**NOTES**

**CHAPTER 07: WORKING WITH CSV FILES**

**INTRODUCTION**

***When you export data from Website, the files can be delivered in the following formats***

1. **CSV - COMMA SEPARATED VALUES**
2. **PSV - PIPE SEPARATED VALUES**
3. **TSV - TAB SEPARATED VALUES**
4. **CSV - COMMA SEPARATED VALUES**

A CSV(Comma Separated Values) is a plain-text file format used to store tabular data such as a spreadsheet or a database. It essentially stores a tabular data which comprises of numbers and text into plain text.

1. **PSV - PIPE SEPARATED VALUES**

When you export contact data and survey data from the Gateway eNews letter and Website systems, the files will be delivered in a "pipe delimited" format. This means that the various columns are separated by a pipe character**.**

1. **TSV - TAB SEPARATED VALUES**

A text file format that uses tab characters as separators between fields. The tab delimited format stores information from a database or spreadsheet in the format of a tabular structure. Both Microsoft and Google allow the user to convert a spreadsheet into tab delimited format.

**WHAT IS A CSV FILE?**

A CSV file is a type of plain text file that uses specific structuring to arrange tabular data. CSV is a common format for data interchange as it's compact, simple and general. Many online services allow its users to export tabular data from the website into a CSV file. Files of CSV will open into Excel, and nearly all databases have a tool to allow import from CSV file. The standard format is defined by rows and columns data. Moreover, each row is terminated by a newline to begin the next row. Also within the row, each column is separated by a comma.

CSV (Comma-separated values) is a common data exchange format used by the applications to produce and consume data. Some other well-known data exchange formats are XML, HTML, JSON etc.

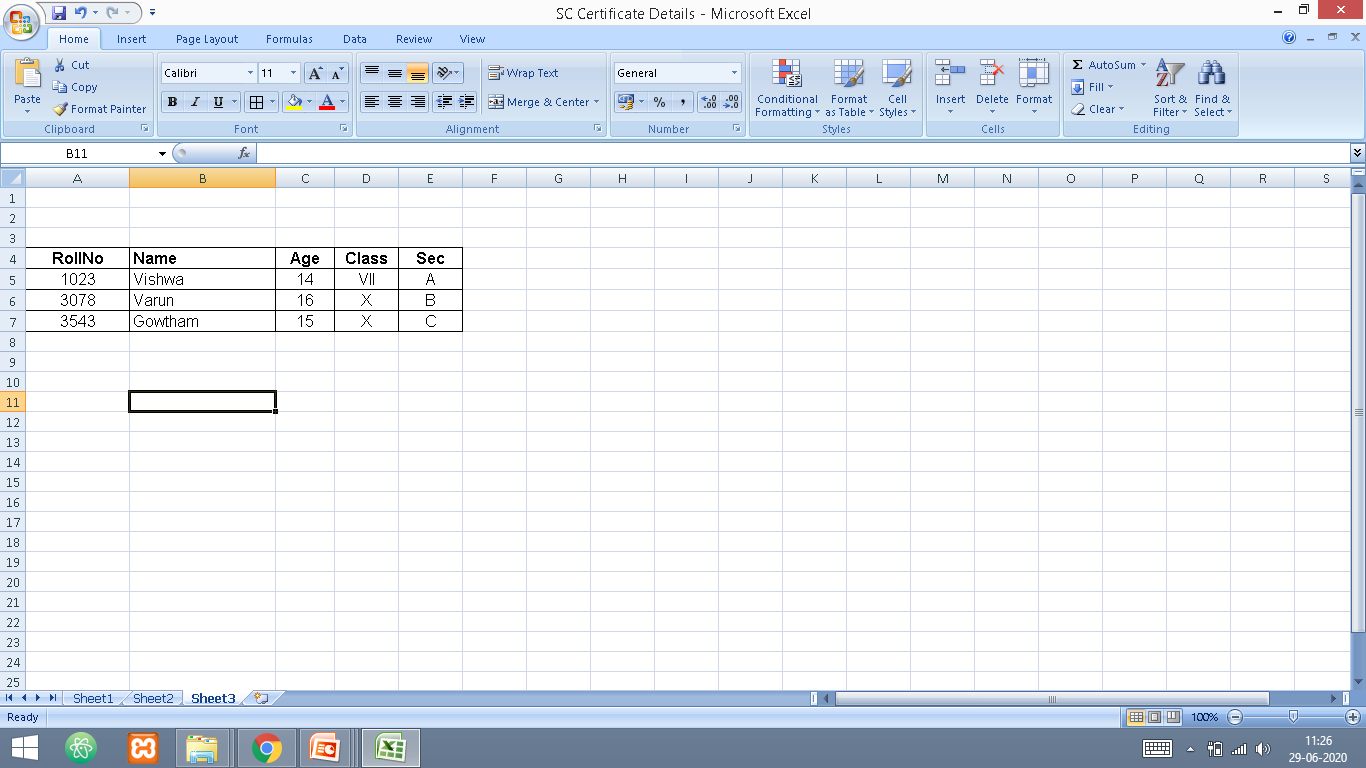
A CSV file is a simple text file where each line contains a list of values (or fields) delimited by commas.

