COMP2152 LAB MANUAL

OPEN SOURCE DEVELOPMENT

This booklet will help the reader understand the concepts, principles, and implementation of the Python programming language. By the end of the booklet, the reader will be able to code comfortably in Python.

Prepared by Ben Blanc

TABLE OF CONTENTS

CHAPTER 10

WORKING WITH FILES

OPENING FILE

Create an empty txt file named **myfile.txt** in the same directory as your python project.

OPEN() METHOD

The structure for opening a file using **the open()** method is:

open(filename, mode)

Filename is the name of the file including extension. The mode is the operation that you will be executing on the file

Mode	Description	Symbol
Read	Open the file for reading. Exception is raised if file not found	r
Write	Open the file for writing. If file not found, it is created. If file found, contents are overwritten	w
Append	Open the file for writing to the end of the file. If file not found, it is created.	a
Read and Write	Open the file for reading and writing. Exception is raised if file not found	r+
Write and Read	Same is W mode but allows read operation	w+
Append and Read	Same as A mode but allows for reading	a+

CHAPTER 10 AT A GLANCE

In this chapter you will learn how to read and write to text and csv files.

```
filestream_1 = open(file,'r')
filestream_2 = open(file,'w')
filestream_3 = open(file,'a')
filestream_4 = open(file,'r+')
filestream_5 = open(file,'w+')
filestream_6 = open(file,'a+')
```

WITH KEYWORD

The structure for opening a file using the with keyword is:

with open(filename, mode) as variable_identifier :

statements()

The benefit of the with keyword is that the file steam is closed after the statement block

```
file = "myfile.txt"

with open(file, 'r') as my_file_stream:
    print("Statements with file", str(my_file_stream))

print("Statements outside file")
```

Closing A File

To close a file, use the close() method

```
filestream_6.close()
filestream_5.close()
filestream_4.close()
filestream_3.close()
filestream_2.close()
```

Read File

Open the txt file named **myfile.txt** in your python project.

Add the following content into the text file (3 lines)

```
Hello There
From A File
This is Line 3
```

READING ALL LINES

To read all lines of the file, use the **read()** method. You can output the contents directly or store the contents in a variable. Read returned data type of the **read()** method is string

```
file = "myfile.txt"
with open(file, 'r') as my_file:
    content = my_file.read()
    print(content)
    print("Data type of <content> variable is", type(content))
```

When you use the **read()** method, the pointer of the file is at the end of the file. That means calling on the **read()** method again will result in no content.

However, opening the file again will put the pointer back to the start of the file.

```
file = "myfile.txt"
with open(file, 'r') as my_file:
    print(my_file.read())
```

The example below does not use the with keyword

```
filestream = open(file,'r')
content = filestream.read()
print(content)
filestream.close()
```

READING LINE BY LINE

To read a file line by line, use the **readline()** method. When you use the **readline()** method, the pointer reads the text of the file until it readings a newline character, at which point it stops reading and returns the text. Calling on the **readline()** method again will continue the process until the end of the file is reached.

```
file = "myfile.txt"
with open(file, 'r') as my_file:
    print("First line is:",my_file.readline())
    print("Second line is:",my_file.readline())
    print("Third line is:",my_file.readline())
    print("Fourth line doesn't exist",my_file.readline())
```

Using a loop would be more efficient

```
file = "myfile.txt"
with open(file, 'r') as my_file:

    while True :
        line = my_file.readline()
        if len(line) > 0 :
            print(line, end='')
        else :
            break
```

```
filestream = open(file,'r')

while True:
    line = filestream.readline()
    if len(line) > 0:
        print(line, end='')
    else:
        break

filestream.close()
```

READ FILE LINES AS LIST

To stores all the lines of a file as a list, use the **readlines()** method. It returns all the lines of the file as a list. The pointer is at the end of the file after executing this method

```
file = "myfile.txt"
with open(file, 'r') as my_file:
    lines = my_file.readlines()
    for current_line in lines :
        print(current_line, end='')
```

```
filestream = open(file,'r')

lines = filestream.readlines()
for current_line in lines:
    print(current_line, end='')

filestream.close()
```

