# [Importing a CSV file into a sqlite3 database table using Python](https://stackoverflow.com/questions/2887878/importing-a-csv-file-into-a-sqlite3-database-table-using-python)

<https://stackoverflow.com/questions/2887878/importing-a-csv-file-into-a-sqlite3-database-table-using-python>

|  |  |  |
| --- | --- | --- |
|  | | import csv, sqlite3  con = sqlite3.connect(":memory:")  cur = con.cursor()  cur.execute("CREATE TABLE t (col1, col2);") # use your column names here  with open('data.csv','rb') as fin: # `with` statement available in 2.5+  # csv.DictReader uses first line in file for column headings by default  dr = csv.DictReader(fin) # comma is default delimiter  to\_db = [(i['col1'], i['col2']) for i in dr]  cur.executemany("INSERT INTO t (col1, col2) VALUES (?, ?);", to\_db)  con.commit()  con.close() |
|  | |  |
|  | Creating an sqlite connection to a file on disk is now a two-liner made possible by the pandas library:  df = pandas.read\_csv(csvfile)  df.to\_sql(table\_name, conn, if\_exists='append', index=False) | |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Importing CSV using DictReader:**  import csv, sqlite3  import logging  def \_get\_col\_datatypes(fin):  dr = csv.DictReader(fin) # comma is default delimiter  fieldTypes = {}  for entry in dr:  feildslLeft = [f for f in dr.fieldnames if f not in fieldTypes.keys()]  if not feildslLeft: break # We're done  for field in feildslLeft:  data = entry[field]  # Need data to decide  if len(data) == 0:  continue  if data.isdigit():  fieldTypes[field] = "INTEGER"  else:  fieldTypes[field] = "TEXT"  # TODO: Currently there's no support for DATE in sqllite  if len(feildslLeft) > 0:  raise Exception("Failed to find all the columns data types - Maybe some are empty?")  return fieldTypes  def escapingGenerator(f):  for line in f:  yield line.encode("ascii", "xmlcharrefreplace").decode("ascii")  def csvToDb(csvFile, outputToFile = False):  # TODO: implement output to file  with open(csvFile,mode='r', encoding="ISO-8859-1") as fin:  dt = \_get\_col\_datatypes(fin)  fin.seek(0)  reader = csv.DictReader(fin)  # Keep the order of the columns name just as in the CSV  fields = reader.fieldnames  cols = []  # Set field and type  for f in fields:  cols.append("%s %s" % (f, dt[f]))  # Generate create table statement:  stmt = "CREATE TABLE ads (%s)" % ",".join(cols)  con = sqlite3.connect(":memory:")  cur = con.cursor()  cur.execute(stmt)  fin.seek(0)  reader = csv.reader(escapingGenerator(fin))  # Generate insert statement:  stmt = "INSERT INTO ads VALUES(%s);" % ','.join('?' \* len(cols))  cur.executemany(stmt, reader)  con.commit()   |  |  |  | | --- | --- | --- | | return con |  |  | |

|  |  |  |
| --- | --- | --- |
|  | | Alternative 1:  import csv, sqlite3  conn = sqlite3.connect("pcfc.sl3")  curs = conn.cursor()  curs.execute("CREATE TABLE PCFC (id INTEGER PRIMARY KEY, type INTEGER, term TEXT, definition TEXT);")  reader = csv.reader(open('PC.txt', 'r'), delimiter='|')  for row in reader:  to\_db = [unicode(row[0], "utf8"), unicode(row[1], "utf8"), unicode(row[2], "utf8")]  curs.execute("INSERT INTO PCFC (type, term, definition) VALUES (?, ?, ?);", to\_db)  conn.commit()  My text file (PC.txt) looks like this: |
|  | | Alternative 2: |
|  | #!/usr/bin/python  # -\*- coding: utf-8 -\*-  import sys, csv, sqlite3  def main():  con = sqlite3.connect(sys.argv[1]) # database file input  cur = con.cursor()  cur.executescript("""  DROP TABLE IF EXISTS t;  CREATE TABLE t (COL1 TEXT, COL2 TEXT);  """) # checks to see if table exists and makes a fresh table.  with open(sys.argv[2], "rb") as f: # CSV file input  reader = csv.reader(f, delimiter=',') # no header information with delimiter  for row in reader:  to\_db = [unicode(row[0], "utf8"), unicode(row[1], "utf8")] # Appends data from CSV file representing and handling of text  cur.execute("INSERT INTO neto (COL1, COL2) VALUES(?, ?);", to\_db)  con.commit()  con.close() # closes connection to database  if \_\_name\_\_=='\_\_main\_\_':  main() | |

**Usage of csv.DictReader**

https://courses.cs.washington.edu/courses/cse140/13wi/csv-parsing.html

[CSV](https://en.wikipedia.org/wiki/Comma-separated_values), or "comma-separated values", is a common file format for data. The [csv module](https://docs.python.org/2/library/csv.html) helps you to elegantly process data stored within a CSV file. Also see the [csv documentation](https://docs.python.org/2/library/csv.html).

This guide uses the following example file, people.csv.

id,name,age,height,weight

1,Alice,20,62,120.6

2,Freddie,21,74,190.6

3,Bob,17,68,120.0

Your Python code must import the csv library.

import csv

Open the file by calling [open](https://docs.python.org/2/library/functions.html#open) and then [csv.DictReader](https://docs.python.org/2/library/csv.html#csv.DictReader).

input\_file = csv.DictReader(open("people.csv"))

You may iterate over the rows of the csv file by iterating ove input\_file. (Similarly to other files, you need to re-open the file if you want to iterate a second time.)

for row in input\_file:

print row

When you iterate over a normal file, each iteration of the loop produces a single string that represents the contents of that line. When you iterate over a CSV file, each iteration of the loop produces a dictionary from strings to strings. They keys are the names of the columns (from the first row of the file, which is skipped over), and the values are the data from the row being read. For example, the above loop prints the following:

{'age': '20', 'height': '62', 'id': '1', 'weight': '120.6', 'name': 'Alice'}

{'age': '21', 'height': '74', 'id': '2', 'weight': '190.6', 'name': 'Freddie'}

{'age': '17', 'height': '68', 'id': '3', 'weight': '120.0', 'name': 'Bob'}

To print the entire contents of the DictReader, you might execute the following code:

Finally, here is a complete example usage of csv.DictReader using people.csv. This example finds the oldest person within the file and that person's age.

import csv

input\_file = csv.DictReader(open("people.csv"))

max\_age = None

oldest\_person = None

for row in input\_file:

age = int(row["age"])

if max\_age == None or max\_age < age:

max\_age = age

oldest\_person = row["name"]

if max\_age != None:

print "The oldest person is %s, who is %d years old." % (oldest\_person, max\_age)

else:

print "The file does not contain any people."

And the output from this program:

The oldest person is Freddie, who is 21 years old.