Although the term "Comma" appears in the format name itself, but you will encounter CSV files where data is delimited using tab (\t) or pipe (|) or any other character that can be used as a delimiter.

The first line of the CSV file represents the header containing a list of column names in the file. The header is optional but highly recommended.

The CSV file is commonly used to represent tabular data. For example, consider the following table:

To retain the commas inside the Address field enclose it in double quotation marks, as follows:



|  |  |
| --- | --- |
|  |  |

Tabular data is represented by comma separated values

**RollNo,Name,Age,Class,Sec**

1023,Vishwa,14,VII,A

3078,Varun,16,X,B

3543,Gowtham,15,X,C

Every line of the file is called a record. And each record consists of fields that are separated by *commas*which are also known as “delimiter” which is the default delimiter, others include pipe(|), semicolon(;). Given below is a structure of a Normal CSV File separated by a comma

**WHY IS CSV FILE FORMAT USED?**

CSV is a plain-text file which makes it easier for data interchange and also easier to import onto spreadsheet or database storage.

For example: You might want to export the data of a certain statistical analysis to CSV file and then import it to the spreadsheet for further analysis. Overall it makes users working experience very easy programmatically. Any language supporting a text file or string manipulation like Python can work with CSV files directly.

**CSV ADVANTAGES**

* CSV is human readable and easy to edit manually
* CSV is simple to implement and parse
* CSV is processed by almost all existing applications
* CSV provides a straightforward information schema
* CSV is faster to handle
* CSV is smaller in size
* CSV is considered to be standard format
* CSV is compact. For XML you start tag and end tag for each column in each row. In CSV you write the column headers only once.
* CSV is easy to generate

**CSV DISADVANTAGES**

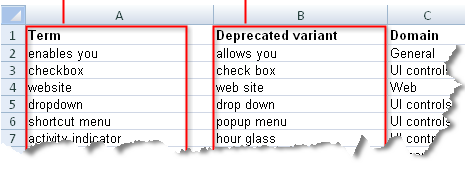
* CSV allows to move most basic data only. Complex configurations cannot be imported and exported this way
* There is no distinction between text and numeric values
* No standard way to represent binary data
* Problems with importing CSV into SQL (no distinction between NULL and quotes)
* Poor support of special characters
* No standard way to represent control characters
* Lack of universal standard

**CSV FILE TYPES**

1. **CSV (COMMA DELIMITED)**
2. **CSV UTF-8 (COMMA DELIMITED)**
3. **CSV (MACINTOSH)**
4. **CSV (MS-DOS)**
5. **CSV (COMMA DELIMITED)**

A comma-separated values file is a delimited text file that uses a comma to separate values. Each line of the file is a data record. Each record consists of one or more fields, separated by commas. Exporting to CSV uses a default encoding of Unicode (UTF-16le)

1. **CSV UTF-8 (COMMA DELIMITED)**

UTF-8 (Unicode Transformation Format -8 ) encoding, also referred to as "Unicode - UTF8" UTF-8 encoded CSV files will work well with Accompa (Accompa is a leading cloud-based requirements management software that helps you capture, track and manage requirements for your products & projects). just English characters, or also contain non-English characters such as é, ç, ü

