Lecture 2.3 : Text files

Introduction

- Our Python programs must be able to save their data to the hard disk. This is *persistent* storage i.e. data saved to the hard disk survives a reboot (unlike data stored in RAM).
- Our Python programs also need to be able to retrieve data from files on the hard disk.
- Our programs will, for now, save their data in and retrieve their data from *text files*. (While text files are human-readable, *binary files* are not. We may cover them later.)
- File processing entails the following steps:

  1. **Open the file:** This step initialises a *file object* that acts as a link between the program and the file on the disk. All subsequent file operations are invoked on the file object. (A file object is sometimes referred to as a *file descriptor* or *stream*.)
  2. **Read and/or write the file:** This is where the work is done. Through the file object the on-disk contents of the file will be read and/or written.
  3. **Close the file:** This step finalises the file on the disk and unlinks the file object from the program.

Reading from `stdin` versus reading from a file

- Up until now we have been reading from `stdin`.
- You can think of `stdin` as a file **that is automatically opened for you** i.e. you do not have to open it in order to read from it.
- When you see a program invoked as follows the program is reading from `stdin` and you will not have to open any files.

```
$ python3 program.py < input.txt
```

- However, when you see a program invoked as shown below the program will have to open the file whose name is supplied in `argv[1]`.

```
$ python3 program.py input.txt
```

Our sample text file

- The file whose contents we will process is called `results.txt`.

```
$ cat results.txt
Mary Connolly 76
John Paul Jones 44
Fred Higgins 30
Laura Timmons 57
Fernandinho 22
```

Opening and closing a file

- Below we open a file for reading.
- Other modes in which a file can be opened include `w` for writing (warning: if the file already exists when opened for writing it will be truncated i.e. its contents deleted) and `a` for appending (additions to the file will follow any existing contents).
- When we specify a file name in the call to `open()` Python will look in the same directory as the program to find the file.
- If we wish to open a file that is not in the same directory as our program we need to supply a path to the file e.g. `f = open(r'/tmp/results.txt', 'r')`. (The `r` indicates this is a *raw string* and prevents characters such as `/` from taking on any special meaning the Python interpreter might ordinarily assign them.)

```python
#!/usr/bin/env python3

import sys

def main():

    # Open a file for reading only. If the open succeeds (why might it fail?)
    # it returns a file object (that we assign to f).
    f = open(sys.argv[1], 'r')

    # Read in the entire file contents. Reading in the entire contents might
    # not be a good idea. Why not?
    contents = f.read()

    # Display the contents
    print(contents)

    # Close the file
    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 file_v01.py results.txt
Mary Connolly 76
John Paul Jones 44
Fred Higgins 30
Laura Timmons 57
Fernandinho 22
```

Reading a file

- There are several methods available to a Python programmer for accessing the contents of a file. The most basic is `read()` which we saw above.
- A variant on `read()` is `readlines()`. While `read()` causes the entire contents of the file to be read, `readlines()` returns a list of strings, with each element of the list being a line from the file:
- A potential drawback to `read()` and `readlines()` is that they read in *the entire file contents* and store them in memory. If the file is large this might not be the most efficient use of resources.

```
#!/usr/bin/env python3

import sys

def main():

    f = open(sys.argv[1], 'r')

    contents = f.readlines()

    print(contents)

    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 file_v02.py results.txt
['Mary Connolly 76\n', 'John Paul Jones 44\n', 'Fred Higgins 30\n', 'Laura Timmons 57\
```

- A sometimes better alternative it to process the file line-by-line. We can read in the contents of a file one line at a time with `readline()` (when we reach the end of the file `readline()` sets `line` to the empty string).

```
#!/usr/bin/env python3

import sys

def main():

    f = open(sys.argv[1], 'r')

    line = f.readline()

    # Repeat until there is nothing left to read
    while line:
        print(line.strip())
        line = f.readline()

    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 file_v03.py results.txt
Mary Connolly 76
John Paul Jones 44
Fred Higgins 30
Laura Timmons 57
Fernandinho 22
```

- Often the most convenient way, however, to read a file line-by-line is to use an *iterator*. This approach is similar to using `readline()` but requires less code as an explicit check for the end of the file is not required (the iterator handles that). This is the least error-prone approach (and therefore the one you should prefer whenever appropriate):

```python
#!/usr/bin/env python3

import sys

def main():

    f = open(sys.argv[1], 'r')

    for line in f:
        print(line.strip())

    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 file_v04.py results.txt
Mary Connolly 76
John Paul Jones 44
Fred Higgins 30
Laura Timmons 57
Fernandinho 22
```

File processing

- Each line of `results.txt` consists of a student name and mark. Let's write a program that reads each line from `results.txt` and prints out whether the student in question has passed (or not).
- We want to read in each line, extract the mark and student name, and print `passed` if the mark is 40+ and `failed` otherwise.
- The only difficulty is in extracting the name and exam mark from the line. Although a student's name may consist of a variable number of tokens we can take advantage of the fact that there is a single mark at the end of each line.

```python
#!/usr/bin/env python3

import sys

def main():

    f = open(sys.argv[1], 'r')

    for line in f:
        tokens = line.strip().split()
        mark = int(tokens[-1])
        name = ' '.join(tokens[:-1])

        if mark >= 40:
            result = 'passed'
        else:
            result = 'failed'

        print(f'{name} {result} with a mark of {mark}')

    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 file_v05.py results.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Fred Higgins failed with a mark of 30
Laura Timmons passed with a mark of 57
Fernandinho failed with a mark of 22
```