Writing to File

There are two writing modes of writing to files: write and append. There are two methods of writing text to a file: write() and writelines()

WRITE() METHOD

This method writes a string to the file.
The **write** mode, writes a new file or overwrites an existing file.
The **append** mode, appends a new file to a file.

```
file = "myfile1.txt"
line = "This line will be the new content"

with open(file, 'w') as my_file:
    my_file.write(line)
```

```
file = "myfile1.txt"
line = "This line will be the new content"

filestream = open(file,'w')
filestream.write(line)
filestream.close()
```

```
file = "myfile.txt"
line = "\n" + "This line will be appended"

with open(file, 'a') as my_file:
    my_file.write(line)
```

```
file = "myfile.txt"
line = "\n" + "This line will be appended"

filestream = open(file,'a')
filestream.write(line)
filestream.close()
```

WRITELINES() METHOD

This method writes a list to the file.

The **write** mode, writes a new file or overwrites an existing file.

The **append** mode, appends a new file to a file.

```
file = "myfile1.txt"
lines = ["New Line Content 1" + '\n',
         "New Line Content 2" + '\n',
         "Third New Line Content "]

with open(file, 'w') as my_file:
    my_file.writelines(lines)

file = "myfile1.txt"
lines = ["New Line Content 1" + '\n',
         "New Line Content 2" + '\n',
         "Third New Line Content "]

filestream = open(file,'w')
filestream.writelines(lines)
filestream.close()


file = "myfile.txt"
lines = ["\n" + "First appended line" + '\n',
         'Line 2 added' + '\n',
         'Third and last line']

with open(file, 'a') as my_file:
    my_file.writelines(lines)


file = "myfile.txt"
lines = ["\n" + "First appended line" + '\n',
         'Line 2 added' + '\n',
         'Third and last line ']

filestream = open(file,'a')
filestream.writelines(lines)
filestream.close()
```

CHANGING THE FILE POINTER LOCATION

By default, the pointer of the file starts at either the beginning or the end of the file (depending on the mode). To put the pointer to a specific place in the file, use the **seek()** method

seek(byte_position_as_integer)

The **byte_position_as_integer** is the equivalent to the number of characters you'd like to skip minus 1. Since the first character starts at position 0.

The following code outputs the 5th character of a file

```
file = "myfile.txt"
chars_to_skip = 4

with open(file, 'r') as my_file:
    my_file.seek(chars_to_skip)
    print("The 5th character is " \
          "of the file is",
           my_file.read()[0])
```

```
file = "myfile.txt"
chars_to_skip = 4

filestream = open(file,'r')
filestream.seek(chars_to_skip)
print("The 5th character is " \
      "of the file is",
      filestream.read()[0])
filestream.close()
```

To seek to the start of the file use **seek(0)**

```
file = "myfile.txt"
with open(file, 'r') as my_file:
    my_file.seek(0)
```

```
file = "myfile.txt"
filestream = open(file,'r')
filestream.seek(0)
filestream.close()
```

To record the current position of the file pointer, use the **tell()** method

```
file = "myfile.txt"

with open(file, 'r') as my_file:
    my_file.readline()
    #print second line of file
    print(my_file.readline())
    #save file pointer position
    position = my_file.tell()
    #reset file pointer position
    my_file.seek(0)
    my_file.seek(position)
    # print third line of file
    print(my_file.readline())



filestream = open(file,'r')
filestream.readline()
# print second line of file
print(filestream.readline())
# save file pointer position
position = filestream.tell()
# reset file pointer position
filestream.seek(0)
filestream.seek(position)
# print third line of file
print(filestream.readline())

filestream.close()
```

WORKING WITH CSV FILES

Create a csv file named **csvfile.csv** in the same directory as your Python project.

