UNIT – IV FILES AND DATA BASES [9]

File I/O operations – Directory Operations – Reading and Writing in Structured Files: CSV and JSON – Data manipulation using Oracle, MySQL and SQLite.

Introduction

File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory
When we want to read from or write to a file we need to open it first. When we are done, it needs to be closed, so that resources that are tied with the file are freed.

Hence, in Python, a file operation takes place in the following order.

- Open a file
- Read or write (perform operation)
- Close the file

Opening a File

The key function for working with files in Python is the open() function.

The open() function takes two parameters; *filename*, and *mode*.

There are four different methods (modes) for opening a 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

“ r+ **“** - for both reading and writing

Creating a new file

The new file can be created by using one of the following access modes with the function open(). **x:** it creates a new file with the specified name. It causes an error a file exists with the same name.

a: It creates a new file with the specified name if no such file exists. It appends the content to the file if the file already exists with the specified name.

w: It creates a new file with the specified name if no such file exists. It overwrites the existing file.

Reading a Text File

To open a file for reading it is enough to specify the name of the file. It is not mandatory to specify the mode of operation. Python will assume it to be " **r** " by default

```
f = open("textfile.txt")

for each in f:
   print (each)
f.close()
```

The open command will open the file in the read mode and the for loop will print each line present in the file.
Once all the operations are done on the file, we must close it through our python script using the close() method. Any unwritten information gets destroyed once the close() method is called on a file object.

Content of testfile

Hello World
This is our new text file
and this is another line.
Why? Because we can do it this easily.

To read a file using the python script, the python provides us the read() method. The read() method reads a string from the file.

The syntax of the read() method is given below.

```
fileobj.read(<count>)
```

Here, the count is the number of bytes to be read from the file starting from the beginning of the file. If the count is not specified, then it may read the content of the file until the end.

Python facilitates us to read the file line by line by using a function readline(). The readline() method reads the lines of the file from the beginning, i.e., if we use the readline() method two times, then we can get the first two lines of the file.

Each time you run the method, it will return a string of characters that contains a single line of information from the file.

```
file = open("testfile.txt", "r")
print file.readline()
```

This would return the first line of the file.

If we wanted to return only the third line in the file, we would use this:

```
file = open(“testfile.txt”, “r”)
print file.readline(3)
```

But what if we wanted to return every line in the file, properly separated? You would use the same function, only in a new form. This is called the *file.readlines()* function.

```
file = open(“testfile.txt”, “r”)
print file.readlines()
```

The output you would get from this is:

[‘Hello World’, ‘This is our new text file’, ‘and this is another line.’, ‘Why? Because we can.’]

Writing the file

To write some text to a file, we need to open the file using the open method with one of the following access modes.

a: It will append the existing file. The file pointer is at the end of the file. It creates a new file if no file exists.

w: It will overwrite the file if any file exists. The file pointer is at the beginning of the file.

write() method

The write method writes the specified string to the file.

Syntax

file.write(*byte*)

Example

Open the file with "a" for appending, then add some text to the file:

```
f = open("demofile2.txt", "a")
f.write("See you soon!")
f.close()
```

writelines()

Python file method **writelines()** writes a sequence of strings to the file. The sequence can be any iterable object producing strings, typically a list of strings.

Syntax

file.writelines(*list*)

Example

Open the file with "a" for appending, then add a list of texts to append to the file:

```
f = open("demofile3.txt", "a")
f.writelines(["See you soon!", "Over and out."])
f.close()
```

Using with statement with files

The with statement is useful in the case of manipulating the files. The with statement is used in the scenario where a pair of statements is to be executed with a block of code in between.

The syntax to open a file using with statement is given below.

```
with open(<file name>, <access mode>) as <file-pointer>:
#statement suite
```

The advantage of using with statement is that it provides the guarantee to close the file regardless of how the nested block exits.

It is always suggestible to use the with statement in the case of file s because, if the break, return, or exception occurs in the nested block of code then it automatically closes the file. It doesn't let the file to be corrupted.

Consider the following example.

Example

```
with open("file.txt",'r') as f:
content = f.read();
print(content)
```

Output:

```
Python is the modern day language. It makes things so simple.
```

Sample Programs

Example 1

Program to count the number of lines, words and characters in a given file

```
f = open("list.txt", 'r')
lines = f.readlines()
print(lines)
print("Lines:",len(lines))
words = 0
chars = 0
for line in lines:
   words += len(line.split())
   chars += len(line.strip())
print("Words:", words)
print("Chars:", chars)
f.close()
```

Output

```
['hai students\n', 'good morning\n', 'how are you \n']
Lines: 3
Words: 7
Chars: 35
```

Example 2

Program to read the content of a text file and write into another

```
with open("test.txt") as f:
   with open("out.txt", "w") as f1:
      for line in f:
         f1.write(line)
```

Example 3

Program to merge the contents of two files

```
filenames = ['file1.txt', 'file2.txt', 'file3.txt']
with open('output_file', 'w') as outfile:
   for fname in filenames:
      with open(fname) as infile:
         for line in infile:
            outfile.write(line)
```

Directories in Python

If there are a large number of files to handle in your Python program, you can arrange your code within different directories to make things more manageable.