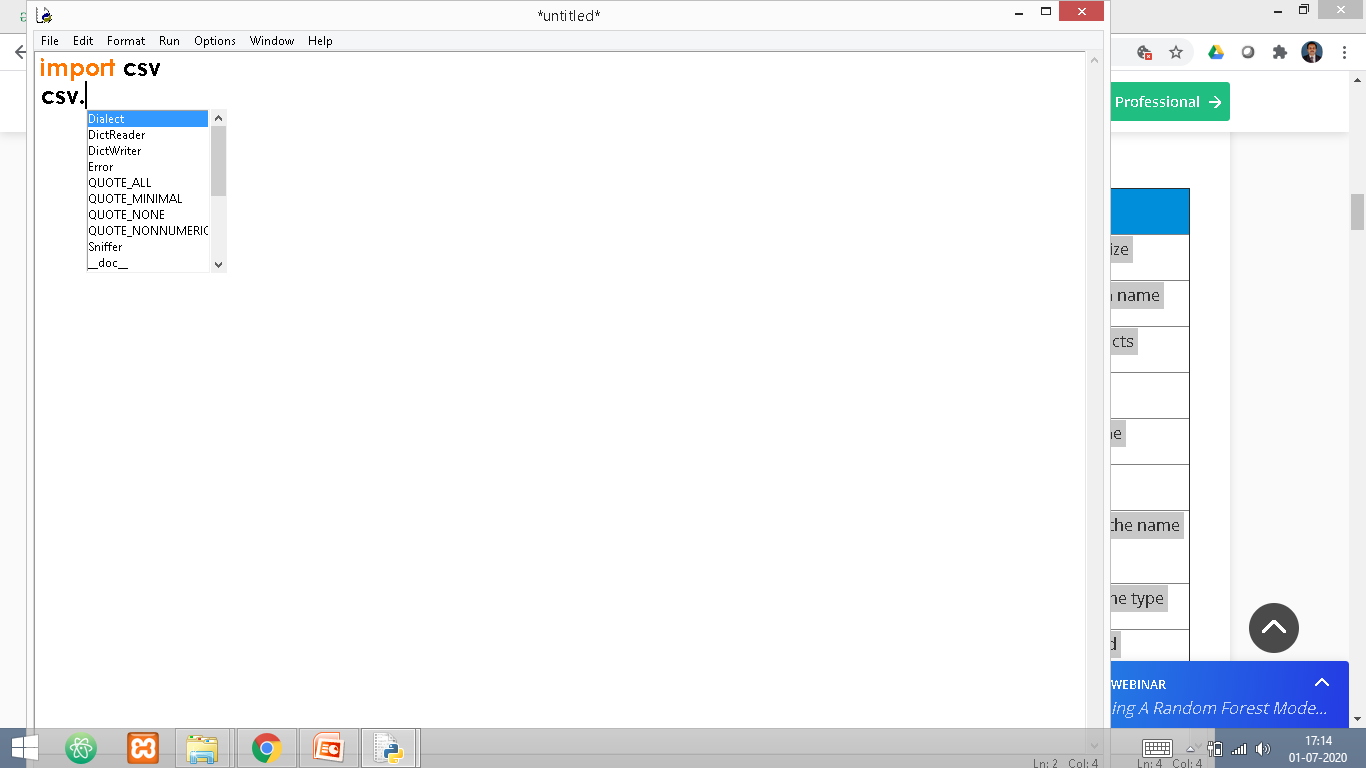
1. **CSV (MACINTOSH)**

CSV (Macintosh) .csv. Saves a workbook as a comma-delimited text file for use on the Macintosh operating system, and ensures that tab characters, line breaks, and other characters are interpreted correctly.

1. **CSV (MS-DOS)**

CSV (DOS).csv. Saves a workbook as a comma-delimited text file for use on the Macintosh operating system, and ensures that tab characters, line breaks, and other characters are interpreted correctly.

**PYTHON CSV MODULE**

Python uses a CSV package which is a part of the standard library, so you need not install it.

Python CSV package contains following functions:

|  |  |  |
| --- | --- | --- |
| **Ser** | **Functions** | **Description** |
| **1** | **csv.field\_size\_limit** | **It returns the maximum field size** |
| **2** | **csv.get\_dialect** | **Fetches the dialect associated with name** |
| **3** | **csv.list\_dialects** | **Displays all the registered dialects** |
| **4** | **csv.reader** | **Read data from csv file** |
| **5** | **csv.register\_dialect** | **Dialect associated with a name** |
| **6** | **csv.writer** | **Writes data to a csv file** |
| **7** | **csv.unregister\_dialect** | **It deletes the dialect associated with the name dialect registry** |
| **8** | **csv.QUOTE\_ALL** | **Quotes everything irrespective of the type** |
| **9** | **csv.QUOTE\_MINIMAL** | **Quotes special character field** |
| **10** | **csv.QUOTE\_NONNUMERIC** | **Quotes fields that are not numeral** |
| **11** | **csv.QUOTE\_NONE** | **Doesn’t quote anything in output** |

**READING AND WRITING FILES**

The csv module is used for reading and writing files. It mainly provides following classes and functions:

**1. reader()**

**2. writer()**

**3. DictReader()**

**4. DictWriter()**

## 1. READING A CSV FILE WITH reader()

## The reader() function takes a file object and returns a \_csv.reader object that can be used to iterate over the contents of a CSV file. The syntax of reader() function is as follows:

## Syntax: *reader(fileobj [, dialect='excel' [, \*\*fmtparam] ]) -> \_csv.reader*

| **Argument** | **Description** |
| --- | --- |
| **fileobj** | (required) It refers to the file object |
| **dialect** | (optional) Dialect refers to the different ways of formatting the CSV document. By default, the csv module uses the same format as Microsoft Excel. We will discuss dialect in detail later in this post. |
| **fmtparam** | (optional) It refers to the set of keyword arguments to customize the dialect (see the next section). |

Let's say we have following CSV file:

employees.csv

|  |  |
| --- | --- |
| 1  2  3  4 | **id,name,email,age,designation**  1,John,john@mail.com,24,programmer  2,Bob,bob@mail.com,34,designer  3,Mary,mary@mail.com,43,sales |

**import csv**

**with open('employees.csv', 'rt') as f:**

**csv\_reader = csv.reader(f)**

**for line in csv\_reader:**

**print(line)**

Here is how to read this CSV file:

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4 | ['id', 'name', 'email', 'age', 'designation']  ['1', 'John', 'john@mail.com', '24', 'programmer']  ['2', 'Bob', 'bob@mail.com', '34', 'designer']  ['3', 'Mary', 'mary@mail.com', '43', 'sales'] |

Notice that each line in the CSV file is returned as a list of strings.

