Introduction to Computer Programming

Week 7.2: Modules for reading & writing files


```
We use/store data in different formats (.txt, .csv....).

Importing Python modules imports sections of pre-written code.
<br>

`csv`: A Python modules for reading/writing delimited files

We will study how the module can make the file read/write
process easier
```

CSV (and other delimited files)

Delimited file:
Uses a set character (tab, space, vertical bar etc) to separate values.

CSV (comma-seperated-value):
A *delimited* text file that uses a comma to separate values.

The CSV file is a widely used format for storing tabular data in plain text and is supported by software applications e.g. Microsoft Excel, Google Spreadsheet.

	A	B	C	D	E	F	G	H	I
1	sample	moisture	knotratio	treering	Edyn	density	beamheig	Estat	bstrength
2	units	%	-	mm	N/mm2	kg/m3	cm	N/mm2	N/mm2
3	DO1	13.3	0.04	3	14053	675	101	15452	58.4
4	DO2	12	0.16	2.5	20611	474	100	17272	74.35
5	DO3	12.8	0.14	3.88	18846	596	99	18456	49.82
6	DO4	11.7	0.13	2.02	18587	582	100	18940	78.52
7	DO5	12	0.16	2.13	19299	678	100	16864	79.31
8	DO6	12.4	0.04	2.98	21695	595	100	19440	64.34
9	DO7	12.5	0.32	3.67	16523	592	100	16152	58.19
10	DO8	11.5	0.07	3.67	18333	634	101	18480	88.39
11	DO9	13.1	0.19	2.44	18628	592	101	14604	33.02
12	DO10	11.7	0.25	3	15683	540	101	16628	60.28
13	DO11	13.2	0.2	3.75	18496	605	100	18476	91.86
14	DO12	11.9	0.11	1.96	19792	646	101	19212	81.88
15	DO13	12.3	0.03	2.4	20098	618	101	19608	91.02

Python's `csv` module can be used to handle files of this type.

https://docs.python.org/3/library/csv.html (https://docs.python.org/3/library/csv.html)

In [51]:
```
import csv
```

Writing CSV files

Writing files using `write` can be time-consuming.

We need to add delimiters (`','` , `' '` etc) and new line markers (`'\n'`).

In [52]:
```
names = ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
scores = [550, 480, 380, 305, 150]
```

In [53]:
```

with open('sample_data/scores_.csv', 'w') as f:

    # loop through two lists
    for n, s in zip(names, scores):
        f.write(n + ',' + str(s) + '\n')
```

The `csv` module provides a way to write lists to csv files directly.

A `writer` object (in instance of the `writer` class) is created.

The class definition for `writer` is stored `csv` module (.py file).

`writerow` :

- a method belonging to the `writer` class
- input argument is list
- automates adding commas when writing data to .csv file

In [54]:
```
with open('sample_data/scores_.csv', 'w', newline='') as f: # f

    w = csv.writer(f)            # writer object is created

    for n, s in zip(names, scores):
        w.writerow([n, s])
```

Example: Use the `csv` module to write the header and first row of the high score table shown to a .txt file.

As a delimiter other than the default `','` is being used, we must specity this when creating the writer object.

Place	Name	Score
1	Elena	550
2	Sajid	480
3	Tom	380
4	Farhad	305
5	Manesha	150

In [ ]:
```

```

On Windows, `writerow` intriduces an additional blank row between subsequent rows.

To avoid the blank line, pass the additional named argument `newline=''` to the `open` function:

```
with open('sample_data/scores_.csv', 'w', newline='') as f:
# open file in write mode
```

__Example:__ Write the high score table data to a csv file, scores_.csv

`writerows` can be used to write a data structure (containing multiple lists) to the file

In [55]:
```
header = ['place', 'name', 'score']

# data is list of lists
data = [[1, 'Elena', 550],
        [2, 'Sajid', 480],
        [3, 'Tom', 380],
        [4, 'Farhad', 305],
        [5, 'Manesha', 150]
       ]
```

In [56]:
```

```

Appending CSV files

In [57]:
```
import csv

with open('sample_data/scores_.csv', 'a') as f:   # open file i

    w = csv.writer(f)                          # writer object

    w.writerow([6, 'Carl', 100])              # list to row
    w.writerow([7, 'Theo', 105])
    w.writerow([8, 'Mark', 75])
    w.writerow([9, 'Grace', 50])
```

Reading CSV files

Reading text files and converting them to a useful format (list of strings) can be a lengthy process.

In [58]:
```
with open('sample_data/scores_.csv') as f:
    file = list(f)                                          # list o

    lines = [line.split(',') for line in file]              # list o

    L = [[w.strip() for w in words] for words in lines]# remove

    print(L)
```

```
[['place', 'name', 'score'], ['1', 'Elena', '550'], ['2', 'Sajid',
'480'], ['3', 'Tom', '380'], ['4', 'Farhad', '305'], ['5', 'Manesh
a', '150'], ['6', 'Carl', '100'], ['7', 'Theo', '105'], ['8', 'Mar
k', '75'], ['9', 'Grace', '50']]
```

`csv` has a `reader` class to make reading delimited files easier.

