Introduction to Data File Handling in Python

Data file handling is an important programming aspect for almost all programming languages. We need data to be written, modified, deleted and read from a physical storage like a disk by using a programming language. This is quite helpful in storing data for future use.

Python too provides several features to perform various operations on disk files using built-in funtions. The **open()** methos is the key metghod that is used to open a file on a disk for various operations.

There are four different **modes** for opening a data file

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

We also can open a file using the following modes

- "r+" - Random access in text files (First read and then write)
- "r+b" or "rb+" - Random access in binary files
- "w+" and "w+b" - Similar to above but previous data gets erased

Create a new file

In [32]:

```
# I am changing the working directory for files to make it convinient to watch
import os
os.chdir("d:/datafile")
# Creating an empty file for writing using "x". If the file exists, it will return erro
r
try:
    f = open("data.txt", "x")
except FileExistsError:
    print("The file already exists")
else:
    f.write("I am writing data into my file\n")
    f.close()
    print("Data written in file")
```

```
The file already exists
```

Writing in to existing file

In [14]:

```
f = open("data.txt", "a")
f.write("I am writing more data at the end of file\n")
f.close()
```

Reading from a file

In [15]:

```
f = open("Data.txt", "r")
print(f.read())
f.close()
```

```
I am writing data into my file
I am writing more data at the end of file
```

Opening a file for writing that deletes previous data

You will see that the previously written data in "data.txt" is replaced.

In [16]:

```
f = open("data.txt","w")
f.write("File has new data")
f.close()
f = open("data.txt","r")
print(f.read())
f.close()
```

```
File has new data
```

Reading from file with specific intentions

In [19]:

```
#Return the 5 first characters of the file:
f = open("data.txt","r")
print(f.read(4))
f.close()
```

```
File
```

Reading a line from a file

In [22]:

```
# Let us write a few more lines into data.txt
f = open("data.txt","w")
f.write("India is my Country\n")
f.write("All Indians are my brothers and Sisters\n")
f.write("I love my Country\n")
f.close()
f = open("data.txt", "r")
# reading the first line
print(f.readline())
# reading the next line
print(f.readline())
f.close()
```

```
India is my Country

All Indians are my brothers and Sisters
```

Reading all the lines in a file

Usining **readlines()** will read all the lines along with the newline character in a list

In [31]:

```
# Let us write a few more lines into data.txt
f = open("data.txt","w")
f.write("India is my Country\n")
f.write("All Indians are my brothers and Sisters\n")
f.write("I love my Country\n")
f.close()
f = open("data.txt", "r")
print(f.readlines())
```

```
['India is my Country\n', 'All Indians are my brothers and Sisters\n', 'I
love my Country\n']
```

Iterating through the lines

In [24]:

```
f = open("data.txt", "r")
for x in f:
    print(x)
```

```
India is my Country

All Indians are my brothers and Sisters

I love my Country
```

Reading Words in a line

We are going to use **split()** function in this case. By default split() breaks a line into words based on the space. Howeer we can use any word delimiter to separate the words. You can also observe that the split function creates a list of words in a line.

In [25]:

```
# Let us write a few more lines into data.txt
f = open("data.txt","w")
f.write("India is my Country\n")
f.write("All Indians are my brothers and Sisters\n")
f.write("I love my Country\n")
f.close()
f = open("data.txt", "r")
for line in f:
    word = line.split()
    print(word)
```

```
['India', 'is', 'my', 'Country']
['All', 'Indians', 'are', 'my', 'brothers', 'and', 'Sisters']
['I', 'love', 'my', 'Country']
```

Using with keyword to open files.

It is good practice to use the with keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point. Using with is also much shorter than writing equivalent try-finally blocks:

In [26]:

```
with open("mydata.txt", "w") as f:
    f.write("I have used with to open file.")
with open("mydata.txt","r") as f:
    print(f.read())
```

```
I have used with to open file.
```

How to write other objects in a file

If we want to write a list, tuple or a dictionary ina text file we need to convert them in to string and then write

In [27]:

```
f = open("objects.txt", "w")
l = ["Lion","Lion", "Tiger"]
s = str(l)
f.write(s)
f.close()
f = open("objects.txt","r")
print(f.read())
```

```
['Lion', 'Lion', 'Tiger']
```

Searching information in Text files (Parsing test files) - part 1

In the example below we are going to count the occurance of some information in a text file. The data in this file is written in a number of lines with each line consisting of one number(stored as string). We are going to count the occurance of 10 in the file.