To get the data from certain fields, you can use indexing. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **import** **csv**  **with** open('employees.csv', 'rt') **as** f:  csv\_reader = csv.reader(f)  **for** line **in** csv\_reader:  **print**(line[0], line[1], line[2]) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4 | id name email  1 John john@mail.com  2 Bob bob@mail.com  3 Mary mary@mail.com |

If you want to skip heading call the next() built-in function on the \_csv.reader object and then loop over the remaining lines as usual.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **import** **csv**  **with** open('employees.csv', 'rt') **as** f:  csv\_reader = csv.reader(f)  next(csv\_reader) *# skip the heading*  **for** line **in** csv\_reader:  **print**(line[0], line[1], line[2]) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3 | 1 John john@mail.com  2 Bob bob@mail.com  3 Mary mary@mail.com |

## CUSTOMIZING THE reader()

By default, the csv module works according to the format used by Microsoft excel, but you can also define your own format using something called Dialect.

The following are some additional arguments that you can pass to the reader() function to customize its working.

* delimiter - It refers to the character used to separate values (or fields) in the CSV file. It defaults to comma (,).
* skipinitialspace  - It controls how the space following the delimiter will be interpreted. If True , the initial whitespaces will be removed. It defaults to False.
* lineterminator - It refers to the character sequence used to terminate the line. It defaults to \r\n.
* quotechar - It refers to the single character string that will be used to quote values if special characters (like delimiter) appears inside the field. It defaults to ".
* quoting  - controls when quotes should be generated by the writer or recognized by the reader. It can take one of the following constants:
  + csv.QUOTE\_MINIMAL means add quote only when required, for example, when a field contains either the quotechar or the delimiter. This is the default.
  + csv.QUOTE\_ALL means quotes everything regardless of the field type.
  + csv.QUOTE\_NONNUMERIC means quotes everything except integers and floats.
  + csv.QUOTE\_NONE means that do not quote anything on output. However, while reading quotes are included around the field values.
* escapechar - It refers to the one-character string used to escape the delimiter when quoting is set to QUOTE\_NONE. It defaults to None.
* doublequote  - controls the handling of quotes inside fields. When True, two consecutive quotes are interpreted as one during read, and when writing, each quote character embedded in the data is written as two quotes. Let's workthrough some examples to better understand how these arguments work:

### DELIMITER ARGUMENT

**employees\_pipe.csv**

|  |  |
| --- | --- |
| 1  2  3  4 | id|name|email|age|designation  1|John|john@mail.com|24|programmer  2|Bob|bob@mail.com|34|designer  3|Mary|mary@mail.com|43|sales |

This file uses pipe (|) character as a delimiter. Here is how to read this CSV file:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **import** **csv**  **with** open('employees.csv', 'rt') **as** f:  csv\_reader = csv.reader(f, delimiter='|')  **for** line **in** csv\_reader:  **print**(line) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4 | ['id', 'name', 'email', 'age', 'designation']  ['1', 'John', 'john@mail.com', '24', 'programmer']  ['2', 'Bob', 'bob@mail.com', '34', 'designer']  ['3', 'Mary', 'mary@mail.com', '43', 'sales'] |

### SKIPINITIALSPACE ARGUMENT

**baseball\_players.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | "Name", "Team", "Position", "Height(inches)", "Weight(lbs)", "Age"  "Adam Donachie", "BAL", "Catcher", 74, 180, 22.99  "Paul Bako", "BAL", "Catcher", 74, 215, 34.69  "Ramon Hernandez", "BAL", "Catcher", 72, 210, 30.78  "Kevin Millar", "BAL", "First Baseman", 72, 210, 35.43  "Chris Gomez", "BAL", "First Baseman", 73, 188, 35.71  "Brian Roberts", "BAL", "Second Baseman", 69, 176, 29.39  "Miguel Tejada", "BAL", "Shortstop", 69, 209, 30.77  "Melvin Mora", "BAL", "Third Baseman", 71, 200, 35.07 |

This CSV file contains spaces following the comma (,). To read this CSV file correctly, set skipinitialspace to True, as follows:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | **import** **csv**  **with** open('baseball\_players.csv', 'rt') **as** f:  csv\_reader = csv.reader(f, skipinitialspace=True)  **for** line **in** csv\_reader:  **print**(line) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | ['Name', 'Team', 'Position', 'Height(inches)', 'Weight(lbs)', 'Age']  ['Adam Donachie', 'BAL', 'Catcher', '74', '180', '22.99']  ['Paul Bako', 'BAL', 'Catcher', '74', '215', '34.69']  ['Ramon Hernandez', 'BAL', 'Catcher', '72', '210', '30.78']  ['Kevin Millar', 'BAL', 'First Baseman', '72', '210', '35.43']  ['Chris Gomez', 'BAL', 'First Baseman', '73', '188', '35.71']  ['Brian Roberts', 'BAL', 'Second Baseman', '69', '176', '29.39']  ['Miguel Tejada', 'BAL', 'Shortstop', '69', '209', '30.77']  ['Melvin Mora', 'BAL', 'Third Baseman', '71', '200', '35.07'] |

