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.

A Python modules for reading/writing, importing/exporting data files -> `csv` : working with delimited files

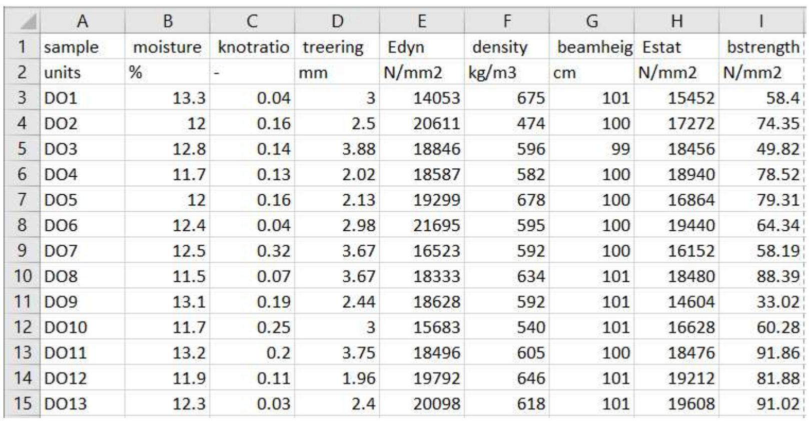
We will study how these imported files can make the file read/write process easier.

CSV (and other delimited files)

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

A delimited file uses a set character (tab, space, vertical bar etc) 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)](https://docs.python.org/3/library/csv.html)

In [73]:

```python
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 [97]:

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

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 straight to a csv file.

The code to do the process of adding delimiters is buried in a file in the `csv` module (.py file(s)).

In [98]:

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

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

    for n, s in zip(names, scores):
        writer.writerow([n, s])        # the writerow function can then be used, input argum
```

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

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

In [1]:

```
import csv

with open('sample_data/scores.csv', 'w') as f:    # open file in write mode

    writer = csv.writer(f)                        # writer object

    writer.writerow(['place', 'name', 'score'])   # list to row

    writer.writerow([1,'Elena', 550])
```

Appending CSV files

In [106]:

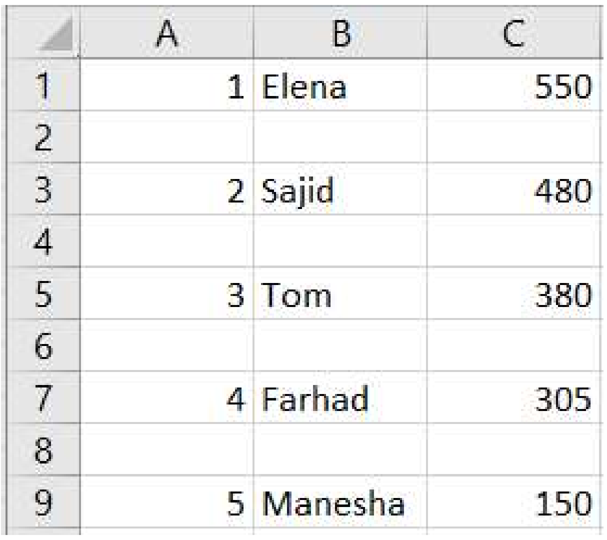
```
import csv

with open('sample_data/scores.csv', 'a') as f:   # open file in append mode

    writer = csv.writer(f)                        # writer object

    writer.writerow([2, 'Sajid', 480])            # list to row
    writer.writerow([3, 'Tom', 380])
    writer.writerow([4, 'Farhad', 305])
    writer.writerow([5, 'Manesha', 150])
```

If you open `scores.csv` you will see there is an additional blank row between subsequent rows:



	A	B	C
1	1	Elena	550
2			
3	2	Sajid	480
4			
5	3	Tom	380
6			
7	4	Farhad	305
8			
9	5	Manesha	150

To avoid the blank line, pass the argument `newline=''` to the `open` function.

Example: Write the high score table data to a csv file

In [101]:

```
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 [77]:

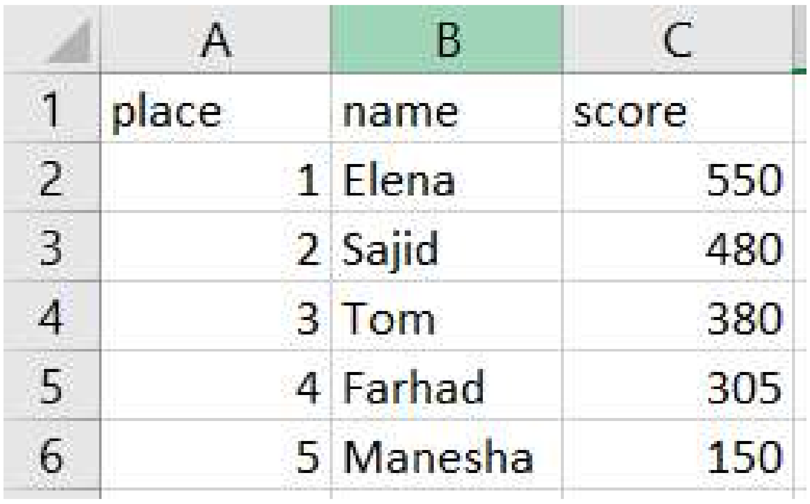
```
import csv

