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, metrics recorded during a simulation)

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)	Must already exist	If no file exists	`write()`	Stream position
`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()`

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 [1]:

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

file.write('Elena 550\n' #  \n creates a line break.
          + 'Sajid 480\n'
          + 'Tom 380\n'
          + 'Farhad 305\n'
          + 'Manesha 150\n') # final \n means next entry will begin on new line

file.close()
```

A file, scores.txt will appear in the same directory (folder) as your python program. You can open the file in a text editor to check the contents.

Writing data structures to files

Values that can be represented as a table are likely to be stored in your program as a data structure.

Let's look at some alternative ways to write data stored in a data structure to a .txt file.

Notice that in each case the data is stored as string data type.

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

In [3]:

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

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

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

file.close()
```

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

In [24]:

```
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()
```

The simplest :

In [14]:

```
open('file.txt', 'w').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

Appending files

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

Elena	550
Sajid	480
Tom	380
Farhad	305
Manesha	150

In [15]:

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

file.write('Jen 100\n')

file.close()
```

Reading Files

We will use the methods

- `read()`
- `close()`

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 string data.

In [16]:

```
file = open('sample_data/scores.txt', 'r')
for line in file:
    print(line, end='')
file.close()
```

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

A string can be broken into groups of characters seperated by spaces (or other delimiters) using `split` .

In [17]:

```
names = []
scores = []

file = open('sample_data/scores.txt', 'r')
for line in file:
    L = line.split()
    names.append(L[0])
    scores.append(L[1])
file.close()

print(names, scores)
print(f"The winner is {names[0]}!\n{names[0]}'s score is {scores[0]}")
```

```
['Elena', 'Sajid', 'Tom', 'Farhad', 'Manesha', 'Jen'] ['550', '480', '380',
'305', '150', '100']
The winner is Elena!
Elena's score is 550
```

`read()` : returns the file contents as a single string

In [18]:

```
file = open('sample_data/scores.txt') # read-only mode is default mode
msg = file.read()
print(msg, type(msg))
file.close()
```

```
Elena 550
Sajid 480
Tom 380
Farhad 305
Manesha 150
Jen 100
 <class 'str'>
```

`readlines()` : returns all lines in the file as a list. Each line is a *string* item in the *list* object

In [19]:

```
file = open('sample_data/scores.txt') # read-only mode is default mode

msg = file.readlines()

print(msg[2]) # print 3rd line
```

```
Tom 380
```

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

In [20]:

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

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

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
Jen 100
```

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 onwards with `truncate()` (useful for overwriting a file in `r+` mode).

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

In [21]:

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

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

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

In [ ]:

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

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

file.truncate(0)                # ERASE FROM START OF FILE

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

file.close()
```

In [ ]:

```

```

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 [11]:

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

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

```
next bit of the program
```

In [12]:

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

```
Elena 550
Sajid 480
Tom 380
Farhad 305
Manesha 150
Jen 100
Ben 50
Ola 500
Ria 460
```

Example: Sort the players and scores so they are shown in the file scores.txt from first place to last place

In [13]:

```
with open('sample_data/scores.txt', 'r+') as file: # read then overwrite
    names = []
    scores = []

    for line in file:                    # read
        L = line.split()
        names.append(L[0])
        scores.append(L[1])

    # perform an operation:
    # sorted can sort zipped lists using order of first list
    sorted_by_score = sorted(zip(scores, names), reverse=True)

    file.truncate(0)                     # erase file

    for item in sorted_by_score:         # write
        file.write(item[1] + ' ' + item[0] + '\n')


    file.seek(0)                         # return position to start
    print(file.read())                   # print contents of file
```

```
Elena 550
Ola 500
Ben 50
Sajid 480
Ria 460
Tom 380
Farhad 305
Manesha 150
Jen 100
```

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

Importing a file from a different directory

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

Like when importing Python files/modules, often we want to read/write a file 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`