Write and Save Files in Python

Estimated time needed: **25** minutes

Objectives

After completing this lab you will be able to:

- Write to files using Python libraries

We can open a file object using the method write() to save the text file to a list. To write to a file, the mode argument must be set to **w**. Let's write a file **Example2.txt** with the line: **"This is line A"**

```python
# Write line to file
exmp2 = '/Example2.txt'
with open(exmp2, 'w') as writefile:
    writefile.write("This is line A")
```

We can read the file to see if it worked:

```python
# Read file

with open(exmp2, 'r') as testwritefile:
    print(testwritefile.read())
```

We can write multiple lines:

```python
# Write lines to file

with open(exmp2, 'w') as writefile:
    writefile.write("This is line A\n")
    writefile.write("This is line B\n")
```

The method .write() works similarly to the method .readline(), except instead of reading a new line it writes a new line. The process is illustrated in the figure. The different colour coding of the grid represents a new line added to the file after each method call.

You can check the file to see if your results are correct.

```python
# Check whether write to file

with open(exmp2, 'r') as testwritefile:
    print(testwritefile.read())
```

We write a list to a **.txt** file as follows:

```
# Sample list of text

Lines = ["This is line A\n", "This is line B\n", "This is line C\n"]
Lines

# Write the strings in the list to text file

with open('/Example2.txt', 'w') as writefile:
    for line in Lines:
        print(line)
        writefile.write(line)
```

We can verify the file is written by reading it and printing out the values:

```
# Verify if writing to file is successfully executed

with open('/Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

However, note that setting the mode to **w** overwrites all the existing data in the file.

```
with open('/Example2.txt', 'w') as writefile:
    writefile.write("Overwrite\n")
with open('/Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

We can write to files without losing any of the existing data as follows by setting the mode argument to append: **a**. You can append a new line as follows:

```
# Write a new line to text file

with open('/Example2.txt', 'a') as testwritefile:
    testwritefile.write("This is line C\n")
    testwritefile.write("This is line D\n")
    testwritefile.write("This is line E\n")
```

You can verify the file has changed by running the following cell:

```
# Verify if the new line is in the text file

with open('/Example2.txt', 'r') as testwritefile:
    print(testwritefile.read())
```

It's fairly inefficient to open the file in **a** or **w** and then reopen it in **r** to read any lines. Luckily we can access the file in the following modes:

- **r+** : Reading and writing. Cannot truncate the file.

- **w+** : Writing and reading. Truncates the file.
- **a+** : Appending and Reading. Creates a new file, if none exists. You dont have to dwell on the specifics of each mode for this lab.

Let's try out the **a+** mode:

```
with open('/Example2.txt', 'a+') as testwritefile:
    testwritefile.write("This is line E\n")
    print(testwritefile.read())
```

There were no errors but read() also did not output anything. This is because of our location in the file.

Most of the file methods we've looked at work in a certain location in the file. .write() writes at a certain location in the file. .read() reads at a certain location in the file and so on. You can think of this as moving your pointer around in the notepad to make changes at a specific location.

Opening the file in **w** is akin to opening the .txt file, moving your cursor to the beginning of the text file, writing new text and deleting everything that follows. Whereas opening the file in **a** is similar to opening the .txt file, moving your cursor to the very end and then adding the new pieces of text. It is often very useful to know where the 'cursor' is in a file and be able to control it. The following methods allow us to do precisely this -

- .tell() - returns the current position in bytes
- .seek(offset,from) - changes the position by 'offset' bytes with respect to 'from'. From can take the value of 0,1,2 corresponding to the beginning, relative to current position and end

Now lets revisit **a+**

```
with open('/Example2.txt', 'a+') as testwritefile:
    print("Initial Location: {}".format(testwritefile.tell()))

    data = testwritefile.read()
    if (not data):  #empty strings return false in python
            print('Read nothing')
    else:
            print(testwritefile.read())

    testwritefile.seek(0,0) # move 0 bytes from beginning.

    print("\nNew Location : {}".format(testwritefile.tell()))
    data = testwritefile.read()
    if (not data):
            print('Read nothing')
    else:
            print(data)

    print("Location after read: {}".format(testwritefile.tell()) )
```

Finally, a note on the difference between **w+** and **r+**. Both of these modes allow access to read and write methods; however, opening a file in **w+** overwrites it and deletes all pre-existing data.

```
with open('/Example2.txt', 'r+') as testwritefile:
    testwritefile.seek(0,0) #write at beginning of file

    testwritefile.write("Line 1" + "\n")
    testwritefile.write("Line 2" + "\n")
    testwritefile.write("Line 3" + "\n")
    testwritefile.write("Line 4" + "\n")
    testwritefile.write("finished\n")
    testwritefile.seek(0,0)
    print(testwritefile.read())
```

To work with a file on existing data, use **r+** and **a+**. While using **r+**, it can be useful to add a .truncate() method at the end of your data. This will reduce the file to your data and delete everything that follows.

```
with open('/Example2.txt', 'r+') as testwritefile:
    testwritefile.seek(0,0) #write at beginning of file

    testwritefile.write("Line 1" + "\n")
    testwritefile.write("Line 2" + "\n")
    testwritefile.write("Line 3" + "\n")
    testwritefile.write("Line 4" + "\n")
    testwritefile.write("finished\n")
    testwritefile.truncate()
    testwritefile.seek(0,0)
    print(testwritefile.read())
```

Let's copy the file **Example2.txt** to the file **Example3.txt**:

```
# Copy file to another

with open('/Example2.txt','r') as readfile:
    with open('Example3.txt','w') as writefile:
          for line in readfile:
                writefile.write(line)
```

We can read the file to see if everything works:

```
# Verify if the copy is successfully executed

with open('Example3.txt','r') as testwritefile:
    print(testwritefile.read())
```

After reading files, we can also write data into files and save them in different file formats like **.txt, .csv, .xls (for excel files) etc**. You will come across these in further examples

NOTE: If you wish to open and view the `example3.txt` file, download this lab here and run it locally on your machine. Then go to the working directory to ensure the `example3.txt` file exists and contains the summary data that we wrote.