The content of this csv file is (3 lines, the last line is blank)

```
John, Doe, 18, 987
Jane, Dow, 19, 654
```

Read A CSV File

The following is how you would read the content of a csv file:

```
import csv
file = 'csvfile.csv'
with open(file, 'r') as my_file :
    reader = csv.reader(my_file)
    rowCount = 0
    for row in reader :
        rowCount+=1
        print("Row", rowCount, "Data:")
        columnCount=0
        for column in row :
            columnCount+=1
            print("Row", rowCount,"Column", columnCount, "=", column)
```

```
import csv
file = 'csvfile.csv'
filestream = open(file,'r')
reader = csv.reader(filestream)
rowCount = 0
for row in reader :
    rowCount+=1
    print("Row", rowCount, "Data:")
    columnCount=0
    for column in row :
        columnCount+=1
        print("Row", rowCount,"Column", columnCount, "=", column)

filestream.close()
```

CSV.READER()

First, we must import the csv module. Then, we use the **reader()** method of the csv module.

csv.reader(csv_file, delimiter=”,” , quotechar= ‘””, quoting=csv.QUOTE_ALL)

- csv_file = csv file to open
- delimiter = the character that separates the columns of the tabular data
- quotechar = what character to use to surround column data
- quoting = when to use quoting in column data. Possible values are:
    - csv.QUOTE_ALL - Quote everything, regardless of type.
    - csv.QUOTE_MINIMAL - Quote fields with special characters
    - csv.QUOTE_NONNUMERIC - Quote all fields that are not integers or floats
    - csv.QUOTE_NONE - Do not quote anything on output

Another example of reading a csv file with additional arguments is below:

```
import csv
file = 'csvfile.csv'
with open(file, 'r') as my_file :
    reader = csv.reader(my_file, delimiter= ' ', quotechar='`', quoting=csv.QUOTE_MINIMAL)
    rowCount = 0
    for row in reader :
        rowCount+=1
        print("Row", rowCount, "Data:")
        columnCount=0
        for column in row :
            columnCount+=1
            print("Row", rowCount,"Column", columnCount, "=", column)
```

```
import csv
file = 'csvfile.csv'
filestream = open(file,'r')
reader = csv.reader(filestream, delimiter=' ', quotechar='`', quoting=csv.QUOTE_MINIMAL)
rowCount = 0
for row in reader :
    rowCount+=1
    print("Row", rowCount, "Data:")
    columnCount=0
    for column in row :
        columnCount+=1
        print("Row", rowCount,"Column", columnCount, "=", column)


filestream.close()
```

Write To A CSV File

The following is how you would write content to a csv file:

```
import csv
file = 'csvfile.csv'
data = [['new', 'row', 20, 543], ['boo','hoo', 21, 432]]

with open(file, 'a') as my_file :
    writer = csv.writer(my_file, lineterminator='\n')
    for row in data :
        writer.writerow(row)
```

```
import csv
file = 'csvfile.csv'
data = [['new', 'row', 20, 543], ['boo','hoo', 21, 432]]

filestream = open(file,'a')
writer = csv.writer(filestream, lineterminator='\n')
for row in data:
    writer.writerow(row)

filestream.close()
```

CVS.WRITER()

First, we must import the csv module. Then, we use the **writer()** method of the csv module.

csv.writer(file, lineterminator='\r\n', delimiter="," , quotechar= '"', quoting=csv.QUOTE_ALL)

The lineterminator parameter is the character that is inserted after each row. Notice the default value for lineterminator is "\r\n" so we must override this parameter so it won't insert two line breaks upon writing our data.

