Introduction to Computer Programming

Week 7.1: Reading & Writing Files

Reading files : Importing data (e.g. experiment results) into a program

Writing files : Exporting data - storing data outside of the program.
(e.g. output of a calculation)

Built-in Python functions for reading and writing text data files (.txt, .csv, .dat):

- `open()`
- `write()`
- `close()`

Before a file can be read or written to, it must be opened using the `open()` function.

```
open(file_path, mode_specifier)
```

Mode specifier:
An open file can be read, overwritten, or added to, depending on the mode specifier used to open it.

Mode specifier	Read (R)/Write (W)	File must already exist	If no file exists	`write()`	Stream position when opened
`r`	R	Yes	N/A	N/A	start
`w`	W	No	Creates new file	overwrites previous contents	start
`a`	W	No	Creates new file	appends text to end of file	end
`r+`	R+W	Yes	N/A	overwrites previous contents	start
`w+`	R+W	No	Creates new file	overwrites previous contents	start
`a+`	R+W	No	Creates new file	appends text to end of file	end

append: start writing at end of file
write: start writing at beginning of file

Once the file is open, it creates a *file object*.

An object (an instance of a class) can have methods.

Methods are actions or functions that the object is able to perform.

Writing files `w`

We will use the methods:

- `write()`
- `close()`

`write` can be used to write string data to a text file.

```
file = open('my_file.txt', 'w') # mode specifier to write

file.write('hello world')

file.close()
```

A file type that is often used to store tabulated data is the .csv file.

.csv files can be opened in spreadsheet programs like excel

A .csv file is simply a text file, with row items separated (or *delimited*) by commas.

Example:
Write the high score table shown to a new file with the filename scores.txt / scores.csv

Elena	550
Sajid	480
Tom	380
Farhad	305
Manesha	150

In [234]:
```
names = ['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
scores = [550, 480, 380, 305, 150]

file = open('sample_data/scores.txt', 'w')


# loop through two lists
for n, s in zip(names, scores):
    file.write(n + ' ' + str(s) + '\n') # numbers converted to


file.close()
```

Importing a file from a different directory

So far we have considered reading/writing files located within the same directory as the Python program.

Like when importing Python files/modules, often we want to read/write a file located in a different directory.

Downstream file location

`/` is used to indicate a sub-directory downstream of the current location.

```
Documents/
 |
 ├── Folder_1/
 |    └── myScores.txt
 |
 ├── Folder_2/
 |    └── scores.txt
 |
 └── read_write.py
```

Example: Open a downstream file within `read_write.py` :

```
file = open('Folder_1/myScores.txt', 'w')
```

Upstream file location

`../` is used to indicate a location one directory upstream of the current location.

```
Documents/
 │
 ├── Folder_1/
 │    └── read_write.py
 │
 ├── Folder_2/
 │    └── scores.txt
 │
 └── myScores.txt
```

Example: Open an upstream file within `read_write.py` :

```
file = open('../myScores.txt', 'w')
```

Example: Open a file in a different directory at the same level as the directory containing `read_write.py:

```
file = open('../Folder_2/scores.txt', 'w')
```

Closing Files

Why do we need to close a file?

1. Not automatically closed.
2. Saves changes to file.
3. Depending on OS, you may not be able to open a file simultaneously for reading and writing e.g. a program attempts to open a file for writing that is already open for reading

`close` is just a method, belonging to the file object.

The simplest open-close process is shown.

This will erase the contents of / create a new file `file.txt` in the folder `sample_data`

In [235]:
```
open('sample_data/file.txt', 'w').close()
```

Appending files `a`

Start writing at end of file.

Example: Append (add a new entry to the end of) scores.txt so that the table reads

Elena	550
Sajid	480
Tom	380
Farhad	305
Manesha	150
Jen	100

In [236]:
```python
file = open('sample_data/scores.txt', 'a')

file.write('Jen 100\n')

file.close()

```

Reading Files `r`

We use the mode specifier `'r'` to open a file in read mode.

`'r'` can be ommitted as it is the default value for the named argument `mode` .

The file object is:

- iterable (can use for loop etc)
- not subscriptable (cannot index individual elements)

In [237]:
```python
f = open('sample_data/scores.txt', 'r')

# print(f[0])    # not subscriptable

for line in f:  # iterable
    print(line) # each line is a string
```

```
Elena 550

Sajid 480

Tom 380

Farhad 305

Manesha 150

Jen 100
```

The **stream position**:

- can be thought of as a curser.
- goes to end of file when an operation run on file object
- can be returned to start (or any position) with seek

Stream position

Be aware of the *stream position* when opening a file to read.

We can imagine the stream position as the position of the cursor in the file

The stream position is:

- at the *start* of the file after reading.
- at the *end* of the file after reading.

The stream position can be moved to the start of the file (or any other position) with `seek()`.

In [238]:
```
f = open('sample_data/scores.txt', 'r')


for line in f:  # iterable
    print(line) # each line is a string

#f.seek(0)      # stream position goes to end of file when oper
                # can be returned to start with seek

for line in f:
    print(line)

f.close()
```

```
Elena 550

Sajid 480

Tom 380

Farhad 305

Manesha 150

Jen 100
```

If we convert the file object to a list:

- it is subsriptable
- the stream position of the list doesn't need to be reset after each operation
- the stream position of the file object is at the end of the file after the list conversion operation

Example:

Print the list of names and a list of scores from the file `'sample_data/scores.txt'`

Print the name and score of the winner.

In [245]:
```python
f = open('sample_data/scores.txt', 'r')

file = list(f)              # convert to list of strings (line

for line in file:
    print(line)

print('winner: ', file[0])  # subscriptable (no need to return

f.close()
```

```
Sid 50

Jo 20

winner:  Sid 50
```

Reading and Writing with `r+`, `w+`, `a+`

All of these modes can be used to read and write files.

Differences that determine which to use:

- Stream position when opened
- How the stream position when opened affects `write()`

Mode specifier	Read (R)/Write (W)	File must already exist	If no file exists	`write()`	Stream position when opened
r+	R+W	Yes	N/A	overwrites previous contents	start
w+	R+W	No	Creates new file	overwrites previous contents	start
a+	R+W	No	Creates new file	appends text to end of file	end

a+

Example: When we want to read and/or edit (append only).

The stream position is:

- at the *end* when opened (must be moved to the start to read).
- always moved to the *end* before writing when `write` is called (previous contents never overwritten).
- at the *end* after writing.

In [240]:
```
file = open('sample_data/scores.txt', 'a+')

```

r+

Example: When we want to read and/or edit.

The stream position is at the *end* of the file:

- after reading
- before appending
- after appending

In [241]:
```
file = open('sample_data/scores.txt', 'r+')

```

w+

Example: When we want to overwrite file then read

The stream position is:

- at the *start* when opened (previous contents overwritten).
- at the *end* after writing (subsequent lines added using `write` will appended the file, not overwrite previous contents, until file is closed).

Writing *must* happen before reading.

Unlike the `+a` mode specifier `+r` allows writing from anywhere in the file.

Notice the effect of overwriting.

In [242]:
```
file = open('sample_data/scores.txt', 'w+')

```

Editing file contents - a word of warning!

Unlike the `+a` mode specifier `+r` and `+w` allow writing from *anywhere* in the file.

In [243]:
```
file = open('sample_data/scores.txt', 'r+')


# stream position at start of file

file.write('Sid 50\n')                      # append some data
file.write('Jo 20\n')

file.seek(0)
for line in file:                           # read file content
    print(line, end='')


file.close()
```

```
Sid 50
Jo 20
```

Be careful when overwriting:

- `'\n'` inserts a 'new line' character
- any trailing characters

```
Tim 50\nMajid  500
Sid 50\nJo 20\n
```

It is advisable to:

- convert the data you want to edit to an format to a easy-to-edit Python data structure
- overwrite the original file

Example: Edit the file to remove the unwanted line between Jo and Ola.

The file can be erased from a position (function argument) onwards with `truncate()` , default position is current position)

In [244]:
```python
f = open('sample_data/scores.txt', 'r+')

file = list(f)       # convert to list of strings (lines)

del file[2]          # remove element 2

print(file)

f.seek(0)            # go to start of file

for line in file:    # overwrite original file
    f.write(line)

f.truncate()         # remove trailing characters

f.close()

```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-244-e0b21f424292> in <module>
      3 file = list(f)       # convert to list of strings (lines)
      4 
----> 5 del file[2]          # remove element 2
      6 
      7 print(file)

IndexError: list assignment index out of range
```

In [ ]:
```python
file = open('sample_data/scores.txt', 'r+')

for line in file:                   # read file contents
    print(line, end='')

file.close()
```

Automatically closing files

It can be easy to forget to close a file with `close()`

`with open()` can be used instead of `open()` to remove the need for `close()` :

In [ ]:
```
with open('sample_data/scores.txt', 'a') as file:
    file.write('8 Ria 460 \n')

print('next bit of the program') # Code unindents. File automat

```

In [ ]:
```
with open('sample_data/scores.txt', 'r') as file:
    print(file.read())

```

Summary

- Python functions for reading and writing files: `open()` , `read()` , `write()` , `close()`
- The **mode specifier** defines operations that can be performed on the opened file
- Files must always be closed after opening
- Files can be automatically closed by opening with `with open`

In-class demos

Try it yourself

Example 1: Write a high score table stored as two **lists** to a new file with the name scores.csv

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

```

Try it yourself

Example 2: Read the file you just created and print each line

In [ ]:
```

```

Try it yourself

Example 3: Read the file you just created and print the first row

In [ ]:
```
 
```

```
__Example 4:__ Read the file you just created and make a Python list of:
- names
- scores
```

In [ ]:
```
 
```

```
__Example 5:__ change the first row to `'Mia, 700'`:
```

In [ ]:
```
 
```