Introduction to Computer Programming

Week 7.2: Modules for reading & writing files

We use/store data in different formats.

Some Python modules for reading/writing, importing/exporting data files:

- `csv` : working with delimited files
- `os` : operating-system level operations e.g. manipulating a file-system

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)

In [18]:

```
import csv
```

Writing CSV files

Steps for writing/appending files using the `csv` library are similar to when using the built-inPytohn functions. Look out for the small differences (steps 2 and 3):

1. open the csv file in `w` (write) or `a` (append) mode using `open / with open`
2. create a CSV writer object using `writer`
3. write data to file using `writerow ( s )` - the row contents are given as a list
4. (close file using `close`)

The same mode specifiers apply.

The `csv` library is useful as it can write mixed data types to a file.

Example: Use the `csv` module to write the 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 [67]:

```
import csv

f = open('sample_data/scores.csv', 'w')

writer = csv.writer(f)

writer.writerow(['place', 'name', 'score']) # item within parentheses should be iterable, e

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

f.close()
```

Appending CSV files

In [68]:

```
import csv

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

    writer = csv.writer(f)

    writer.writerow([2, 'Sajid', 480])
    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.

High score table data:

In [69]:

```
header = ['place', 'name', 'score']

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

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

In [70]:

```
import csv

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

    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

The `csv` module has functions (`DictReader` , `DictWriter`) for exporting/importing Python dictionaries to/from .csv files) https://docs.python.org/3/library/csv.html (https://docs.python.org/3/library/csv.html)

Writing data as columns

The data for the high score table may be stored in the Python programme as:

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

Often we want to oganise 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

But 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 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 [56]:

```
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 [57]:

```
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 [58]:

```
import csv

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

    writer = csv.writer(f)

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

In [59]:

```
import csv

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

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

    writer.writerows(data_cols)
```

Reading CSV files

Steps for reading a CSV are almost the same as for a text file (only step 2 is different):

1. open the csv file in `r` (read) mode (default mode specifier)
2. create a CSV reader object using `reader`
3. The file contents are imported as an *iterable* object i.e. behaves as a list.
   The items of the iterable object are the lines of the file as lists.
   Then items of each list are the comma separated values as strings.
4. (close file using `close`)

In [60]:

```
import csv

f = open('sample_data/scores.csv')

reader = csv.reader(f)

for line in reader:
    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']
```

Import a file with spaces as delimiter

Covert reader to list

In [48]:

```
import csv

f = open('sample_data/scores.txt')

reader = csv.reader(f, delimiter=' ') # delimiter must be specified if other than ,

reader = list(reader)

for line in reader:
    print(line)

f.close()
```

```
['Elena', '550']
['Sajid', '480']
['Tom', '380']
['Farhad', '305']
['Manesha', '150']
['Jen', '100']
['Ben', '50']
['Ola', '500']
['Ben', '50']
['Ola', '500']
```

Converting column data to lists

We may want to use the columns of high score table as lists in our Python programme:

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

The object created using the `reader` method behaves as a list of lists.

We can use `zip` to transpose the data from columns to rows for use in our program.

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

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

In [45]:

```
import csv

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

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

    data = zip(*reader)          # transpose the column data to rows

    data = list(data)            # convert to list to print, iterate, subscript etc

    print(data)
```

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

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')
```

Reading and writing csv files

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

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

In [87]:

```
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)
```

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

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)

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
```

Summary

Some Python modules for reading/writing, importing/exporting data files:

- `csv` : working with delimited files
- `os` : operating-system level operations e.g. manipulating a file-system

Further reading

- 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)
- Will will learn more ways to read and write files using packages we study later on the course (e.g. `matplotlib` , `numpy`).
- Pandas is a useful Python package 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)