Your local university's Raptors fan club maintains a register of its active members on a .txt document. Every month they update the file by removing the members who are not active. You have been tasked with automating this with your Python skills. Given the file `currentMem`, Remove each member with a 'no' in their Active column. Keep track of each of the removed members and append them to the `exMem` file. Make sure that the format of the original files in preserved. (*Hint: Do this by reading/writing whole lines and ensuring the header remains*) Run the code block below prior to starting the exercise. The skeleton code has been provided for you. Edit only the `cleanFiles` function.

```
#Run this prior to starting the exercise
from random import randint as rnd

memReg = 'members.txt'
exReg = 'inactive.txt'
fee =('yes','no')

def genFiles(current,old):
    with open(current,'w+') as writefile:
        writefile.write('Membership No  Date Joined  Active  \n')
        data = "{:^13}  {:<11}  {:<6}\n"

        for rowno in range(20):
            date = str(rnd(2015,2020))+ '-' + str(rnd(1,12))
+'-'+str(rnd(1,25))

writefile.write(data.format(rnd(10000,99999),date,fee[rnd(0,1)]))


    with open(old,'w+') as writefile:
        writefile.write('Membership No  Date Joined  Active  \n')
        data = "{:^13}  {:<11}  {:<6}\n"
        for rowno in range(3):
            date = str(rnd(2015,2020))+ '-' + str(rnd(1,12))
+'-'+str(rnd(1,25))
            writefile.write(data.format(rnd(10000,99999),date,fee[1]))


genFiles(memReg,exReg)
```

Now that you've run the prerequisite code cell above, which prepared the files for this exercise, you are ready to move on to the implementation.

Exercise: Implement the cleanFiles function in the code cell below.

```
'''
The two arguments for this function are the files:
    - currentMem: File containing list of current members
    - exMem: File containing list of old members

    This function should remove all rows from currentMem containing
'no'
    in the 'Active' column and appends them to exMem.
    '''
def cleanFiles(currentMem, exMem):
    # TODO: Open the currentMem file as in r+ mode
        #TODO: Open the exMem file in a+ mode

        #TODO: Read each member in the currentMem (1 member per row)
file into a list.
        # Hint: Recall that the first line in the file is the header.

        #TODO: iterate through the members and create a new list of
the innactive members

        # Go to the beginning of the currentMem file
        # TODO: Iterate through the members list.
        # If a member is inactive, add them to exMem, otherwise write
them into currentMem



    pass # Remove this line when done implementation


# The code below is to help you view the files.
# Do not modify this code for this exercise.
memReg = 'members.txt'
exReg = 'inactive.txt'
cleanFiles(memReg,exReg)


headers = "Membership No  Date Joined  Active  \n"
with open(memReg,'r') as readFile:
    print("Active Members: \n\n")
    print(readFile.read())

with open(exReg,'r') as readFile:
    print("Inactive Members: \n\n")
    print(readFile.read())
```

The code cell below is to verify your solution. Please do not modify the code and run it to test your implementation of `cleanFiles`.

```python
def testMsg(passed):
    if passed:
       return 'Test Passed'
    else :
       return 'Test Failed'

testWrite = "testWrite.txt"
testAppend = "testAppend.txt"
passed = True

genFiles(testWrite,testAppend)

with open(testWrite,'r') as file:
    ogWrite = file.readlines()

with open(testAppend,'r') as file:
    ogAppend = file.readlines()

try:
    cleanFiles(testWrite,testAppend)
except:
    print('Error')

with open(testWrite,'r') as file:
    clWrite = file.readlines()

with open(testAppend,'r') as file:
    clAppend = file.readlines()

# checking if total no of rows is same, including headers

if (len(ogWrite) + len(ogAppend) != len(clWrite) + len(clAppend)):
    print("The number of rows do not add up. Make sure your final
files have the same header and format.")
    passed = False

for line in clWrite:
    if  'no' in line:
        passed = False
        print("Inactive members in file")
        break
    else:
        if line not in ogWrite:
            print("Data in file does not match original file")
            passed = False
print ("{}".format(testMsg(passed)))
```

Congratulations, you have completed this lesson and hands-on lab in Python.

Author

Joseph Santarcangelo

Other Contributors

Mavis Zhou

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-01-10	2.1	Malika	Removed the readme for GitShare
2020-10-16	1.3	Arjun Swani	Added exercise
2020-10-16	1.2	Arjun Swani	Added section additional file modes
2020-10-16	1.1	Arjun Swani	Made append a different section
2020-08-28	0.2	Lavanya	Moved lab to course repo in GitLab