In [59]:
```
with open('sample_data/scores_.csv') as f:

    r = csv.reader(f)

    for line in r:      # iterable
        print(line)

    # stream position is now at end of file
    # may be returned to start with f.seek(0)

    f.close()
```

```
['place', 'name', 'score']
['1', 'Elena', '550']
['2', 'Sajid', '480']
['3', 'Tom', '380']
['4', 'Farhad', '305']
['5', 'Manesha', '150']
['6', 'Carl', '100']
['7', 'Theo', '105']
['8', 'Mark', '75']
['9', 'Grace', '50']
```

`reader` and `writer` objects have the same limitations as the file object returned by `open` :

- iterable but not subscriptable
- moves stream position to end of file

It can be useful to convert the reader object to a list of lists:

Example: Read the file scores_.csv and print the third line.

In [ ]:
```
1
```

Reading and writing csv files

The same mode specifiers are used as for .txt files.

A `reader` *and* `writer` object are created.

Example: Print the data in the file sample_data/scores_.csv and add a new entry.

In [ ]:
```
1
```

Summary

Functions imported from modules can shorten processes that are lengthy to produce in pure Python by importing code stored elsewhere.

`csv` : A Python modules for reading/writing delimited files

Further reading

- More ways to read and write files using packages we study later on the unit (e.g. `matplotlib` , `numpy`).
- Explore the `os` module for system-level operations (e.g. creating a new directory in your filesystem) https://docs.python.org/3/library/os.html (https://docs.python.org/3/library/os.html)
- Explore the `Pandas` package: useful for handling spreadsheet-style data https://pandas.pydata.org/docs/getting_started/index.html#getting-started (https://pandas.pydata.org/docs/getting_started/index.html#getting-started)

Extra example : Writing data as columns

Data stored as lists

```
places = [1, 2, 3, 4, 5]
names = ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
scores = [550, 480, 380, 305, 150]
```

Often we want to organise our data in columns, not rows:

	A	B	C
1	place	name	score
2	1	Elena	550
3	2	Sajid	480
4	3	Tom	380
5	4	Farhad	305
6	5	Manesha	150

We can't write a column explicitly in Python, as we would in Excel, we can only write rows.

The CSV file is essentially a text file with commas to separate values.

We re-arrange the data into lists that when written to a file, will arrange the data in columns.

This can be achieved using a loop (+ list comprhension)

[An identical process can be used to to the inverse operation : we can transform imported data arranged in columns into lists so that it's easier to use in the Python program]

Example: Write `places`, `names` and `scores` to columns of a csv file.

In [70]:
```
places = [1, 2, 3, 4, 5]
names = ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
scores = [550, 480, 380, 305, 150]

data = [places, names, scores]

with open('sample_data/scores.csv', 'w') as f: # no gap between

    w = csv.writer(f)

    for i in range(len(places)):
        w.writerow([d[i] for d in data])
        # OR
        #w.writerow([places[i], names[i], scores[i]])

```

Extra example : Using `zip` to convert between rows and columns

Import the data from the file sample_data/scores.csv and generate a row containing the data from each column:

Alternatively, can use `zip` to transpose the data from rows to columns before writing to a CSV file.

Like when using `zip` to iterate through two lists, items from mutiple lists are regrouped elementwise.

In [71]:
```
places = [1, 2, 3, 4, 5]
names = ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
scores = [550, 480, 380, 305, 150]

data = zip(places, names, scores)

print(list(data)) # must be converted to a list to print, itera
```

```
[(1, 'Elena', 550), (2, 'Sajid', 480), (3, 'Tom', 380), (4, 'Farha
d', 305), (5, 'Manesha', 150)]
```

To transpose a list of lists we can use `*`.

This *unpacks* the list (removing the outer brackets).

In [72]:
```
data = [[1, 2, 3, 4, 5],
        ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha'],
        [550, 480, 380, 305, 150]
       ]

data_cols = list(zip(*data)) # must be converted to a list to p

print(data_cols)
```

```
[(1, 'Elena', 550), (2, 'Sajid', 480), (3, 'Tom', 380), (4, 'Farha
d', 305), (5, 'Manesha', 150)]
```

This can then be written to a .csv or .txt file

In [73]:
```
with open('sample_data/scores.csv', 'w', newline='') as f:

    w = csv.writer(f)

    w.writerows(data_cols)
```

In [74]:
```
with open('sample_data/scores.txt', 'w', newline='') as f:

    w = csv.writer(f, delimiter=' ') # specify the delimiter

    w.writerows(data_cols)
```

In [75]:
```
import csv

with open('sample_data/scores.csv') as f:

    r = list(csv.reader(f))  # reader is iterable but not subsc

    header = reader[0]            # choose first row as header
    print(header)

    data = list(zip(*r[1:])) # transpose data EXCLUDING header

    print(data[0])                # place
    print(data[1])                # name
    print(data[2])                # score
```

```
['1', 'Elena', '550']
('2', '3', '4', '5')
('Sajid', 'Tom', 'Farhad', 'Manesha')
('480', '380', '305', '150')
```