## WRITING CSV FILES WITH writer()

To write data to a CSV file we use the writer() function. It accepts the same argument as the reader() function but returns a writer object (i.e \_csv.writer):

**Syntax:** writer(fileobj [, dialect='excel' [, \*\*fmtparam] ]) -> csv\_writer

| **ARGUMENT** | **DESCRIPTION** |
| --- | --- |
| **fileobj** | (required) It refers to the file object |
| **dialect** | (optional) Dialect refers to the different ways of formatting the CSV document. By default, the csv module uses the same format as Microsoft Excel. We will discuss dialect in detail later in this post. |
| **fmtparam** | (optional) Formatting parameters, work same as the reader()'s function. |

The writer instance provides the following two methods to write data:

| **Method** | **Description** |
| --- | --- |
| **writerow(row)** | Writes a single row of data and returns the number of characters written. The row must be a sequence of strings and number. |
| **writerows(rows)** | Writes multiple rows of data and returns None. The rows must be a sequence. |

Here are examples:

**Example 1**: Using writerow()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f)  csv\_writer.writerow(header) *# write header*  **for** row **in** rows:  csv\_writer.writerow(row) |

**Example 2**: Using writerows()

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f)  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

The output generated by both listing will be the same and it looks like this:

**customers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip  1,Hannah,"4891 Blackwell Street, Anchorage, Alaska",99503  2,Walton,"4223 Half and Half Drive, Lemoore, California",97401  3,Sam,"3952 Little Street, Akron, Ohio",93704  4,Chris,"3192 Flinderation Road, Arlington Heights, Illinois",62677  5,Doug,"3236 Walkers Ridge Way, Burr Ridge",61257 |

Notice that only the address field is wrapped around double quotes. This is because by default the quoting argument is set to QUOTE\_MINIMAL. In other words, fields will be quoted only when quotechar or delimiter appears in the data.

Let's say you want double quotes around all textual data. To achieve this, set quoting argument to QUOTE\_NONNUMERIC.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f, quoting=csv.QUOTE\_NONNUMERIC)  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

**customers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | "id","name","address","zip"  1,"Hannah","4891 Blackwell Street, Anchorage, Alaska",99503  2,"Walton","4223 Half and Half Drive, Lemoore, California",97401  3,"Sam","3952 Little Street, Akron, Ohio",93704  4,"Chris","3192 Flinderation Road, Arlington Heights, Illinois",62677  5,"Doug","3236 Walkers Ridge Way, Burr Ridge",61257 |

Now all the names and addresses have double quotes around them.

If you want double quotes around all fields regardless of whether quotechar or delimiter appears in the data or not, set quoting to csv.QUOTE\_ALL.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f, quoting=csv.QUOTE\_ALL)  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | "id","name","address","zip"  "1","Hannah","4891 Blackwell Street, Anchorage, Alaska","99503"  "2","Walton","4223 Half and Half Drive, Lemoore, California","97401"  "3","Sam","3952 Little Street, Akron, Ohio","93704"  "4","Chris","3192 Flinderation Road, Arlington Heights, Illinois","62677"  "5","Doug","3236 Walkers Ridge Way, Burr Ridge","61257" |

Everything is double-quoted now.

It is important to note that when quoting is on (i.e quoting parameter has a value other than csv.QUOTE\_NONE), the csv module uses the quotechar (which defaults to ") to quote field.

The following listing changes the quote character from double quote (") to a single quote (').

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f, quotechar="'")  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip  1,Hannah,'4891 Blackwell Street, Anchorage, Alaska',99503  2,Walton,'4223 Half and Half Drive, Lemoore, California',97401  3,Sam,'3952 Little Street, Akron, Ohio',93704  4,Chris,'3192 Flinderation Road, Arlington Heights, Illinois',62677  5,Doug,'3236 Walkers Ridge Way, Burr Ridge',61257 |

In this case, the csv module uses the single quote (') instead of (") to quote fields containing quotechar or delimiter.

We can also turn off quoting all-together by setting quoting to csv.QUOTE\_NONE. However, if you do that and delimiter character appears in the data then you will get an error like this:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f, quoting=csv.QUOTE\_NONE)  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4 | Traceback (most recent call last):  ...  csv\_writer.writerows(rows)  \_csv.Error: need to escape, but no escapechar set |

The problem is that the address field contains embedded commas (,) and since we have turned off the ability to quote fields, the csv module doesn't know how to escape them properly.

This where the escapechar argument comes into play. It takes a one-character string that will be used to escape the delimiter when the the quoting is turned off (i.e quoting=csv.QUOTE\_NONE).

