**CSV Files**

The CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases. However, CSV is not a precise standard with multiple different applications having different conventions and specific standards. The Python csv module implements classes to read and write tabular data in CSV format. As part of this it supports the concept of a dialect which is a CSV format used by a specific application or suite of programs, for example, it supports an Excel dialect

This allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel.

The csv module provides a range of functions including:

• csv.reader (csvfile, dialect='excel', \*\*fmtparams) Returns a reader object which will iterate over lines in the given csvfile. An optional dialect parameter can be given. This may be an instance of a subclass of the Dialect class or one of the strings returned by the list\_dialects() function. The other optional fmtparams keyword arguments can be given to override individual formatting parameters in the current dialect.

• csv.writer (csvfile, dialect='excel', \*\*fmtparams) Returns a writer object responsible for converting the user’s data into delimited strings on the given csvfile. An optional dialect parameter provided. The fmtparams keyword arguments can be given to override individual formatting parameters in the current dialect.

• csv.list\_dialects() Return the names of all registered dialects. For example on a Mac OS X the default list of dialects is ['excel', 'excel-tab', 'unix'].

**The CSV Writer Class**

A CSV Writer is obtained from the csv.writer() function. The csvwriter supports two methods used to write data to the CSV file:

• csvwriter.writerow(row) Write the row parameter to the writer’s file object, formatted according to the current dialect.

• csvwriter.writerows(rows) Write all elements in rows (an iterable of row objects as described above) to the writer’s file object, formatted according to the current dialect.

• Writer objects also have the following public attribute:

• csvwriter.dialect A read-only description of the dialect in use by the writer.

**import** csv

**with** open("Example10.csv",'w',newline**=**'') **as** csvhand: *#to avoid one more newline as by default this will insert \n*

writer **=** csv.writer(csvhand) *#Returns a writer object responsible for converting the user’s data into delimited strings on the given csvfile*

writer.writerow(['She comes here', 'Sept 1963']) *#using this object what ever data is written will be of csv format*

writer.writerow(['She sits here', 'Dec 1963'])

writer.writerow(['She buys here', 'Apr 1964'])

writer.writerow(['She sleeps at night', 'July 1964'])

print(csv.list\_dialects()) *#list of exisitng dialects*

​

*#we can open the file , since it is csv , we can open in excel also*

['excel', 'excel-tab', 'unix']

​

**The CSV Reader Class**

A CSV Reader object is obtained from the csv.reader() function. It implements the iteration protocol. If a csv reader object is used with a for loop then each time round the loop it supplies the next row from the CSV file as a list, parsed according to the current CSV dialect. Reader objects also have the following public attributes:

• csvreader.dialect A read-only description of the dialect in use by the parser.

• csvreader.line\_num The number of lines read from the source iterator. This is not the same as the number of records returned, as records can span multiple lines.

**import** csv

**with** open("Example10.csv",'r',newline**=**'') **as** csvhand:

reader **=** csv.reader(csvhand)

**for** row **in** reader:

print(row,sep**=**',') *#will be displayed like a list*

print(**\***row,sep**=**',') *#if we want to convert that list into string , one of doing like this*

['She comes here', 'Sept 1963']

She comes here,Sept 1963

['She sits here', 'Dec 1963']

She sits here,Dec 1963

['She buys here', 'Apr 1964']

She buys here,Apr 1964

['She sleeps at night', 'July 1964']

She sleeps at night,July 1964

**The CSV DictWriter Class**

In many cases the first row of a CSV file contains a set of names (or keys) that define the fields within the rest of the CSV. That is the first row gives meaning to the columns and the data held in the rest of the CSV file. It is therefore very useful to capture this information and to structure the data written to a CSV file or loaded from a CSV file based on the keys in the first row. The csv.DictWriter returns an object that can be used to write values into the CSV file based on the use of such named columns. The file to be used with the DictWriter is provided when the class is instantiated.

**import** csv

**with** open("Example10.csv",'w',newline**=**'') **as** csvhand:

fieldname **=**['firstname','lastname','age']

writer **=** csv.DictWriter(csvhand,fieldnames **=** fieldname)

writer.writeheader()

writer.writerow({'firstname':'john',

'lastname':'smith',

'age':54})

writer.writerow({'firstname':'Henry',

'lastname':'smith',

'age':64})

writer.writerow({'firstname':'Harry',

'lastname':'smith',

'age':74})

​

*# Note that when the DictWriter is created a list of the keys must be provided*

*# that are used for the columns in the CSV file.*

*# The method writeheader() is then used to write the header row out to the*

*# CSV file.*

*# The method writerow() takes a dictionary object that has keys based on the*

*# keys defined for the DictWriter. These are then used to write data out to the*

*# CSV*

**The CSV DictReader Class**

As well as the csv.DictWriter there is a csv.DictReader. The file to be used with the DictReader is provided when the class is instantiated. As with the DictReader the DictWriter class takes a list of keys used to define the columns in the CSV file. If the headings to be used for the first row can be provided although this is optional (if a set of keys are not provided, then the values in the first row of the CSV file will be used as the fieldnames). The DictReader class provides several useful features including the fieldnames property that contains a list of the keys/headings for the CSV file as defined by the first row of the file. The DictReader class also implements the iteration protocol and thus it can be used in a for loop in which each row (after the first row) is returned in turn as a dictionary. The dictionary object representing each row can then be used to access each column value based on the keys defined in the first row.

**import** csv

**with** open("Example10.csv",'r',newline**=**'') **as** csvhand:

reader **=** csv.DictReader(csvhand)

**for** header **in** reader.fieldnames:

print(header,end**=**' ')

print("\n--------------------\n")

**for** rows **in** reader:

print(rows['firstname'],rows['lastname'],rows['age'])

firstname lastname age

--------------------

john smith 54

Henry smith 64

Harry smith 74

*#Example for overwriting the formatting \*\*fmt*

**import** csv

​

**with** open('eggs.csv', 'w', newline**=**'') **as** csvfile:

csv\_writer **=** csv.writer(csvfile, delimiter**=**',',quotechar**=**'"', quoting**=**csv.QUOTE\_MINIMAL)

csv\_writer.writerow(['Spam'] **\*** 5 **+** ['Baked Beans'])

csv\_writer.writerow(['Spam', **None** ,'Lovely "Spam"', 'Wonderful, Spam'])

**import** csv

​

**with** open("eggs.csv", newline**=**'') **as** csvfile:

**for** row **in** csv.reader(csvfile, delimiter**=**',', quotechar**=**'"'):

print(', '.join(row))

Spam, Spam, Spam, Spam, Spam, Baked Beans

Spam, , Lovely "Spam", Wonderful, Spam

lst **=** [["a", "a1"], ["b", "b1"], ["d", "d1 ,one"], "c", "f" ]

**with** open('list.csv', 'w', newline**=**'') **as** csvfile:

listWriter **=** csv.writer(csvfile)

**for** l **in** lst:

listWriter.writerow(l)

**with** open('list.csv','r',newline**=**'') **as** csvfile:

listreader **=** csv.reader(csvfile)

**for** l **in** listreader:

print(**\***l) *#for converting as string one way of doing like this or*

print(','.join(l))

['a', 'a1']

a,a1

['b', 'b1']

b,b1

['d', 'd1 ,one']

d,d1 ,one

['c']

c

['f']

f

​