THE UNIVERSITY OF WESTERN AUSTRALIA

Lecture 14
File Processing Part II

Objectives

- To understand the basic text file processing concepts in Python.
- To learn how to read and write text files in Python and string formatting.
- Take an example to read data from a file, process it and write in a file.
- To learn how to process the file using `with` statement

Revision: File Processing in Python

- Working with text files in Python
  - *Associate a disk file with a file object using the open function*

    <filevar> = `open(`<name>, <mode>`)`

  - <name> *is a string with the actual file name on the disk. The* <mode> *can be* `'r'`, `'w'` *or* `'a'` *depending on whether we are reading, writing or appending the file. You can also add* `'+'` *in mode to mention both read and write operations.*

  - `infile = open("numbers.dat", "r")`

Revison: File Methods

- <file>.`read()` – returns the entire remaining contents of the file as a single (possibly large, multi-line) string. Watch out for final `\n`

- <file>.`readline()` – returns the next line of the file. This is all text up to *and including* the next newline character

- <file>.`readlines()` – returns a list of the remaining lines in the file. Each list item is a single line including the newline characters.

Revision: File reading

```
# printfile.py
#     Prints a file to the screen.

def main():
    fname = input("Enter filename: ")
    infile = open(fname,'r')
    data = infile.read()
    infile.close()
    print(data)
```

- First, prompt the user for a file name
- Open the file for reading
- The file is read as one string and stored in the variable data

Revision: File reading using loop

- Python treats the file itself as a sequence of lines!

```
infile = open(someFile, "r")
for line in infile:
   # process the line here
infile.close()
```

- Most efficient way to read through (and process) file
  - *Multiple calls to readline() is inefficient*

Revision: File writing

- Opening a file for writing prepares the file to receive data
- If you open an existing file for writing, you wipe out the file's contents. If the named file does not exist, a new one is created.

```
outfile = open("mydata.out", "w")
outlife.write(<string>)
```

May use `writelines()` for writing sequence (list) of strings

Example Program: Batch Usernames

- Batch mode processing is where program input and output are done through files (the program is not designed to be interactive)
  - *Real strength of Python of many applications. GUI is fine for small number of cases, but need automation for larger number.*
- Let's create usernames for a computer system where the first and last names come from an input file.

Example Program: Batch Usernames

```
# userfile.py
#    Program to create a file of usernames in batch mode.

def main():
    print ("This program creates a file of usernames from")
    print ("a file of names.")

    # get the file names
    infileName = input("Which file are the names in? ")
    outfileName = input("Where should the usernames go? ")

    # open the files
    infile = open(infileName, 'r')
    outfile = open(outfileName, 'w')
```

Example Program: Batch Usernames

```
# process each line of the input file
 for line in infile:
     # get the first and last names from line
     first, last = line.split()
     # create a username
     uname = (first[0]+last[:7]).lower()
     # write it to the output file
     outfile.write(uname)

 # close both files
 infile.close()
 outfile.close()

 print("Usernames written to”, outfileName)
```

Example Program: Batch Usernames

- Things to note:
  - *It's not unusual for programs to have multiple files open for reading and writing at the same time. However, if a file is no longer needed, close it as there is a limit to number of open files.*
  - *The* `lower` *method is used to convert the names into all lower case, in the event the names are mixed upper and lower case, e.g., de Witt.*

File processing using `with`

- We can think of files as effectively being one (potentially very long) string, stored on disk.
- To use a file, we need to "open" it – this associates the physical data stored on the disk, with a Python object which we can think of as being connected to that physical data.
- Once the file has been opened, we may want to read data from it, or write data to it; and when we are done, we should "*close*" the file.
- Closing the file means Python can recycle any resources being used to manage the file, and ensures all data has been written to it. Failing to close a file can result in data loss.

File processing using `with`

- To avoid data loss, we will try and avoid having to remember to close files ourselves; instead, we'll get Python to remember for us. We do this using Python's "`with`" statement.
- It's used as follows:

```
with open('my_file.txt') as myfile:
    # do things with myfile
```

- Beneath the `with` statement is a block of code we wish to run, which makes use of the file; and when the statements in that block have finished running, the file will be closed for us automatically.

Example: Batch Usernames using `with`

```
# userfile_with.py
#     Program to create a file of usernames in batch mode.
#     Program uses with statement to do file processing.

def main():
    print ("This program creates a file of usernames from")
    print ("a file of names.")

    # get the file names
    infileName = input("Which file are the names in? ")
    outfileName = input("Where should the usernames go? ")

    # initializing a string which accumulates all usernames
    usernames = ""
```

Example Program: Batch Usernames

```
  # process each line of the input file
with open(infileName, 'r') as infile:
    for line in infile:
        # get the first and last names from line
        first, last = line.split()
        # create a username
        uname = (first[0]+last[:7]).lower() + "\n"
        # append it to the string of all usernames
        usernames += uname

# writing usernames in the output file
with open(outfileName, 'w') as outfile:
    print(usernames, file=outfile)
    #above line is similar to outfile.write(usernames)
```

Summary

- We learned how to read files using `with` statement.
- We learned how to write into files using `with` statement.
- We solved an example using two different file processing statements.