In [46]:

```
f = open("data.txt","w")
f.write("10\n")
f.write("20\n")
f.write("30\n")
f.write("10\n")
f.write("20\n")
f.write("40\n")
f.close()
countTens=0
f = open("data.txt","r")
lines = f.readlines()   # reading all the lines
for data in lines:      # iterating through the lines and acquiring each line in data
    data = data.strip() # used strip() to remove trailing newline charecter
    if int(data)==10:   # the int() function is used to convert the data which is a str
ing to integer
        countTens=countTens+1
f.close()
print("Number of 10s in the file:", countTens)
```

```
Number of 10s in the file: 2
```

Searching information in Text files (Parsing test files) - part 2

In the example below we are going to count the occurance of some information in a text file. The data in this file is written in a single line with numbers delimitted with space. We are going to count the occurance of 10 in the file.

In [47]:

```
f = open("data.txt", "w")
f.write("10 20 30 10 20 10")
f.close()
countTens=0
f = open("data.txt","r")
line = f.readline()
data = line.split()   #the split() function is used to split each word in the line and
 store in a list
for x in data:
    if int(x)==10:
        countTens=countTens+1
f.close()
print("Number of 10s in the file:", countTens)
```

```
Number of 10s in the file: 3
```

Using file pointers seek() and tell()

- The file pointer **seek(offset, from)** is used to go to a perticular location in the file. The **offset** gives how many bytes and **from** gives the location from where it has to go. 0 from beginning, 1 from current and 2 from end
- The file pointer **tell()** is used to get the byte position from begining of the location in the file

In [42]:

```
f = open("numbers.txt","w")
f.write("ABCDEFGHIJ")
f.close()
f = open("numbers.txt","rb")
print("Location of file pointer after opening the file:",f.tell())
f.read()
print("Location of file pointer after reading the whole file:", f.tell())
f.seek(5,0)
print("Location of file pointer after five bytes from beginning:", f.tell())
f.seek(2,1)
print("Location of file pointer after two bytes from current location:", f.tell())
f.seek(-2,2)
print("Location of file pointer before two bytes from end :", f.tell())
```

```
Location of file pointer after opening the file: 0
Location of file pointer after reading the whole file: 10
Location of file pointer after five bytes from beginning: 5
Location of file pointer after two bytes from current location: 7
Location of file pointer before two bytes from end : 8
```

Simple modification of text file

In the example below we are going to replace one pre-determined character(d) with another character(D). We are going to open the file ins read-write-binary mode(**r+b**). The **f.seek(3)** function reaches to the charecter d (4th charecter has an offset of 3 starting from 0). since we have opened the file for writing in binary mode we cannot write the string 'D' directly in to the file. Hence we convert the character to byte using the **str.encode('D')** before writing to the file.

In [62]:

```
f = open("data.txt", "w")
f.write("abcdef")
f.close()
f = open("data.txt","r+b") # opeing the file in binary mode for random access
f.seek(3)
f.write(str.encode('D'))   # Converting the string into byte before writing to disc
f.close()
f = open("data.txt","r")
print(f.read())
f.close()
```

```
abcDef
```

Another example of modification of file using a different approach

In this example we are replacing each 'd' from a text file with 'D'. Here we are using the **replace()** function of string library to our advantage.

```
In [63]:
```

```
f = open("data.txt", "w")
f.write("dbcdefd")
f.close()
f = open("data.txt","r")
data = f.read()
data1=data.replace('d','D')
f.close()
f = open("data.txt", "w")
f.write(data1)
f.close()
f = open("data.txt","r")
print(f.read())
f.close()
```

```
DbcDefD
```

Writing and reading objects to a data file

We are going to use the in-built **pickle module** to **serialize** and **de-serialize** complex objects like a dictionary or a class object for writing and reading from a file. the **pickle.dump()** and **pickle.load()** functions will be used for this. In the following example we are going to write and read a dictionary as it is into a binary file.

```
In [68]:
```

```
import pickle
dout = {'Name':'Ravi', 'Age':18, 'Gender':'M'}
f = open('dict.bin',"wb")  # Opening the file for writing in binary mode
pickle.dump(dout,f)           # Serializing the dictionary and writing in file
f.close()
f = open("dict.bin", "rb")  # Opening the file for reading in binary mode
din = pickle.load(f)        # de-serializing the object and loading the dictionary into
din
print(din)
print(din.keys())
f.close()
```

```
{'Name': 'Ravi', 'Age': 18, 'Gender': 'M'}
dict_keys(['Name', 'Age', 'Gender'])
```

File handling with objects of a class

In the given example below we have created a class and performing the following operations

- Writing an object to a binary file
- Reading all the objcts from the file
- Modifying an object based on a given criteria and writing the object back to the file in same location

Python handles classes in a different way and only the data is stored in the form of a dictionary in the file. The **dict** portion of the object contains the valid info and we are going to store only that dictionary using **pickle.dump()**.