A directory or folder is a collection of files and sub directories. Python has the os module, which provides us with many useful methods to work with directories

Get Current Directory

We can get the present working directory using the getcwd() method.

```
print(os.getcwd())
```

Changing Directory

We can change the current working directory using the chdir() method.

```
os.chdir('C:\\Python33')
print(os.getcwd())
```

List Directories and Files

All files and sub directories inside a directory can be known using the listdir() method.

```
print(os.listdir())
```

Making a New Directory

We can make a new directory using the mkdir() method.

This method takes in the path of the new directory. If the full path is not specified, the new directory is created in the current working directory.

```
>>> os.mkdir('test')

>>> os.listdir()
['test']
```

Renaming a Directory or a File

The rename() method can rename a directory or a file.

The first argument is the old name and the new name must be supplies as the second argument.

```
>>> os.listdir()
['test']

>>> os.rename('test','new_one')

>>> os.listdir()
['new_one']
```

Removing Directory or File

A file can be removed (deleted) using the remove() method. Similarly, the rmdir() method removes an empty directory.

```
>>> os.listdir()
['new_one', 'old.txt']

>>> os.remove('old.txt')
>>> os.listdir()
['new_one']

>>> os.rmdir('new_one')
>>> os.listdir()
        []
```

However, note that rmdir() method can only remove empty directories.

Comma Separated Values (CSV) files

A CSV file is a type of plain text file that uses specific structuring to arrange tabular data. CSV is a common format for data interchange as it's compact, simple and general. Many online services allow its users to export tabular data from the website into a CSV file. Files of CSV will open into Excel, and nearly all databases have a tool to allow import from CSV file. The standard format is defined by rows and columns data. Moreover, each row is terminated by a newline to begin the next row. Also within the row, each column is separated by a comma.

Sample CSV file

CSV Data

Programming language, Designed by, Appeared, Extension

Python, Guido van Rossum, 1991, .py

Java, James Gosling, 1995, .java

C++, Bjarne Stroustrup,1983,.cpp

Python CSV Module

Python provides a CSV module to handle CSV files. To read/write data, you need to loop through rows of the CSV. You need to use the split method to get data from specified columns.

CSV Module Functions

In CSV module documentation you can find following functions:

- csv.field_size_limit – return maximum field size
- csv.get_dialect – get the dialect which is associated with the name
- csv.list_dialects – show all registered dialects
- csv.reader – read data from a csv file
- csv.register_dialect - associate dialect with name
- csv.writer – write data to a csv file
- csv.unregister_dialect - delete the dialect associated with the name the dialect registry
- **csv.QUOTE_ALL** - Quote everything, regardless of type.
- **csv.QUOTE_MINIMAL** - Quote fields with special characters
- **csv.QUOTE_NONNUMERIC** - Quote all fields that aren't numbers value
- **csv.QUOTE_NONE** – Don't quote anything in output

Read a CSV File

To read data from CSV files, you must use the reader function to generate a reader object.

The reader function is developed to take each row of the file and make a list of all columns. Then, you have to choose the column you want the variable data for.

It sounds a lot more intricate than it is. Let's take a look at this example, and we will find out that working with csv file isn't so hard.

```
#import necessary modules
import csv
with open('data.csv','rt')as f:
  data = csv.reader(f)
  for row in data:
        print(row)
```

Output

```
['Programming language; Designed by; Appeared; Extension']
['Python; Guido van Rossum; 1991; .py']
['Java; James Gosling; 1995; .java']
['C++; Bjarne Stroustrup;1983;.cpp']
```

Read a CSV as a Dictionary

You can also you use DictReader to read CSV files. The results are interpreted as a dictionary where the header row is the key, and other rows are values.

Consider the following code

```
#import necessary modules
import csv

reader = csv.DictReader(open("data.csv"))
for raw in reader:
    print(raw)
```

The result of this code is:

```
OrderedDict([('Programming language', 'Python'), ('Designed by', 'Guido van Rossum'), (' Appeared', ' 1991'), (' Extension', ' .py')])
OrderedDict([('Programming language', 'Java'), ('Designed by', 'James Gosling'), (' Appeared', ' 1995'), (' Extension', ' .java')])
OrderedDict([('Programming language', 'C++'), ('Designed by', ' Bjarne Stroustrup'), (' Appeared', ' 1985'), (' Extension', ' .cpp')])
```

Write CSV File

When you have a set of data that you would like to store in a CSV file you have to use writer() function. To iterate the data over the rows(lines), you have to use the writerow() function.

