CSV Files

1. Introduction

What is a CSV File?

CSV files are plain text files which use specific format to store tabular data. CSV stands for "Comma Separated Values".

- Each line of the file is a data record.
- Each record consists of one or more fields, separated by commas.
- Normally first line of the file gives table header.

```
year,sex,type_of_course,no_of_graduates
1993,Males,Humanities & Social Sciences,481
1993,Males,Mass Communication,na
1993,Males,Accountancy,295
1993,Males,Business & Administration,282
```

Why Uses CSV?

- CSV is a common format for data exchange because it is simple and compact.
- Most relational databases provides tools to import and export CSV files.
- CSV files can be easily opened in Excel.

Python CSV Module

While we could use the built-in `open()` function to work with CSV files in Python, there is a dedicated `csv` module that makes working with CSV files much easier. It contains following built-in functions:

- csv.reader
- csv.writer
- writerow()

In [4]:
```
import csv

csv.*?
```

2. Read CSV Files

Using `csv.reader`

After opening a CSV file, create a `csv.reader` object which returns a iterable object to process CSV data.

- Each record is represented as a list.
- All fields are `string` type.

Exercise:

- Use `csv.reader` to read and print out all rows in `'olympics-medals-sample.csv'`

In [25]:
```
import csv
with open('olympics-medals-sample.csv') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

```
['NOC', 'Country', 'Total', 'Medal']
['USA', 'United States', '2088', 'Gold']
['URS', 'Soviet Union', '838', 'Gold']
['GBR', 'United Kingdom', '498', 'Gold']
['FRA', 'France', '378', 'Gold']
['GER', 'Germany', '407', 'Gold']
['AUS', 'Australia', '293', 'Gold']
```

Question:

Instead of printing out, how do you save all rows in `'olympics-medals-sample.csv'` into a list `data` ?

In [26]:
```
import csv
with open('olympics-medals-sample.csv') as f:
    reader = csv.reader(f)
    data = [row for row in reader]

print(data)
```

```
[['NOC', 'Country', 'Total', 'Medal'], ['USA', 'United States', '2088', 'Gol
d'], ['URS', 'Soviet Union', '838', 'Gold'], ['GBR', 'United Kingdom', '498',
'Gold'], ['FRA', 'France', '378', 'Gold'], ['GER', 'Germany', '407', 'Gold'],
['AUS', 'Australia', '293', 'Gold']]
```

Iterable Objector

For any iterator object, you can use `next()` function to retrieve its next item.

Try Code:

```
obj = iter([1,3,5,7])
print(next(obj))
print(next(obj))
```

In [31]:
```
obj = iter([1,3,5,7])
print(next(obj))
print(next(obj))
```

```
1
3
```

Skip Header Row

Besides using list slicing, you can also use `next()` function to get first row, which is commonly its header.

Exercise:

- From file `'olympics-medals-sample.csv'`, retrieve header and data in separate lists.

In [32]:
```
import csv
with open('olympics-medals-sample.csv') as f:
    reader = csv.reader(f)
    header = next(reader)
    data = [row for row in reader]

print(header)
print(data)
```

```
['NOC', 'Country', 'Total', 'Medal']
[['USA', 'United States', '2088', 'Gold'], ['URS', 'Soviet Union', '838', 'Gol
d'], ['GBR', 'United Kingdom', '498', 'Gold'], ['FRA', 'France', '378', 'Gol
d'], ['GER', 'Germany', '407', 'Gold'], ['AUS', 'Australia', '293', 'Gold']]
```

Optional Keyword Arguments

The `csv.reader()` function only has one required argument, which is the file object, but it has a couple of other optional arguments:

- **delimiter:** This argument specifies which delimiter the writer will use. It defaults to `','`, but you can set it to any other character.
- **quotechar:** This specifies which character will be used for quoting. It defaults to `'"'`
- **escapechar:** This specifies the character that will be used to escape the delimiter if quoting is not being used. It defaults to nothing.

Exercise:

Check out documentation of `csv.reader` function.

In [38]:
```
import csv
csv.reader?
```

Delimiter

The character used to separate values is called a **delimiter**. Apart from comma (`,`), other delimiters include the tab (`\t`), colon (`:`) and semi-colon (`;`) characters.

For tab separated values, it is common to save it with extension `*.tsv` .

Exercise:

- Use `csv.reader` to read file `'olympics-medals-sample.tsv'` ; save both header and data in list.

In [39]:
```
import csv
with open('olympics-medals-sample.tsv') as f:
    reader = csv.reader(f, delimiter='\t')
    header = next(reader)
    data = [row for row in reader]

