I wanted to use [Python's built-in CSV reader class](https://docs.python.org/2/library/csv.html#csv.reader) and skip any parsing of the header line. So I did this:

with open('mycsv.csv', 'r') as csvfile:

csvreader = csv.reader(csvfile)

# This skips the first row of the CSV file.

# csvreader.next() also works in Python 2.

next(csvreader)

for row in csvreader:

# do stuff with rows...

The call to next reads the first row and discards it. From there, we're ready to iterate through the actual data! One small caveat: I had issues in Python 3 when opening the file in binary mode (rb instead of r).

**How to convert python tuple to dict**

http://www.yilmazhuseyin.com/blog/dev/how-convert-python-tuple-dict/

Sometimes you might need to convert tuple (or list) to dict object to make it more readable. After all l['product\_id'] is more readable than l[4]. For this example I will use following sample data:

#create sample keys for our dict

keys = ['id','user\_id','user\_age','product\_id','product\_weight']

#create sample data

l = list()

for i in range(1,10):

l.append((1\*i,2\*i,3\*i,4\*i,5\*i))

print l

# [(1, 2, 3, 4, 5),

# (2, 4, 6, 8, 10),

# (3, 6, 9, 12, 15),

# (4, 8, 12, 16, 20),

# (5, 10, 15, 20, 25),

# (6, 12, 18, 24, 30),

# (7, 14, 21, 28, 35),

# (8, 16, 24, 32, 40),

# (9, 18, 27, 36, 45)]

For start, I have a list of tuples and I would like to have a dictionary. To archive this first I need to zip two tuples together

zip(keys,l[2])

# [('id', 3),

# ('user\_id', 6),

# ('user\_age', 9),

# ('product\_id', 12),

# ('product\_weight', 15)]

if we send this to dict constructor, we will have first values of tuples as keys and second value of tuples as values and we will achieve our goal.

dict(zip(keys,l[2]))

# {'id': 3, 'product\_id': 12, 'product\_weight': 15, 'user\_age': 9, 'user\_id': 6}

This is how we can convert tuple to dict. If we need to convert whole list of tuples to list of dictionaries

[dict(zip(keys,row)) for row in l]

#[{'id': 1, 'product\_id': 4, 'product\_weight': 5, 'user\_age': 3, 'user\_id': 2},

# {'id': 2, 'product\_id': 8, 'product\_weight': 10, 'user\_age': 6, 'user\_id': 4},

# {'id': 3, 'product\_id': 12, 'product\_weight': 15, 'user\_age': 9, 'user\_id': 6},

# {'id': 4, 'product\_id': 16, 'product\_weight': 20, 'user\_age': 12, 'user\_id': 8},

# {'id': 5, 'product\_id': 20, 'product\_weight': 25, 'user\_age': 15, 'user\_id': 10},

# {'id': 6, 'product\_id': 24, 'product\_weight': 30, 'user\_age': 18, 'user\_id': 12},

# {'id': 7, 'product\_id': 28, 'product\_weight': 35, 'user\_age': 21, 'user\_id': 14},

# {'id': 8, 'product\_id': 32, 'product\_weight': 40, 'user\_age': 24, 'user\_id': 16},

# {'id': 9, 'product\_id': 36, 'product\_weight': 45, 'user\_age': 27, 'user\_id': 18}]

This technique is quite useful to convert query results to more meaningful format.

|  |  |  |  |
| --- | --- | --- | --- |
| **How to convert python tuple to dict-2**  For the tuple, t = ((1, 'a'),(2, 'b')) dict(t) returns {1: 'a', 2: 'b'}  Is there a good way to get {'a': 1, 'b': 2} (keys and vals swapped)? | | | |
|  | >>> t = ((1, 'a'),(2, 'b'))  >>> dict((y, x) for x, y in t)  {'a': 1, 'b': 2} | | |
|  | | A slightly simpler method:  >>> t = ((1, 'a'),(2, 'b'))  >>> dict(map(reversed, t))  {'a': 1, 'b': 2} |

Even more concise if you are on python 2.7:

>>> t = ((1,'a'),(2,'b'))

>>> {y:x for x,y in t}

{'a':1, 'b':2}

|  |  |
| --- | --- |
|  | **As others have pointed out, sqlite's ALTER TABLE statement does *not* support DROP COLUMN, and the standard recipe to do this does not preserve constraints & indices.**  Here's some python code to do this generically, while *maintaining* all the key constraints and indices.  **Please back-up your database before using!** This function relies on doctoring the original CREATE TABLE statement and is potentially a bit unsafe - for instance it will do the wrong thing if an identifier contains an embedded comma or parenthesis.  ***UPDATE*** I found a better way to parse using the open-source sqlparse package. If there is any interest I will post it here, just leave a comment asking for it ...  import re  import random  def DROP\_COLUMN(db, table, column):  columns = [ c[1] for c in db.execute("PRAGMA table\_info(%s)" % table) ]  columns = [ c for c in columns if c != column ]  sql = db.execute("SELECT sql from sqlite\_master where name = '%s'"  % table).fetchone()[0]  sql = format(sql)  lines = sql.splitlines()  findcol = r'\b%s\b' % column  keeplines = [ line for line in lines if not re.search(findcol, line) ]  create = '\n'.join(keeplines)  create = re.sub(r',(\s\*\))', r'\1', create)  temp = 'tmp%d' % random.randint(1e8, 1e9)  db.execute("ALTER TABLE %(old)s RENAME TO %(new)s" % {  'old': table, 'new': temp })  db.execute(create)  db.execute("""  INSERT INTO %(new)s ( %(columns)s )  SELECT %(columns)s FROM %(old)s  """ % {  'old': temp,  'new': table,  'columns': ', '.join(columns)  })  db.execute("DROP TABLE %s" % temp)  def format(sql):  sql = sql.replace(",", ",\n")  sql = sql.replace("(", "(\n")  sql = sql.replace(")", "\n)")  return sql |