Creating csv file using python program

To create csv file using python program, csv module provides two classes or objects

1. writer
2. Dictwriter

Writer object allows write data inside csv file using sequences/list.
Writer object provides the following methods for writing list/sequences

writerow(sequence)
writerows(sequence)

this method write a list inside csv file

DictWriter object allows to write data inside csv file. This data is represented in dictionary object.
DictWriter provides the following methods

1. Writeheader()
2. Writerow()

Example of creating csv file using writer object

```
import csv
def main():
    f=open("f:\\python6pmaug\\student.csv","a",newline='')
    w=csv.writer(f)
    headerrow=['rollno','name','course']
    w.writerow(headerrow)
    while True:
        rno=int(input("Rollno"))
        name=input("Name")
        course=input("Course")
        data=[rno,name,course]
        w.writerow(data)
        ans=input("Add another student?")
        if ans=="no":
            f.close()
            break
```

```
main()
```

Output:

```
Rollno1
Namenaresh
Coursepython
Add another student?yes
Rollno2
Namekishore
Coursec
Add another student?no
```

Example of writing more than one row

```
import csv
def main():
    f=open("f:\\python6pmaug\\student1.csv","a",newline='')
    w=csv.writer(f)
    studentData=[['rno','name'],
                 [1,'naresh'],
                 [2,'suresh']]
    w.writerows(studentData)
    f.close()

main()
```

Output:

Output is saved inside student1.csv file

Example of creating csv file using DictWriter

```
import csv
def main():
    f=open("f:\\python6pmaug\\employee.csv","w",newline='')
```

```
    dw=csv.DictWriter(f,fieldnames=['empno','ename','salary'])
    dw.writeheader()
    while True:
        eno=int(input("EmployeeNo"))
        ename=input("EmployeeName")
        sal=float(input("Salary"))
        data={'empno':eno,'ename':ename,'salary':sal}
        dw.writerow(data)
        ans=input("Add another employee?")
        if ans=="no":
            f.close()
            break

main()
```

Output
Output is saved inside employee.csv file

JSON file

JSON stands for **J**ava**S**cript **O**bject **N**otation
JSON is a **text format** for storing and transporting data
JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write.
Python provides json module to work with json file. Json module provides encoder and decoder, to convert python object to json object and json objects to python objects.

Extensible JSON encoder for Python data structures.
Supports the following objects and types by default:

Python	**JSON**
dict	object
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false

None	null

Simple JSON decoder.
Performs the following translations in decoding by default:

JSON	Python
object	dict
array	list
string	str
number (int)	int
number (real)	float
true	True
false	False
null	None

in json file data is represented as key and value
json module provides the following function for encoding and decoding.

1. dump(object,file)
2. load(file)

Example of creating json file

```
import json
def main():
    f=open("users.json","w")
    user={'name':'naresh',
          'uname':'nit','password':'nit123'}
    # json.dump(user,f)
    usersDict={'name':['naresh','ramesh'],
               'uname':['nit','ram'],
               'password':['nit123','ram123']}
    json.dump(usersDict,f)
    f.close()
main()
```

Output:
Output is saved inside users.json file

Example of reading content of json file

```
import json
def main():
    f=open("users.json","r")
    d=json.load(f)
    print(d)
    for key,value in d.items():
        print(key,value)

main()
```

Output:

```
{'name': ['naresh', 'ramesh'], 'uname': ['nit', 'ram'], 'password': ['nit123',
'ram123']}
name ['naresh', 'ramesh']
uname ['nit', 'ram']
password ['nit123', 'ram123']
```

Example of reading data from json file

```
import json
def main():
   f=open("f:\\iris.json","r")
   data=json.load(f)
   print(type(data))
   for row in data:
       for key,value in row.items():
           print(key,value)
main()
```

Output:

```
<class 'list'>
sepalLength 5.1
sepalWidth 3.5
petalLength 1.4
petalWidth 0.2
species setosa
sepalLength 4.9
sepalWidth 3.0
```

pickle module

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

Pickle module provides the following method or functions

1. dump
2. load

Example of creating binary file

```
import pickle
def main():
    a=100 #integer object
    b=1.5 # float object
    c=1+2j # complex object
    f=open("datafile.ser","wb")
    pickle.dump(a,f)
    pickle.dump(b,f)
    pickle.dump(c,f)
    f.close()
main()
```

Output:
Output is saved inside datafile.ser

Dump function write the following data inside file

1. type information (data type)
2. data

Example of reading or unpicking

```
import pickle
def main():
    f=open("datafile.ser","rb")
```

```
    obj1=pickle.load(f)
    print(obj1,type(obj1))
    obj2=pickle.load(f)
    print(obj2,type(obj2))
    obj3=pickle.load(f)
    print(obj3,type(obj3))

main()
```

Output:

```
100 <class 'int'>
1.5 <class 'float'>
(1+2j) <class 'complex'>
```

Reading and writing user defined objects

```
import pickle
class Employee:
    empno=101
    ename="naresh"
    salary=5000


e1=Employee()
f=open("emp.ser","wb")
pickle.dump(e1,f)
f.close()
f=open("emp.ser","rb")
emp1=pickle.load(f)
print(emp1.empno,emp1.ename,emp1.salary)
```

Output:

```
101 naresh 5000
```