Reading and Writing text files with Python

- Text files are very commonly used to store information
  - They are the most ‘portable’ types of data files

- Examples of text files include files that are created with a simple text editor, such as Windows Notepad, Notepad++

Working with files - Three basic steps:

- **Opening a file** tells Python which file are we going to work with and how (read/write and are we using it as a text or as a binary file). If a file doesn't exist, it can be created or an exception may be raised, depending on how are we opening it.
- **Reading from or writing to a file** can be done on an open, existing file.
- **Closing a file** ‘tidies up’ and includes flushing the buffer. Closing is an important part of the process, especially when writing to files. Python can close a file automatically, as we shall see later.

File input/output

Python has build in features to work with files. For example, to open a file we use open() and commonly provide the filename and a mode:

```
open(filename, mode)
```

We use a mode for specific operations to :

1. write data to a file
2. read the contents of a file
3. append data to a file

open(filename, mode) – common modes:

Character	Meaning
'w’	open for writing, if it already exists its content will be deleted
'r’	open for reading (default)
'a'	open for writing, appending to the **end** of the file if it exists
‘x’	Will create a file, returns and error if the file exists
Additionally we can add:	
'b'	binary mode
't'	text mode (default)
'+'	open for updating (reading and writing)

https://docs.python.org/3.9/library/functions.html#open

## open a file for writing

#open a file for writing and create it if it doesn’t exist.

```
    f = open('test.txt', 'w')   # write mode
```

- If the file exists, it will be **overridden with an empty file!**

- By default, Python assumes text files or explicitly state a **t**ext file:

```
    f = open('test.txt', 'wt’)
```

Writing to files: A three phase process

```
f = open('test.txt', 'w')
```

- If the file is open, we can use the file object method `write()`. Note: `write()` does not add a new line so we we add these were required.

```
f.write("first line\n")
f.write("second line\n")
```

- Close the file at the end of writing process using `close()` method .

```
f.close()
```

- Variable **f** is called a file object and it contains everything that Python needs to work with the file. Except when opening the file, do we refer to the file by its name!

Reading data

Reading - to read we use the mode **`'r'`**:

```
f = open('test.txt', 'r')   # Read
```

- By default, if the mode is omitted the file is opened in ‘r’ mode (opened for reading).

```
f = open('test.txt')            # Read
```

Reading – read() & readlines()

- read() returns one long string for the whole file

```
    f = open('test.txt', 'r')
    data = f.read()
    f.close()
    print(data)          #first line
                         #second line
```

- readlines() returns a list of strings (each being one line)

```
    f = open('test.txt', 'r’)
    lines = f.readlines()
    f.close()
    print(lines)
             # ['first line\n', 'second line']
```

Reading a text file (continued)

3.Use text file `f` as an iterable object: process one line in each iteration (important for large files):

```
f = open('test.txt', 'r')
for line in f:
        print(line, end='')

f.close()
```

```
                    #first line
                    #second line
```

Appending data

- **Appending** - If you do not want to overwrite the content of the file, you can open a file for appending with mode **'a'**.

```
f = open('test.txt', 'a')  # append mode
f.write('third line\n’)
f.write('fourth line\n’)
f.close()
```

Opening and *automatic* file closing through context manager

- Python provides *context managers* that we use using `with`.

```
with open('test.txt', 'r') as f:
        data = f.read()

print(data)
                    #first line
                    #second line
                    #third line
                    #fourth line
```

- If we use the file context manager, it will **close the file automatically** (when the control flows leaves the indented block).
- This method is recommended when confident with using Python with files.

Example of two methods to close file

- Typical code fragment:

```
f = open('test.txt', 'r')
lines = f.readlines()
f.close()
```

- `lines` is a list of strings, each representing one line of the file.

- Equivalent example using the context manager:

```
with open('test.txt', 'r') as f:
    lines = f.readlines()
```

What if the file does not exist?