The following listing set the escapechar to backslash (\).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | **import** **csv**  header = ['id', 'name', 'address', 'zip']  rows = [  [1, 'Hannah', '4891 Blackwell Street, Anchorage, Alaska', 99503 ],  [2, 'Walton', '4223 Half and Half Drive, Lemoore, California', 97401 ],  [3, 'Sam', '3952 Little Street, Akron, Ohio', 93704],  [4, 'Chris', '3192 Flinderation Road, Arlington Heights, Illinois', 62677],  [5, 'Doug', '3236 Walkers Ridge Way, Burr Ridge', 61257],  ]  **with** open('customers.csv', 'wt') **as** f:  csv\_writer = csv.writer(f, quoting=csv.QUOTE\_NONE, escapechar='**\\**')  csv\_writer.writerow(header) *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip  1,Hannah,4891 Blackwell Street**\,** Anchorage**\,** Alaska,99503  2,Walton,4223 Half and Half Drive**\,** Lemoore**\,** California,97401  3,Sam,3952 Little Street**\,** Akron**\,** Ohio,93704  4,Chris,3192 Flinderation Road**\,** Arlington Heights**\,** Illinois,62677  5,Doug,3236 Walkers Ridge Way**\,** Burr Ridge,61257 |

Notice that the commas (,) in the address field is escaped using the backslash (\) character.

You should now have a good understanding of various formatting arguments and the context in which they are used with the reader() and writer() function. In the next section will see some other ways to read and write data.

## READING A CSV FILE WITH DictReader

DictReader works almost exactly like reader() but instead of retuning a line as a list, it returns a dictionary. Its syntax is as follows:

**Syntax:**: DictReader(fileobj, fieldnames=None, restkey=None, restval=None, dialect='excel', \*\*fmtparam)

| **ARGUMENT** | **DESCRIPTION** |
| --- | --- |
| fileobj | (required) It refers to the file object. |
| fieldnames | (optional) It refers to the list of keys that will be used in the returned dictionary in order. If omitted, the field names are inferred from the first row of the CSV file. |
| restkey | (optional) If the row has more fields than specified in the fieldnames parameter, then the remaining fields is stored as a sequence keyed by the value of restkey argument. |
| restval | (optional) It provides value to fields which are missing from the input. |
| dialect | (optional) Dialect refers to the different ways of formatting the CSV document. By default, the csv module uses the same format as Microsoft excel. We will discuss dialect in detail later in this post. |
| fmtparam | It refers to formatting arguments and works exactly like reader() and writer(). |

Let's take some examples:

**Example 1**:

**customers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip  1,Hannah,4891 Blackwell Street\, Anchorage\, Alaska,99503  2,Walton,4223 Half and Half Drive\, Lemoore\, California,97401  3,Sam,3952 Little Street\, Akron\, Ohio,93704  4,Chris,3192 Flinderation Road\, Arlington Heights\, Illinois,62677  5,Doug,3236 Walkers Ridge Way\, Burr Ridge,61257 |
| 1  2  3  4  5  6  7 | **import** **csv**  **with** open('customers.csv', 'rt') **as** f:  csv\_reader = csv.DictReader(f, escapechar='**\\**')  **for** row **in** csv\_reader:  **print**(row) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | {'id': '1', 'name': 'Hannah', 'zip': '99503', 'address': '4891 Blackwell Street, Anchorage, Alaska'}  {'id': '2', 'name': 'Walton', 'zip': '97401', 'address': '4223 Half and Half Drive, Lemoore, California'}  {'id': '3', 'name': 'Sam', 'zip': '93704', 'address': '3952 Little Street, Akron, Ohio'}  {'id': '4', 'name': 'Chris', 'zip': '62677', 'address': '3192 Flinderation Road, Arlington Heights, Illinois'}  {'id': '5', 'name': 'Doug', 'zip': '61257', 'address': '3236 Walkers Ridge Way, Burr Ridge'}  \*\*Note:\*\* Order of keys in the result may vary. Since dictionary doesn't preserve the order of elements. |

In this case, the field names are inferred from the first line (or header) of the CSV file.

**Example 2**: Using fieldnames parameter

|  |  |
| --- | --- |
| 1  2  3  4  5 | 1,Hannah,4891 Blackwell Street\, Anchorage\, Alaska,99503  2,Walton,4223 Half and Half Drive\, Lemoore\, California,97401  3,Sam,3952 Little Street\, Akron\, Ohio,93704  4,Chris,3192 Flinderation Road\, Arlington Heights\, Illinois,62677  5,Doug,3236 Walkers Ridge Way\, Burr Ridge,61257 |