Consider the following example. We write data into a file "writeData.csv" where the delimiter is an apostrophe.

```
#import necessary modules
import csv

with open('X:\writeData.csv', mode='w') as file:
    writer = csv.writer(file, delimiter=',', quotechar='"', quoting=csv.QUOTE_MINIMAL)

    #way to write to csv file
    writer.writerow(['Programming language', 'Designed by', 'Appeared', 'Extension'])
    writer.writerow(['Python', 'Guido van Rossum', '1991', '.py'])
    writer.writerow(['Java', 'James Gosling', '1995', '.java'])
    writer.writerow(['C++', 'Bjarne Stroustrup', '1985', '.cpp'])
```

Result in csv file is:

```
Programming language, Designed by, Appeared, Extension
Python, Guido van Rossum, 1991, .py
Java, James Gosling, 1995, .java
C++, Bjarne Stroustrup,1983,.cpp
```

DictWriter

The csv module also provides us the DictWriter classes, which allow us to write to files using dictionary objects.

The class DictWriter() works very similarly to the csv.writer method, although it maps the dictionary to output rows. However, be aware that since Python's dictionaries are not ordered, we cannot predict the row order in the output file.

```
import csv
myFile = open('countries.csv', 'w')
with myFile:
   myFields = ['country', 'capital']
   writer = csv.DictWriter(myFile, fieldnames=myFields)
   writer.writeheader()

   writer.writerow({'country' : 'France', 'capital': 'Paris'})

   writer.writerow({'country' : 'Italy', 'capital': 'Rome'})

   writer.writerow({'country' : 'Spain', 'capital': 'Madrid'})

   writer.writerow({'country' : 'Russia', 'capital': 'Moscow'})
```

And finally, running this code gives us the correct CSV output, with labels and all:

OUTPUT

```
country,capital

France,Paris

Italy,Rome

Spain,Madrid

Russia,Moscow
```

JSON

JSON is a standard format for data exchange, which is inspired by JavaScript. Generally, JSON is in string or text format. **JSON** stands for **J**ava**S**cript **O**bject **N**otation. It's common to transmit and receive data between a server and web application in JSON format.

A JSON object contains data in the form of key/value pair. The keys are strings and the values are the JSON types. Keys and values are separated by colon. Each entry (key/value pair) is separated by comma.

The **{** (curly brace) represents the JSON object.

JSON is very similar to Python dictionary. Python supports JSON, and it has an inbuilt library as a JSON.

Import json Module

To work with JSON (string, or file containing JSON object), you can use Python's json module. You need to import the module before you can use it.

import json

Following methods are available in the JSON module

Method	**Description**
dumps()	encoding to JSON objects
dump()	encoded string writing on file
loads()	Decode the JSON string
load()	Decode while JSON file read

Python to JSON (Encoding)

JSON Library of Python performs following translation of Python objects into JSON objects by default

Python	**JSON**
dict	Object

list	Array
str	String
number - int, long	number – int
float	number – real
True	true
False	false
None	Null

Parse JSON in Python

The json module makes it easy to parse JSON strings and files containing JSON object.

Example 1: Python JSON to dict

You can parse a JSON string using json.loads() method. The method returns a dictionary.

```
import json

person = '{"name": "Bob", "languages": ["English", "French"]}'
person_dict = json.loads(person)

# Output: {'name': 'Bob', 'languages': ['English', 'French']}
print( person_dict)

# Output: ['English', 'French']
print(person_dict['languages'])
```

Here, person is a JSON string, and person_dict is a dictionary.

Example 2: Python read JSON file

You can use json.load() method to read a file containing JSON object.
Suppose, you have a file named person.json which contains a JSON object.

```
{"name": "Bob",
"languages": ["English", "Fench"]
```

```
    }
```

Here's how you can parse this file:

```
import json
with open('path_to_file/person.json') as f:
  data = json.load(f)

# Output: {'name': 'Bob', 'languages': ['English', 'Fench']}
print(data)
```

Here, we have used the open() function to read the json file. Then, the file is parsed using json.load() method which gives us a dictionary named data.

Python Convert to JSON string

You can convert a dictionary to JSON string using json.dumps() method.

Example 3: Convert dict to JSON

```
import json

person_dict = {'name': 'Bob',
'age': 12,
'children': None
}
person_json = json.dumps(person_dict)

# Output: {"name": "Bob", "age": 12, "children": null}
print(person_json)
```

Here's a table showing Python objects and their equivalent conversion to JSON.