- A FileNotFoundError results if the file we try to read does not exist:

```
f = open('test.txt', 'r')
```

- Python has a `os` (operating system) module that we can import. Returns a Boolean - `os.path.exists()`

```
import os
if os.path.exists('test.txt'):
    f = open('test.txt', 'r')
    lines = f.readlines()
    f.close()
```

Accessing parts of the returned list of strings

- `lines` is a list of strings, each representing one line of the file.

```
f = open('test.txt', 'r')

lines = f.readlines()

f.close()

print(lines)


#['first line\n', 'second line\n', 'third
line\n', 'fourth line\n’]
```

Accessing parts of file – method 1

- `lines` contains a list of strings, each representing one line of the file.

```
['first line\n', 'second line\n', 'third line\n', 'fourth line\n’]
```

- We can split() a string (each line of the file) into a list where each word is a list item. Then we can select which list item we want:

```
f = open('test.txt', 'r')
lines = f.readlines()
f.close()
for line in lines:
    words = line.split()
    print(words[0])
```

Output:

```
first
second
third
fourth
```

Accessing parts of file – method 2

- Without readlines() - here we use `f` as an iterable object.
- We can split() each line of the file into a list where each word is a list item.
- Then we can select which list item we want:

```
f = open('test.txt', 'r')
for line in f:
    words = line.split()
    print(words[0])
f.close()
```

Output:

```
first
second
third
fourth
```

Exercise – shopping list

- **Part 1** - Use Python to write the following items (item, quantity, price) to a text file `shopping.txt`(three items per line)

  Bread 1 1.39

  Tomatoes 6 0.26

  Milk 3 1.45

Possible Solution – Part 1 a)

- **Part 1 a)**- Use Python to write the following items (item, quantity, price) to a text file shopping.txt (three items per line)

```
Bread 1   1.39
Tomatoes 6  0.26
Milk 3  1.45
```

```
f = open('shopping.txt', 'w')
f.write("Bread 1 1.39\nTomatoes 6 0.26\nMilk 3 1.45")
f.close()
```

Exercise – shopping list

- **Part 2** - Write a program that reads and computes the total cost per item and prints the following:

```
Item Bread = total 1.39
Item Tomatoes = total 1.56
Item Milk = total 4.35
```

```
fin = open('shopping.txt', 'r’)
lines = fin.readlines()
fin.close()   # close file

#lines  contains a list of strings, each list item is one line of the file:
#['Bread 1 1.39\n', 'Tomatoes 6 0.26\n', 'Milk 3 1.45']

for line in lines:
    words = line.split()
    number = int(words[1])
    cost = float(words[2])
    print(f'Item {words[0]} = total {number * cost}')
```

Exercise – Student Data

- The following file called `students.txt` contains one line for each student in a class. The student's name is the first thing on each line, followed by some exam scores. The number of scores might be different for each student.

```
joe 10 15 20 30 40
bill 23 16 19 22
sue 8 22 17 14 32 17 24 21 2 9 11 17
grace 12 28 21 45 26 10
john 14 32 25 16 89
```

- Create that text file.

Exercise – Student Data

- Create that text file.

```
f = open('students.txt', 'w')
f.write('joe 10 15 20 30 40\n’)
f.write('bill 23 16 19 22\n’)
f.write('sue 8 22 17 14 32 17 24 21 2 9 11 17\n')
f.write('grace 12 28 21 45 26 10\n')
f.write('john 14 32 25 16 89\n')
f.close()
```

Exercise – Student Data

- Using the text file `students.txt` write a program that prints out **the names of students that have more than six quiz scores**.

Exercise – Student Data

- Using the text file `students.txt` write a program that prints out **the names of students that have more than six quiz scores**.

Possible Solution – Student Data

```
f = open('students.txt', 'r')

for aline in f:
    items = aline.split()
    if len(items[1:]) > 6:
        print(items[0])

f.close()
```

Summary

- open() - function takes two parameters; *filename*, and *mode*.
- We looked at examples using the following modes:
  - ”r” – opens file for reading
  - “a” - Opens a file for appending
  - “w” - Opens a file for writing
- close() - When you are finished with a file, you should close it.
- read() – Can be used to read the entire file as a single string.
- readlines() - returns a list of , each list item representing one line of the file.
- write() - writes specified text to the file (used with ‘a’ or ‘w’)

- The split() method **divides a string into a list**. Useful here with readlines()