This CSV file has no header. So we have to provide field names via the fieldnames parameter.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | **import** **csv**  **with** open('customers.csv', 'rt') **as** f:  fields = ['id', 'name', 'address', 'zip']  csv\_reader = csv.DictReader(f, fieldnames=fields, escapechar='**\\**')  **for** row **in** csv\_reader:  **print**(row) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | {'name': 'Hannah', 'zip': '99503', 'id': '1', 'address': '4891 Blackwell Street, Anchorage, Alaska'}  {'name': 'Walton', 'zip': '97401', 'id': '2', 'address': '4223 Half and Half Drive, Lemoore, California'}  {'name': 'Sam', 'zip': '93704', 'id': '3', 'address': '3952 Little Street, Akron, Ohio'}  {'name': 'Chris', 'zip': '62677', 'id': '4', 'address': '3192 Flinderation Road, Arlington Heights, Illinois'}  {'name': 'Doug', 'zip': '61257', 'id': '5', 'address': '3236 Walkers Ridge Way, Burr Ridge'} |

**Example 3**: Using restkey parameter

|  |  |
| --- | --- |
| 1  2  3  4  5 | 1,Hannah,4891 Blackwell Street\, Anchorage\, Alaska,99503  2,Walton,4223 Half and Half Drive\, Lemoore\, California,97401  3,Sam,3952 Little Street\, Akron\, Ohio,93704  4,Chris,3192 Flinderation Road\, Arlington Heights\, Illinois,62677  5,Doug,3236 Walkers Ridge Way\, Burr Ridge,61257 |
| 1  2  3  4  5  6  7  8  9  10 | **import** **csv**  **with** open('customers.csv', 'rt') **as** f:  fields = ['id','name',]  csv\_reader = csv.DictReader(f, fieldnames=fields, restkey='extra', escapechar='**\\**')  **for** row **in** csv\_reader:  **print**(row) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | {'id': '1', 'name': 'Hannah', 'extra': ['4891 Blackwell Street, Anchorage, Alaska', '99503']}  {'id': '2', 'name': 'Walton', 'extra': ['4223 Half and Half Drive, Lemoore, California', '97401']}  {'id': '3', 'name': 'Sam', 'extra': ['3952 Little Street, Akron, Ohio', '93704']}  {'id': '4', 'name': 'Chris', 'extra': ['3192 Flinderation Road, Arlington Heights, Illinois', '62677']}  {'id': '5', 'name': 'Doug', 'extra': ['3236 Walkers Ridge Way, Burr Ridge', '61257']} |

Notice that the address and zip code are now stored as a sequence keyed by value extra.

**Example 4**: Using restval

|  |  |
| --- | --- |
| 1  2  3  4  5 | 1,Hannah,4891 Blackwell Street\, Anchorage\, Alaska,99503  2,Walton,4223 Half and Half Drive\, Lemoore\, California,97401  3,Sam,3952 Little Street\, Akron\, Ohio,93704  4,Chris,3192 Flinderation Road\, Arlington Heights\, Illinois,62677  5,Doug,3236 Walkers Ridge Way\, Burr Ridge,61257 |
| 1  2  3  4  5  6  7  8  9  10 | **import** **csv**  **with** open('customers.csv', 'rt') **as** f:  fields = ['id','name', 'address', 'zip', 'phone', 'email'] *# two extra fields*  csv\_reader = csv.DictReader(f, fieldnames=fields, restkey='extra', restval='NA', escapechar='**\\**')  **for** row **in** csv\_reader:  **print**(row) |

**Expected Output:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | {'id': '1', 'name': 'Hannah', 'email': 'NA', 'phone': 'NA', 'address': '4891 Blackwell Street, Anchorage, Alaska', 'zip': '99503'}  {'id': '2', 'name': 'Walton', 'email': 'NA', 'phone': 'NA', 'address': '4223 Half and Half Drive, Lemoore, California', 'zip': '97401'}  {'id': '3', 'name': 'Sam', 'email': 'NA', 'phone': 'NA', 'address': '3952 Little Street, Akron, Ohio', 'zip': '93704'}  {'id': '4', 'name': 'Chris', 'email': 'NA', 'phone': 'NA', 'address': '3192 Flinderation Road, Arlington Heights, Illinois', 'zip': '62677'}  {'id': '5', 'name': 'Doug', 'email': 'NA', 'phone': 'NA', 'address': '3236 Walkers Ridge Way, Burr Ridge', 'zip': '61257'} |

In this case, we have specified field two extra fields: phone and email. The values for extra fields is provided by the restval argument.

## Writing CSV files with DictWriter()

The DictWriter object writes a dictionary to a CSV file. Its syntax is as follows:

**Syntax:** DictWriter(fileobj, fieldnames, restval='', extrasaction='raise', dialect='excel', \*\*fmtparam)

| **Argument** | **Description** |
| --- | --- |
| fileobj | It refers to the file object |
| fieldnames | It refers to the field names and the order in which they will be written the file. |
| restval | It provides the missing value for the keys which doesn't exist in the dictionary. |
| extrasaction | It controls what action to take if the dictionary contains a key, that is not found in the fieldnames argument. By default, extrasaction is set to raise, which means an exception will be raised in such an event. If you want to ignore the extra values set extrasaction to ignore. |

The DictWriter provides the following three methods to write data.