print(header)
print(data)
```

```
['NOC', 'Country', 'Total', 'Medal']
[['USA', 'United States', '2088', 'Gold'], ['URS', 'Soviet Union', '838', 'Gol
d'], ['GBR', 'United Kingdom', '498', 'Gold'], ['FRA', 'France', '378', 'Gol
d'], ['GER', 'Germany', '407', 'Gold'], ['AUS', 'Australia', '293', 'Gold']]
```

Using csv.DictReader

Rather than deal with a list of individual String elements, `csvDictReader` read CSV data directly into an `OrderedDict` (Ordered Dictionary).

An **`OrderedDict`** is a dictionary subclass that preserves the order that keys were inserted. A regular dict doesn't track the insertion order, and iterating it gives the values in an arbitrary order.

Exercise:

- Use `csv.DictReader()` to read `'olympics-medals-sample.csv'` file
- Save header into a list and data into a list of OrderedDict objects.

In [53]:
```
import csv
with open('olympics-medals-sample.csv') as f:
    reader = csv.DictReader(f)
    header = reader.fieldnames
    print(header)
    for row in reader:
        print('{} won {} {} medals'.format(row['Country'], row['Total'], row['Med
```

```
['NOC', 'Country', 'Total', 'Medal']
United States won 2088 Gold medals
Soviet Union won 838 Gold medals
United Kingdom won 498 Gold medals
France won 378 Gold medals
Germany won 407 Gold medals
Australia won 293 Gold medals
```

3. Write CSV Files

Using `csv.writer`

A `csv.writer` can be used to write a CSV file. The `csv.writer()` function returns a `writer` object that converts the user's data into a delimited string and write to file using its `writerow()` function.

The `newline` argument is set to '' when opening a file which the `csv.writer` will write each row in a line.

Exercise:

- Use `csv.writer` to save following data into a csv file `'sample.csv'` .

```
["Symbol", "Name", "Price (Intraday)"]
["TMVWY", "TeamViewer AG", 21.05]
["AXSM", "Axsome Therapeutics, Inc.", 88.87]
["SAGE", "Sage Therapeutics, Inc.",    53.36]
```

In [5]:
```
import csv

data = [["Symbol", "Name", "Price (Intraday)"],
["TMVWY", "TeamViewer AG", 21.05],
["AXSM", "Axsome Therapeutics, Inc.", 88.87],
["SAGE", "Sage Therapeutics, Inc.",    53.36]]

with open('sample.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    for row in data:
        writer.writerow(row)
```

In [6]:
```
!notepad sample.csv
```

Optional Keyword Arguments

The `csv.writer()` function has only 1 required parameter, the file object. You can also add following optional keyword arguments:

- `delimiter` : This argument specifies which delimiter the writer will use. It defaults to ',', but you can set it to any other character.
- `quotechar` : This specifies which character will be used for quoting. It defaults to ""
- `escapechar` : This specifies the character that will be used to escape the delimiter if quoting is not being used. It defaults to nothing.

The `quoting` argument: this specifies which fields should be quoted, there are a few options:

- `csv.QUOTE_ALL` : All fields will be quoted
- `csv.QUOTE_MINIMAL` : Only fields containing the delimiter or quotechar will be quoted.
- `csv.QUOTE_NONNUMERIC` : The writer will quote all fields containing text and it converts all numbers to float values
- `csv.QUOTE_NONE` : No fields will be quoted, the writer instead escapes delimiters. If you use this value, you also need to provide the escapechar argument.

Try Code:

```
import csv
with open('stock_sample.tsv', 'w', newline='') as file:
    writer = csv.writer(
        file,
        delimiter='\t',
        quotechar='|',
        quoting=csv.QUOTE_ALL
    )
    writer.writerow(['stock', 'price', 'cost', 'profit'])
    writer.writerow(['21', '121.34', '45.34', '76'])
```

In [63]:
```
import csv
with open('stock_sample.tsv', 'w', newline='') as file:
    writer = csv.writer(
        file,
        delimiter='\t',
        quotechar='|',
        quoting=csv.QUOTE_ALL
    )
    writer.writerow(['stock', 'price', 'cost', 'profit'])
    writer.writerow(['21', '121.34', '45.34', '76'])
```

In [69]:
```
!notepad stock_sample.tsv
```

Write Multiple Rows

The `writerows()` function of writer allow you to write 2-dimensional list to a CSV file.

Exercise:

Save following data to a csv file `stock_sample.csv` using `csv.writer` .

