CSV Module – Python

Information from Corey Schafer Youtube Tutorial

<https://www.youtube.com/watch?v=q5uM4VKywbA>

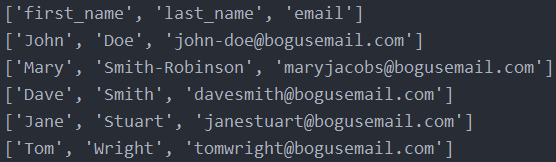
CSV stands for comma separated value, but it doesn’t have to be a comma, it can be a dash, underscore, tab delimiter ‘\t’ etc (This is commonly called a delimiter).

# How to read a csv file – default Way

This is how we usually read a csv file:

import csv  
  
filepath = 'names.csv'  
  
with open(filepath) as f:  
 csv\_reader = csv.reader(f)  
  
 for line in csv\_reader:  
 print(line)

And it should look something like this:



If you want a specific column of this .csv file you could just type:

for line in csv\_reader:  
 print(line[1])

This code will print out the second column of our csv file

The csv\_reader variable is something we have to iterate over. Everytime we type next() it is going to iterate over a line in the .csv file:

print(next(csv\_reader))



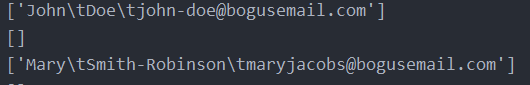
To loop over multiple lines we could do something like this:

[next(csv\_reader) for number in range(1, 10)]  
print(next(csv\_reader))

This will print the 10th line of our file

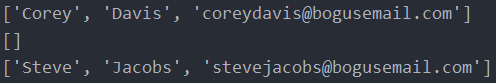
!! If you have a different delimiter than ‘,’ in your .csv file you have to specify that in the reader function otherwise your reading of the file will be messed up.

Example of reading a file with a tab delimiter without specifying the delimiter.



Notice it reads the whole line as one single string .

csv\_reader = csv.reader(file, delimiter='\t')



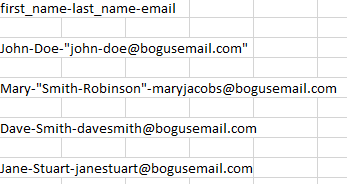
# Writing to a csv file – default way

Example code:

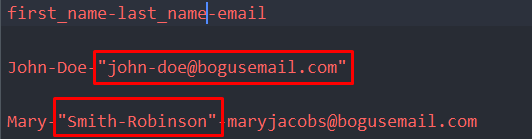
with open(filepath) as file:  
 csv\_reader = csv.reader(file)  
  
 with open('new\_names.csv', 'w') as new\_file:  
 csv\_writer = csv.writer(new\_file, delimiter='-') # To change delimiter  
  
 for line in csv\_reader:  
 csv\_writer.writerow(line)

We make a newfile and make a new variable called csv\_writer that hodls the function csv.writer(). After that we define the function arguments : (new\_file, delimiter =’-‘)

The outcome looks like this:



Notice this is not a good idea since we already had dashes in our .csv file but the function noticed this and the strings that contained a dash are now in quotes and are considered a separate value. In the end our csv file shouldn’t be affected. Notice example below:



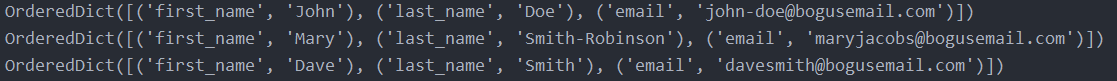
# Using csv.DictReader() – easier way

You can also read these .csv files with the DictReader class. This makes readability much easier.

The csv.Dictreader() gives you an ordered dictionary of your .csv file

with open(filepath) as file:  
 csv\_reader = csv.DictReader(file)  
  
 for line in csv\_reader:  
 print(line)

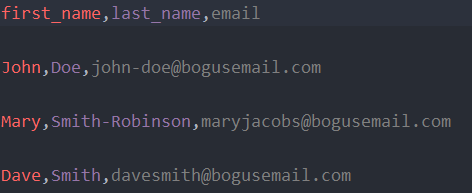
The outcome looks like this:



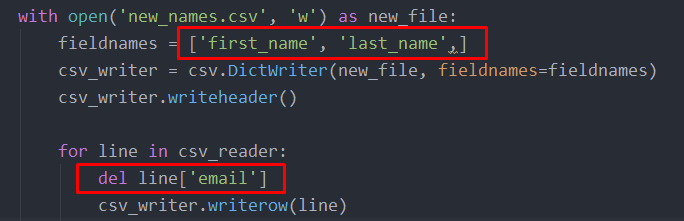
# Using csv.DictWriter() – easier way

import csv  
  
filepath = 'names.csv'  
  
with open(filepath) as file:  
 csv\_reader = csv.DictReader(file)  
  
 with open('new\_names.csv', 'w') as f:  
 fieldnames = ['first\_name', 'last\_name', 'email']  
 csv\_writer = csv.DictWriter(f, fieldnames=fieldnames)  
   
 csv\_writer.writeheader() # This line writes the headers  
   
 for row in csv\_reader:  
 csv\_writer.writerow(row)

Outcome:



And let’s say you don’t want to print out one key of your dictionary. It’s pretty easy to do that, all you have to do is delete the name from fieldnames and delete the key in your for loop



Outcome:

