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)

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

- `open()`
- `read()`
- `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	overwrites previous contents	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

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

As you studied last week, an object (an instance of a class) has methods: actions that an object is able to perform.

Writing files

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()
```

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

Elena	550
Sajid	480
Tom	380
Farhad	305
Manesha	150

In [138]:

```
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 string

file.close()
```

A file, scores.txt will be created.

The file's location is determined when `open` is called.

You can open the file in a text editor to check the contents.

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

In [174]:

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

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

# loop through two lists
for n, s in zip(names, scores):
    file.write(n + ',' + str(s) + '\n') # a comma seperates the values

file.close()
```

Example: Write a high score table stored as a **dictionary** to a new file with the filename scores.txt

In [190]:

```
scores = {'Elena': 550,
          'Sajid': 480,
          'Tom': 380,
          'Farhad': 305,
          'Manesha': 150}

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

# loop through two lists - keys and values of dictionary
for k, v in scores.items():
    file.write(k + ' ' + str(v) + '\n')

file.close()
```

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

The simplest open-close process.

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

In [1]:

```
open('sample_data/file.txt', 'w').close()
```

Appending files

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 [115]:

```
file = open('sample_data/scores.txt', 'a') # mode specifier to append not overwrite

file.write('Jen 100\n')

file.close()
```

Reading Files

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

`'r'` is in fact the default mode specifier so we can omit it.

In [3]:

```
f = open('sample_data/scores.txt', 'r')    # file object returned by `open` is ITERABLE but

#print(f[0])    # not subscriptable

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

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

for line in f:
    print(line)

f.close()
```

```
1 Elena 550

2 Sajid 480

3 Tom 380

4 Farhad 305

5 Manesha 150
```

If we convert the file object to a list:

- it is subsriptable
- the stream position doesn't need to be reset after each operation

Example:

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

Print the name and score of the winnder.

In [176]:

```
f = open('sample_data/scores.txt', 'r')

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

names, scores = [], []
for line in file:              # iterable: collect names and scores using loop
    L = line.split()           # split() converts string (line) to list of strings (words), s
    names.append(L[0])
    scores.append(L[1])

print('winner: ', file[0])     # subscriptable

f.close()

print(names, scores)
```

```
winner:  Elena 550

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

Alternatively, we can also use list comprehension instead of a loop to get the list of names and list of scores.

In [5]:

```
f = open('sample_data/scores.txt', 'r')

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

L = [line.split() for line in file]    # list of lists
print(L)
names = [i[0] for i in L]               # names and scores
scores = [i[1] for i in L]

print(names, scores)
```

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

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

In [182]:

```
file = open('sample_data/scores.txt', 'r+') # We want to read then append

for line in file:                            # read file contents
    print(line)

# stream position at end of file

file.write('Ben 50\n')                       # append some data
file.write('Ola 500\n')

file.close()
```

```
Elena 550

Sajid 480

Tom 380

Farhad 305

Manesha 150

Ben 50

Ola 500
```

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

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

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

- before appending
- after appending
- after over-writing
- after reading

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

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

Example: Open a file, add some data then read new contents

In [14]:

```
file = open('sample_data/scores.txt', 'a+')

file.write('6 Ben 50\n')         # append some data
file.write('7 Ola 500\n')

file.seek(0)                     # GO BACK TO THE START OF FILE

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

file.close()
```

```
Ben 50
Ola 500
6 Ben 50
7 Ola 500
```

Example: Open a file, read the contents, then overwrite the file

In [15]:

```
file = open('sample_data/scores.txt', 'r+')

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

#file.truncate(0)              # ERASE FROM START OF FILE
file.seek(0)                   # GO BACK TO THE START OF FILE

file.write('6 Ben 50\n')       # write some data
file.write('7 Ola 500\n')

file.truncate()                # ERASE FROM CURRENT POSITION

file.close()
```

```
Ben 50
Ola 500
6 Ben 50
7 Ola 500
```

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 [16]:

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

print('next bit of the program') # Code unindents. File automatically closed
```

```
next bit of the program
```

In [17]:

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

```
6 Ben 50
7 Ola 500
8 Ria 460
```

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` :

- using `open` :

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

- using `with open` :

```
with open('Folder_2/scores.txt', 'a') as file:
```

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` using `open` :

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

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

```
with open('../Folder_2/scores.txt', 'a') as file:
```

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`

Extra Example:

Re-order table in scores.txt so it shows the players and their scores ranked in order of highest score to lowest score.

In [191]:

```python
with open('sample_data/scores.txt', 'r+') as f: # read then overwrite
    file = list(f)                          # convert to list of strings (lines)
    L = [line.split() for line in file]     # list of lists
    print(L)
    names = [i[0] for i in L]
    scores = [i[1] for i in L]
    scores = [float(s) for s in scores]     # convert to numerical data
    print(names, scores)

    print(names)
    print(scores)

    # sorted can sort lists, and also zipped lists using order of first list
    sorted_by_score = sorted(zip(scores, names), reverse=True)
    print(sorted_by_score)

    f.truncate(0)                     # erase file

    f.seek(0)                         # go to start


    for item in sorted_by_score:        # write edited table to file
        f.write(item[1] + ' ' + str(item[0]) + '\n')


    f.seek(0)                         # go to start
    print(f.read())                   # read returns contents of file as single string
```

```
[['Elena', '550'], ['Sajid', '480'], ['Tom', '380'], ['Farhad', '305'], ['Ma
nesha', '150']]
['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha'] [550.0, 480.0, 380.0, 305.0,
150.0]
['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha']
[550.0, 480.0, 380.0, 305.0, 150.0]
[(550.0, 'Elena'), (480.0, 'Sajid'), (380.0, 'Tom'), (305.0, 'Farhad'), (15
0.0, 'Manesha')]
Elena 550.0
Sajid 480.0
Tom 380.0
Farhad 305.0
Manesha 150.0
```

Question: What if we wanted to write only the top three scores to the file?