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

    writer = csv.writer(f)

    writer.writerow(header) # write single row

    writer.writerows(data)  # write multiple rows
```



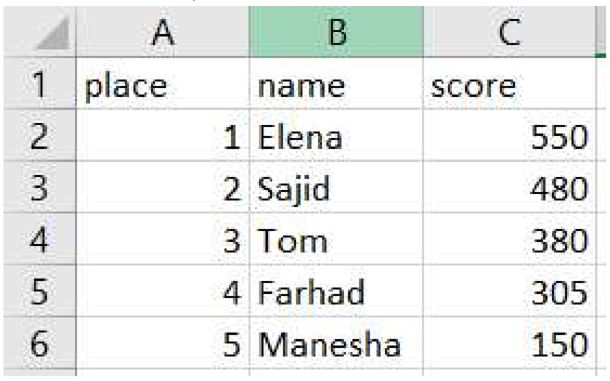
	A	B	C
1	place	name	score
2	1	Elena	550
3	2	Sajid	480
4	3	Tom	380
5	4	Farhad	305
6	5	Manesha	150

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 [2]:

```
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', newline='') as f: # no gap between each line

    writer = csv.writer(f)

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

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 [79]:

```
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, iterate etc
```

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

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

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

In [80]:

```
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 print, iterate etc

print(data_cols)
```

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

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

In [111]:

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

    writer = csv.writer(f)

    writer.writerows(data_cols)
```

In [110]:

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

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

    writer.writerows(data_cols)
```

Reading CSV files

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

In [113]:

```
with open('sample_data/scores.csv') as f:
    file = list(f)                                    # list of strings (lines)
    lines = [line.split(',') for line in file]        # list of list (lines) of strings (w
    print(lines)
    print()
    L = [[w.strip() for w in words] for words in lines]# remove '\n'
    print(L)
```

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

[['1', 'Elena', '550'], ['2', 'Sajid', '480'], ['3', 'Tom', '380'], ['4', 'F
arhad', '305'], ['5', 'Manesha', '150']]
```

`csv` has an equivalent to the `writer` object to make reading delimited files easier.

In [85]:

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

    reader = csv.reader(f)

    for line in reader:      # iterable
        print(line)

    # 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']
```

`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:

In [3]:

```
import csv

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

    reader = csv.reader(f, delimiter=' ') # delimiter must be specified if not default valu

    reader = list(reader)                 # convert to list

    for line in reader:                   # iterable
        print(line)

    print('third line: ', reader[1])      # subscriptable
```

```
['6', 'Ben', '50']
['7', 'Ola', '500']
['8', 'Ria', '460', '']
third line:  ['7', 'Ola', '500']
```

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 [4]:

```
with open('sample_data/scores.csv', 'r+',  newline='') as f:
    reader = csv.reader(f)  # do not convert to list
    writer = csv.writer(f)

    for line in reader:
        print(line)

    # position is at end of file

    writer.writerow([6, 'Lois', 70])

    f.seek(0)

    for line in reader:
        print(line)
```

```
['1', 'Elena', '550']
['2', 'Sajid', '480']
['3', 'Tom', '380']
['4', 'Farhad', '305']
['5', 'Manesha', '150']
['1', 'Elena', '550']
['2', 'Sajid', '480']
['3', 'Tom', '380']
['4', 'Farhad', '305']
['5', 'Manesha', '150']
['6', 'Lois', '70']
```

Summary

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

A Python module for reading/writing, importing/exporting data files -> `csv` : working with delimited files

Further reading

- Will will learn more ways to read and write files using packages we study later on the course (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:

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

In [42]:

```
import csv

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

    reader = list(csv.reader(f))  # reader is iterable but not subscriptable --> convert to

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

    data = list(zip(*reader[1:])) # transpose data excluding header row

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

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

Iterating multiple files

We may want to include multiple files in a program, e.g. iterate over multiple files in a directory.

Python module `os` has many useful functions for system level operations: [https://docs.python.org/3/library/os.html (https://docs.python.org/3/library/os.html)](https://docs.python.org/3/library/os.html)

Example: Print the names of all the files in the directory `sample_data/a_folder`

In [87]:

```
import os

# gets the current directory as a string
current_directory = os.getcwd()

# joins directory names to create a path to the target directory
directory_to_iterate = os.path.join(current_directory, 'sample_data', 'a_folder')

# loops through the files in that directory as a list
for file in os.listdir(directory_to_iterate):
    print(file)
```

```
sample_student_data.csv
signal_data.csv
temperature_data.csv
```