecomillar los campos. Por defecto (")
quoting	Indica cuándo se debe entrecomillar. Los valores son los de las constantes QUOTE_*. El valor por defecto es QUOTE_MINIMAL.
skipinitialspace	Si está en True ignora el espacio inmediatamente posterior al delimitador. Por defecto esta en False.
strict	Cuando está en True lanza una excepción en el caso de ingresar un CSV mal formado. Por defecto esta en False.

Lectura de archivos CSV

Existen dos formas de leer archivos csv, una es utilizando el objeto **reader** y otra utilizando la clase **DictReader**. El objeto **reader** lee los datos en secuencias cada línea del archivo es una secuencia. En cambio la clase **DictReader** lee los datos en diccionarios, es decir, cada línea del archivo es un diccionario, donde la clave es el nombre de la columna (el valor de la primer fila o uno especificado por el usuario) y el valor es el valor leído en la fila actual.

El objeto reader

Para crear un objeto **reader** se le debe pasar a la función *reader* un objeto de tipo archivo o una lista. Adicionalmente se le puede pasar un dialecto y se pueden pasar parámetros para modificar uno o más valores del dialecto.

La función *reader* devuelve un objeto *reader* que es un iterable sobre cada una de las líneas del archivo.

A continuación se muestran algunos ejemplos:

```
import csv

with open('some.csv', newline='') as f:
   reader = csv.reader(f)
   for row in reader:
      print(row)

with open('passwd', newline='') as f:
   reader = csv.reader(f, delimiter=':', quoting=csv.QUOTE_NONE)
   for row in reader:
      print(row)

with open('example.csv', newline='') as csvfile:
   reader = csv.reader(csvfile, delimiter=' ', quotechar='|')
   for row in reader:
      print(', '.join(row))

csv.register_dialect('unixpwd', delimiter=':', quoting=csv.QUOTE_NONE)
with open('passwd', newline='') as f:
   reader = csv.reader(f, 'unixpwd')
```

La clase DictReader

La clase **DictReader** crea un objeto que opera de forma similar al objeto *reader* pero mapea la información de cada línea del archivo en un diccionario ordenado. Donde sus claves son dadas por el usuario en el parámetro *fieldnames*. Si el parámetro *fieldnames* no esta definido entonces se tomará los valores de la primer línea del archivo como las claves.

Al igual que con el objeto *reader* se puede indicar el dialecto y modificar uno o más valores del mismo.

A continuación se muestra un ejemplo:

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

Escritura de archivos CSV

Al igual que en la lectura de archivos CSV, la librería nos provee de dos formas de escribir un archivo CSV. Una utilizando el objeto **writer** y otra utilizando la clase **DictWriter**. El objeto **writer** escribe los datos en un archivo a partir de secuencias. En cambio la clase **DictWriter** lo hace a partir de diccionarios.

El objeto writer

Para crear un objeto **writer** se le debe pasar a la función *writer* un objeto de tipo archivo o cualquier objeto que sepa responder el mensaje *write*. Adicionalmente se le puede pasar un dialecto y se pueden pasar parámetros para modificar uno o más valores del dialecto.

La función *writer* devuelve un objeto **writer** que es responsable de convertir y guardar los datos que pasa el usuario en una cadena de texto delimitada sobre el objeto archivo dado.

A continuación se muestran algunos ejemplos:

```
import csv

someiterable = [[‘1’, ‘2’, ‘3’], [‘4’, ‘5’, ‘6’]]

with open('some.csv', 'w', newline='') as f:
   writer = csv.writer(f)
   writer.writerows(someiterable)

with open(example.csv', 'w', newline='') as csvfile:
   writer = csv.writer(csvfile, delimiter=' ',
                       quotechar='|', quoting=csv.QUOTE_MINIMAL)
   writer.writerow(['Banana', ‘6.50’])
   writer.writerow([‘Manzana', ‘7.40’])
```

La clase DictWriter

La clase **DictWriter** crea un objeto que opera de forma similar al objeto *writer* pero mapea diccionarios a cada línea del archivo. El parámetro *fieldnames* es una secuencia con las claves del diccionario, en el orden como se van a pasar a la función *writerow*. A diferencia de la clase **DictReader**, en este caso el parámetro *fieldnames* es obligatorio.

Al igual que con el objeto *writer* se puede indicar el dialecto y modificar uno o más valores del mismo.

A continuación se muestra un ejemplo:

```
import csv

with open('names.csv', 'w', newline='') as csvfile:
   fieldnames = ['first_name', 'last_name']
   writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

   writer.writeheader()
   writer.writerow({'first_name': 'Pablo', 'last_name': 'Garcia'})
   writer.writerow({'first_name': ‘Paula', 'last_name': 'Tina'})
```