Python	**JSON Equivalent**
dict	object
list, tuple	array

str	string
int, float, int	number
True	true
False	false
None	null

Writing JSON to a file
To write JSON to a file in Python, we can use json.dump() method.

Example 4: Writing JSON to a file

```
import json
person_dict = {"name": "Bob",
"languages": ["English", "Fench"],
"married": True,
"age": 32
}
with open('person.txt', 'w') as json_file:
json.dump(person_dict, json_file)
```

In the above program, we have opened a file named person.txt in writing mode using 'w'. If the file doesn't already exist, it will be created. Then, json.dump() transforms person_dict to a JSON string which will be saved in the person.txt file.
When you run the program, the person.txt file will be created. The file has following text inside it.

```
{"name": "Bob", "languages": ["English", "French"], "married": true, "age": 32}
```

Creation of Database

SQLite is an in-process library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. It is a database, which is zero-configured, which means like other databases you do not need to configure it in your system.

SQLite engine is not a standalone process like other databases, you can link it statically or dynamically as per your requirement with your application. SQLite accesses its storage files directly.

Creation, Insertion and Selection of table

```
import sqlite3
```

```
conn = sqlite3.connect('test.db')
print("Opened database successfully")
conn.execute('CREATE TABLE COMPANY\
       (ID INT PRIMARY KEY     NOT NULL,\
       NAME           TEXT    NOT NULL,\
       AGE            INT     NOT NULL,\
       ADDRESS        CHAR(50),\
       SALARY         REAL);')
print("Table created successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (1, 'Deepa', 32, 'Coimbatore', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (2, 'Kalai', 25, 'Delhi', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (3, 'Malar', 23, 'Erode', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (4, 'Deepak', 25, 'Salem ', 65000.00 )");
conn.commit()
print("Records created successfully")
cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print("ID = ", row[0])
   print("NAME = ", row[1])
   print("ADDRESS = ", row[2])
   print("SALARY = ", row[3], "\n")
print("Operation done successfully")
conn.close()
```

Updation of Table

```
import sqlite3
conn = sqlite3.connect('test.db')
print("Opened database successfully")
conn.execute('CREATE TABLE COMPANY\
       (ID INT PRIMARY KEY     NOT NULL,\
       NAME           TEXT    NOT NULL,\
       AGE            INT     NOT NULL,\
       ADDRESS        CHAR(50),\
       SALARY         REAL);')
print("Table created successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (1, 'Deepa', 32, 'Coimbatore', 20000.00 )");
```

```
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (2, 'Kalai', 25, 'Delhi', 15000.00 )");
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (3, 'Malar', 23, 'Erode', 20000.00 )");
conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
     VALUES (4, 'Deepak', 25, 'Salem ', 65000.00 )");
conn.commit()
print("Records created successfully")

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print("ID = ", row[0])
   print("NAME = ", row[1])
   print("ADDRESS = ", row[2])
   print("SALARY = ", row[3], "\n")
   conn.execute("UPDATE COMPANY set SALARY = 25000.00 where ID = 1")
conn.commit

print("Total number of rows updated :", conn.total_changes)
cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print("ID = ", row[0])
   print("NAME = ", row[1])
   print("ADDRESS = ", row[2])
   print("SALARY = ", row[3], "\n")
print("Operation done successfully")
conn.close()
```

DELETE Operation

Following Python code shows how to use DELETE statement to delete any record and then fetch and display the remaining records from the COMPANY table.

```
import sqlite3

conn = sqlite3.connect('test.db')
print("Opened database successfully")
conn.execute('CREATE TABLE COMPANY\
       (ID INT PRIMARY KEY     NOT NULL,\
       NAME           TEXT    NOT NULL,\
       AGE            INT     NOT NULL,\
       ADDRESS        CHAR(50),\
       SALARY         REAL);')
```

```
print("Table created successfully")

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (1, 'Deepa', 32, 'Coimbatore', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (2, 'Kalai', 25, 'Delhi', 15000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (3, 'Malar', 23, 'Erode', 20000.00 )");

conn.execute("INSERT INTO COMPANY (ID,NAME,AGE,ADDRESS,SALARY) \
      VALUES (4, 'Deepak', 25, 'Salem ', 65000.00 )");

conn.commit()
print("Records created successfully")

conn.execute("DELETE from COMPANY where ID = 2;")
conn.commit()
print("Total number of rows deleted :", conn.total_changes)

cursor = conn.execute("SELECT id, name, address, salary from COMPANY")
for row in cursor:
   print("ID = ", row[0])
   print("NAME = ", row[1])
   print("ADDRESS = ", row[2])
   print("SALARY = ", row[3], "\n")

print("Operation done successfully")
conn.close()
```