The file modification is being achieved in a different approach altogether as discussed below:

- We are loading all the objects(stored as dictionaries) from the file in a list.
- Then we are traversing the list and loading each dictionry to a temporary object of the class peer iteration.
- Then we are looking into the object for the required trait (marks between 28 and 32)
- If there is a match we are modifying the object and update the dictionary in the list.
- The same file is opened in write mode and all the objects from the array are written back

In [23]:

```
# A python program to perform read, write and modify a binary file
import os
import pickle
os.chdir('D:/datafile')
size=0
# Defining the student class
class Student:
    # Declaring private data members
    __rollno=0
    __name=''
    __marks=0

    # Method to acquire data
    def setdata(self,r,n,m):
        self.rollno=r
        self.name=n
        self.marks=m

    # Method to modify the marks of current object
    def modify(self):
        self.marks=33

    # Method to get the marks of the current object
    def getmarks(self):
        return self.marks

    # Method to display data
    def showdata(self):
        print("Rollno=%d, Name=%s, Marks=%d" % (self.rollno, self.name,self.marks))

# The beginning of data file handling program
print("Type 1 for writing an object to file.")
print("Type 2 to print all the objects in file.")
print("Type 3 to modify the marks.")
choice=int(input("Enter your choice:"))

# Writing an object of student class to a file
if choice==1:
    s = Student() # instantiating an object
    r = int(input("Enter the rollno:"))
    n = input("Enter the Name:")
    m = int(input("Enter the marks:"))
    s.setdata(r,n,m)  # Loading object with data
    f = open("student1.dat","ab")  # opening the datafile for appending binary data
    pickle.dump(s.__dict__,f, pickle.HIGHEST_PROTOCOL) # serializng object and writing
 to file
    f.close()

# Reading and displaying all the students
elif choice==2:
    f = open("student1.dat","rb") # opening the object for reading in binary mode
    x =Student()
    while True:
        try:
            x.__dict__ = pickle.load(f) # reading de-serialized object
            x.showdata() # calling the display method of Student class
        except EOFError:
            break
    f.flush()
    f.close()

# Modifying all the students who have got marks in range 28 and 32 and writing back to
 file
elif choice==3:
    f = open("student1.dat", "rb") # opening the file for random access
    tmp_dict = []   #Creating an empty list to hold dictionaries
    while True:
        try:
            tmp_val = pickle.load(f)  # Reading an object from file
            tmp_dict.append(tmp_val)  # appending the object to the list
        except EOFError:
            break

    y = Student()
    for i in range(len(tmp_dict)):    # Traversing the list of dictionaries
        y.__dict__ = tmp_dict[i]      # loading the read dictionary in to the temporary
object
        if y.getmarks()>=28 and y.getmarks()<33:
            y.modify()
            tmp_dict[i] = y.__dict__    # modifying the dictionary with updated value

    f = open("student1.dat","wb")  # opening the datafile for writing back all the obje
cts

    for i in range(len(tmp_dict)):
        pickle.dump(tmp_dict[i],f, pickle.HIGHEST_PROTOCOL) # serializng object and wri
ting to file

    f.close()
```

```
Type 1 for writing an object to file.
Type 2 to print all the objects in file.
Type 3 to modify the marks.
Enter your choice:2
Rollno=10, Name=Ashok, Marks=33
Rollno=20, Name=Sudha, Marks=50
Rollno=30, Name=Rupa, Marks=33
Rollno=40, Name=Jagan, Marks=33
```

References for further studies

- <a href ="https://www.devdungeon.com/content/working-binary-data-python"> (https://www.devdungeon.com/content/working-binary-data-python">) Binary file handling in python</a>
- <a href ="https://pythonprogramming.net/python-pickle-module-save-objects-serialization/">Using (https://pythonprogramming.net/python-pickle-module-save-objects-serialization/">Using) Pickle module to serialize objects for data file handling</a>
- <a href ="https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/">Writing (https://stackabuse.com/reading-and-writing-json-to-a-file-in-python/">Writing) and reading JSON in files</a>

Tutorial created and compiled by Ashok Sengupta - PGT CS KV No 1 Jalahalli West Bengaluru