| **Method** | **Description** |
| --- | --- |
| writeheader() | Writes the header (i.e fieldnames) to the CSV file and returns None. |
| writerow(row) | Writes a single row of data and returns the number of characters written. The row must be a sequence of strings and number. |
| writerows(rows) | Writes multiple rows of data and returns None. The rows must be a sequence. |

Let's take some examples:

**Example 1**:

import csv

header = ['id', 'name', 'address', 'zip']

rows = [

{'id': 1, 'name': 'Hannah', 'address': '4891 Blackwell Street, Anchorage, Alaska', 'zip': 99503 },

{'id': 2, 'name': 'Walton', 'address': '4223 Half and Half Drive, Lemoore, California', 'zip': 97401 },

{'id': 3, 'name': 'Sam', 'address': '3952 Little Street, Akron, Ohio', 'zip': 93704 },

{'id': 4, 'name': 'Chris', 'address': '3192 Flinderation Road, Arlington Heights, Illinois', 'zip': 62677},

{'id': 5, 'name': 'Doug', 'address': '3236 Walkers Ridge Way, Burr Ridge', 'zip': 61257},

]

with open('dictcustomers.csv', 'wt') as f:

csv\_writer = csv.DictWriter(f, fieldnames=header)

csv\_writer.writeheader() # write header

csv\_writer.writerows(rows)

**Expected Output:**

**dictcustomers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip  1,Hannah,"4891 Blackwell Street, Anchorage, Alaska",99503  2,Walton,"4223 Half and Half Drive, Lemoore, California",97401  3,Sam,"3952 Little Street, Akron, Ohio",93704  4,Chris,"3192 Flinderation Road, Arlington Heights, Illinois",62677  5,Doug,"3236 Walkers Ridge Way, Burr Ridge",61257 |

**Example 2**: Using restval

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | **import** **csv**  header = ['id', 'name', 'address', 'zip', 'email'] *# an extra field email*  rows = [  {'id': 1, 'name': 'Hannah', 'address': '4891 Blackwell Street, Anchorage, Alaska', 'zip': 99503 },  {'id': 2, 'name': 'Walton', 'address': '4223 Half and Half Drive, Lemoore, California', 'zip': 97401 },  {'id': 3, 'name': 'Sam', 'address': '3952 Little Street, Akron, Ohio', 'zip': 93704 },  {'id': 4, 'name': 'Chris', 'address': '3192 Flinderation Road, Arlington Heights, Illinois', 'zip': 62677},  {'id': 5, 'name': 'Doug', 'address': '3236 Walkers Ridge Way, Burr Ridge', 'zip': 61257},  ]  **with** open('dictcustomers.csv', 'wt') **as** f:  csv\_writer = csv.DictWriter(f, fieldnames=header, restval="NA")  csv\_writer.writeheader() *# write header*  csv\_writer.writerows(rows) |

**Expected Output:**

**dictcustomers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address,zip,email  1,Hannah,"4891 Blackwell Street, Anchorage, Alaska",99503,NA  2,Walton,"4223 Half and Half Drive, Lemoore, California",97401,NA  3,Sam,"3952 Little Street, Akron, Ohio",93704,NA  4,Chris,"3192 Flinderation Road, Arlington Heights, Illinois",62677,NA  5,Doug,"3236 Walkers Ridge Way, Burr Ridge",61257,NA |

In this case, the value of email field is missing from the dictionaries. As s result, the value of restval will be used for the email field.

**Example 3**: Using extrasaction

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | **import** **csv**  header = ['id', 'name', 'address'] *# notice zip is missing*  rows = [  {'id': 1, 'name': 'Hannah', 'address': '4891 Blackwell Street, Anchorage, Alaska', 'zip': 99503 },  {'id': 2, 'name': 'Walton', 'address': '4223 Half and Half Drive, Lemoore, California', 'zip': 97401 },  {'id': 3, 'name': 'Sam', 'address': '3952 Little Street, Akron, Ohio', 'zip': 93704 },  {'id': 4, 'name': 'Chris', 'address': '3192 Flinderation Road, Arlington Heights, Illinois', 'zip': 62677},  {'id': 5, 'name': 'Doug', 'address': '3236 Walkers Ridge Way, Burr Ridge', 'zip': 61257},  ]  **with** open('dictcustomers.csv', 'wt') **as** f:  csv\_writer = csv.DictWriter(  f,  fieldnames=header,  restval="NA",  extrasaction='ignore' *# ignore extra values in the dictionary*  )  csv\_writer.writeheader() *# write header*  csv\_writer.writerows(rows) |

Here, the dictionary contains an extra key named zip which is not present in the header list. To prevent the exception from being raised we have set extrasaction to ignore.

**Expected Output:**

**dictcustomers.csv**

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | id,name,address  1,Hannah,"4891 Blackwell Street, Anchorage, Alaska"  2,Walton,"4223 Half and Half Drive, Lemoore, California"  3,Sam,"3952 Little Street, Akron, Ohio"  4,Chris,"3192 Flinderation Road, Arlington Heights, Illinois"  5,Doug,"3236 Walkers Ridge Way, Burr Ridge" |

## \*\*\*