WRITEROW() VS WRITEROWS()

In the **writer()** method, we can use **writerow()** or **writerows(). Writerow()** inserts one row of data (list, tuple), while **writerows()** inserts multiple rows of data(two-dimensional list or tuple)

```
import csv
file = 'csvfile.csv'
data = [['new', 'row', 20, 543], ['boo','hoo', 21, 432]]

with open(file, 'a') as my_file :
    writer = csv.writer(my_file, lineterminator='\n')
    writer.writerows(data)
```

```
import csv
file = 'csvfile.csv'
data = [['new', 'row', 20, 543], ['boo','hoo', 21, 432]]

filestream = open(file,'a')
writer = csv.writer(filestream, lineterminator='\n')
writer.writerows(data)
filestream.close()
```

WRITING DICTIONARY DATA

There will be times you would have to write a dictionary to a csv file. Below is how you would accomplish that task:

```
import csv
file = 'csvfile_dict.csv'

fieldnames = ['first_name', 'last_name','age', 'num']
data = {
    'first_name' :'Jack',
    'last_name': 'Hot',
    'age' : 45,
    'num' :  834
}

with open(file, 'a') as my_file :
    writer = csv.DictWriter(my_file, fieldnames=fieldnames,lineterminator='\n')
    writer.writeheader()
    writer.writerow(data)


import csv
file = 'csvfile_dict.csv'

fieldnames = ['first_name', 'last_name','age', 'num']
data = {
    'first_name' :'Jack',
    'last_name': 'Frost',
    'age' : 45,
    'num' :  834
}

filestream = open(file,'a')
writer = csv.DictWriter(filestream, fieldnames=fieldnames, lineterminator='\n')
writer.writeheader()
writer.writerow(data)
filestream.close()
```

You use the **DictWriter()** method of the csv module that takes the same parameters as the **writer()** method. The argument of fieldnames must be set and must match the dictionary keys passed as to the **DictWriter()**. The **writerheader()** method is optional but is recommended so you have the names of the fields.

An example of passing multiple dictionary values is as follows:

```
import csv
file = "csvfile_multi.csv"

data_1 = {
    'first_name' : "John",
    'last_name' : "Smith",
    'age' : 41,
    'num' : 234
}
data_2 = {
    'first_name' : "Jane",
    'last_name' : "Dice",
    'age' : 47,
    'num' : 382
}

combined = [data_1, data_2]

with open(file, 'a') as my_file :
    writer = csv.DictWriter(my_file, fieldnames=data_1.keys(), lineterminator='\n')
    writer.writeheader()
    writer.writerows(combined)
```

READING DICTIONARY DATA

There will be times you will want to read the rows of a csv file as a dictionary. In those instances, use the **DictReader()** of the csv module:

```
import csv
file = 'csvfile_dict.csv'

with open(file, 'r') as my_file :
    reader = csv.DictReader(my_file)
    for row in reader :
        for column in row :
            print(row[column], end=' ')
        print()
```

The first for loop stores an OrderDictionary collection object that stores the dictionary key-value as a list of tuple pairs.

The second loop iterates over that collection object so we can output the value of the specified row-column data.

We could have accomplished the task above with the following code:

```
import csv
file = 'csvfile_dict.csv'

with open(file, 'r') as my_file :
    reader = csv.DictReader(my_file)
    for row in reader :
        print(row['first_name'], row['last_name']
              , row['age'] , row['num'])
```

But it is significantly more tedious

Optionally, we can pass the fieldnames to our **DictReader()** as a parameter. Doing so will print out the fieldnames before the tabular data is displayed.

```
import csv
file = 'csvfile_dict.csv'
fieldnames = ['first_name', 'last_name','age', 'num']

with open(file, 'r') as my_file :
    reader = csv.DictReader(my_file, fieldnames=fieldnames)
    for row in reader :
        for column in fieldnames :
            print(row[column], end=' ')
        print()
```

WRITING BINARY DATA

You can write binary data to a file using the **dump()** method of the **pickle** module.

dump(data_to_write, file)

```
import pickle

file = 'myfile.bin'
string_data = "hello there in binary"

with open(file, 'wb') as my_file :
    pickle.dump(string_data, my_file)
```

READING BINARY DATA

You can read binary data from a file using the **load()** method of the **pickle** module.

```
import pickle

file = 'myfile.bin'
with open(file, 'rb') as my_file :
    content = pickle.load(my_file)
    print(content)
```