```
[['stock', 'price', 'cost', 'profit'], ['21', '121.34', '45.34', '76']]
```

In [70]:
```
import csv
csv_rowlist = [['stock', 'price', 'cost', 'profit'], ['21', '121.34', '45.34', '7
with open('stock_sample.csv', 'w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(csv_rowlist)
```

In [71]:
```
!notepad stock_sample.csv
```

4. Basic CSV File Processing

Load Data into List

Exercise:

Read `'sample-sales-data.csv'` file; save its header into variable `header` and its data into variable `data` .

In [72]:
```
with open('sample-sales-data.csv') as file:
    reader = csv.reader(file)
    header = next(reader)
    data = [row for row in reader]
```

Find Distinct Values

You can use `Set()` to find distinct value of a column.

Exercise:

- List all the companies contained in the file.

In [73]:
```
ls = [r[1]for r in data]
# print(ls[:5])
companies = set(ls)
# companies = {r[1]for r in table}
print(companies)
```

```
{'Acme Coporation', 'Hooli', 'Initech', 'Streeplex', 'Mediacore'}
```

Exercise:

- List all dates which have sale recorded by any company.

In [76]:
```
dates = {r[0] for r in data}
dates = sorted(list(dates))
print(dates[:3])
```

```
['2015-01-01', '2015-01-02', '2015-01-03']
```

Filter Data

The list can be filtered based on condition(s).

- Use `for-loop`
- Use list comprehension

Exercise:

- Find all sales records by company `Initech`
- Print out first 3 records

In [78]:
```
NAME = 1   # index value
name = 'Initech'
result = [r for r in data if r[NAME] == name]
print(result[:3])
```

```
[['2015-01-06', 'Initech', 'Hardware', '-17'], ['2015-01-24', 'Initech', 'Softw
are', '1'], ['2015-01-25', 'Initech', 'Service', '6']]
```

Exercise:

- Find all sales done on date '2015-01-06'

In [80]:
```
DATE = 0   # index
date = '2015-01-06'
result = [r for r in data if r[DATE]==date]
print(result)
```

```
[['2015-01-06', 'Initech', 'Hardware', '-17'], ['2015-01-06', 'Acme Coporatio
n', 'Software', '16']]
```

Validate Numeric Data

Both `isdigit()` and `isnumeric()` can be used to check a string which can be converted to **a positive integer**, e.g. `'1234'` .

- But it will return `False` for either `'-1234'` or `'12.34'`

Try Code:

```
print('1234'.isdigit())
print('-1234'.isdigit())
print('12.34'.isdigit())
```

In [85]:
```
print('1234'.isdigit())
print('-1234'.isdigit())
print('12.34'.isdigit())
```

```
True
False
False
```

To test check if a string can be converted to a float or integer, we can use `float()` function with `try-except`.

Try Code:

```
def is_numeric(num):
    try:
        float(num)
        return True
    except ValueError:
        return False

is_numeric('-123.2323')
```

In [84]:
```
def is_numeric(num):
    try:
        float(num)
        return True
    except ValueError:
        return False

is_numeric('-123.2323')
```

Out[84]:
```
True
```

Compute on Records

You can perform simple data analysis on the data:

- `sum()`, `count()`, `min()`, `max()` etc
- Remember to convert data to numerical value for computation or comparision

Exercise:

- Remove records with invalid Units value
- Find total units of sales on "Hardware"

In [88]:
```
CAT = 2
UNITS = 3

records = [r for r in data if is_numeric(r[UNITS])]
category = 'Hardware'
records = [int(r[UNITS]) for r in records if r[CAT]==category]
total = sum(records)

print(total)
```

```
198
```

5. Open Data

Data.gov.sg

The Singapore government's one-stop portal to publicly-available datasets from 70 public agencies. It is an open repository of data captured by the public sector. It helps people understand the data using visualizations and articles, and provides realtime APIs for developers to use.

https://data.gov.sg/ (https://data.gov.sg/)