Content delivered in class_14_23-August-2016

- Chapter 9: Working with Files
  - reading the data from an existing file
  - Creating a new text file
  - Reading and writing data from/to files
  - Working with txt files
  - Working with csv files
- Chapter 10: Serialization
  - Using Pickle
  - Using Shelve
  - Using JSON

Interview Questions Discussed:

Interview Question 1: Difference between compression and serialization ?

Interview Question 2: When a function object is deleted, but the object created with that function call is retained, will it malfunction, or work properly?

Interview Question 3: How different is python dictionary, from that of json?

Assignments Given

Assignment 1 : Try to open the same file in two different modes, in parallel, and observe the problem occurring

Assignment 2: Write a script for camel casing to underscore casing, and vice versa; need to be done on an existing .py file. Ensure that functions imported from modules, are not disturbed.

Assignment 3: write a function to display all the cars, in this sampleCSVFile.csv

Assignment 4: Use json.loads to correspondingly decode it

Assignment 5: Go through this Yahoo weblink (https://developer.yahoo.com/python/python-rest.html) and have an insight on accessing web content using python.

Working with files

In [1]:
```
import os
os.chdir(r'C:\pyExercises') # changing the cwd
os.getcwd()
```

Out[1]:
```
'C:\\pyExercises'
```

File operation modes

r - read only

w - write only

a - appending the data

Note: If you open an existing file with 'w' mode, it's existing data get vanished.

r+ - both for read and write

a+ - both for read and append

In windows, the data is stored in binary format. Placing this 'b' doesn't effect in unix and linux.

rb - read only

wb - write only

ab - append only

ab+ - Both reading and appending data

Default file operation is **read only.**

Accessing a file

Taking a sample file, named test.txt

```
test.txt

Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
```

Now, run this demonstration script

```
#!/usr/bin/python
# fileOperations.py
'''
Purpose: File operations demonstration
'''

# accessing an exising file
f = open('test.txt', 'rb')
# 'f' is the file handler
print f, type(f)


# reading the data present in the test.txt
data = f.read()
print "data  = ", data

print 'Again trying to print the data from file'
data = f.read()
print "data  = ", data

print 'The current position of cursor in file is ', f.tell()

print 'moving the cursor position to 0'
f.seek(0)
print 'The current position of cursor in file is ', f.tell()

print 'The first 12 characters in the file are ', f.read(12)

print 'The next 6 characters in the file are \n', f.read(6)

print 'The current line in file is \n', f.readline()

print 'The current line in file is \n', f.readline()

print 'checking whether file is closed or not'
print 'return True, if the file is closed', f.closed

f.close()

print 'checking whether file is closed or not'
print 'return True, if the file is closed', f.closed

try:
    f.read() #No operation can be performed on a closed file object, as the ob
ject gets dereferenced
    # garbage collector deletes all the unreferenced objects
except ValueError, ve:
    print ve
    print "IO operation can't be performed on closed file handler"


g = open('test.txt', 'wb')   # opening existing file in read mode will erase i
ts existing data.

try:
    datag = g.read()
    print "datag = ", datag
```

```
except IOError, ex:
    print ex
    print 'opened file in write mode. can not read the data'

g.write('Python programming is interesting\n')
g.write('It is coming with batteries, in built\n')
g.write('It means that almost every operation has a module !')

g.close()  # it is not mandatory , but extremely recommended.
# python interpreter with close the file, but
# IronPython, Jython, ... may not close the file automatically.

# Using Context manager for file handling

with open('test.txt', 'ab+') as f:
    print 'The cursor position is at %d'%(f.tell())
    dataf = f.read()
    print 'The cursor position is at %d' % (f.tell())
    print 'The content of the data is \n', dataf
    # append mode supports both read and write operations
    print 'The cursor position is at %d' % (f.tell())
    print f.read(123)
    f.write('This is last line')
    print 'The cursor position is at %d' % (f.tell())
    f.close()
```

```
<open file 'test.txt', mode 'rb' at 0x04809548> <type 'file'>
data  =  Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
Again trying to print the data from file
data  =
The current position of cursor in file is  123
moving the cursor position to 0
The current position of cursor in file is  0
The first 12 characters in the file are  Python progr
The next 6 characters in the file are
amming
The current line in file is
 is interesting

The current line in file is
It is coming with batteries, in built

checking whether file is closed or not
return True, if the file is closed False
checking whether file is closed or not
return True, if the file is closed True
I/O operation on closed file
IO operation can't be performed on closed file handler
File not open for reading
opened file in write mode. can not read the data
The cursor position is at 0
The cursor position is at 123
The content of the data is
Python programming is interesting
It is coming with batteries, in built
It means that almost every operation has a module !
The cursor position is at 123

The cursor position is at 140
```

Assignment 1 : Try to open the same file in two different modes, in parallel, and observe the problem occurring

Assignment 2: Write a script for camel casing to underscore casing, and vice versa; need to be done on an existing .py file. Ensure that functions imported from modules, are not disturbed.

Working with CSV files

Taking a sample csv file

```
sampleCSVFile.csv
    fruits, vegetables, cars
    Apple, Cabbagge, Benz
    Mango, Cucumber, Volvo
    Banana, Raddish, Maruthi suzuki
```

Creating a csv file

In [19]:
```
with open('sampleCSVFile.csv', 'ab+') as myCsv:
    myCsv.write("fruits, vegetables, cars\n")
    myCsv.write("Apple, Cabbagge, Benz\n")
    myCsv.write("Mango, Cucumber, Volvo\n")
    myCsv.write("Banana, Raddish, Maruthi suzuki\n")
    myCsv.close()
```

In [20]:
```
import os;
print os.listdir(os.getcwd())
```

```
['.ipynb_checkpoints', 'decoratorExample.py', 'fileOperations.py', 'sampleCSV
File.csv', 'test.txt', 'Type conversions in Python.ipynb']
```

In [22]:
```
#!/usr/bin/python
# workingWithCSV.py
import csv
'''
    Purpose : Working with CSV files

'''
# with is called as a context manager
with open('sampleCSVFile.csv') as csvFile:    # the default opening mode is re
ad mode
    data = csv.reader(csvFile, delimiter = ',')
    print data  # it is an iterator object
    print type(data)
    print dir(data)
    print data.next()
    for ele in data:
        print ele
        #print ele[0]

    csvFile.close()
```

```
<_csv.reader object at 0x047B5EB0>
<type '_csv.reader'>
['__class__', '__delattr__', '__doc__', '__format__', '__getattribute__', '__
hash__', '__init__', '__iter__', '__new__', '__reduce__', '__reduce_ex__', '_
_repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', 'dialec
t', 'line_num', 'next']
['fruits', ' vegetables', ' cars']
['Apple', ' Cabbagge', ' Benz']
['Mango', ' Cucumber', ' Volvo']
['Banana', ' Raddish', ' Maruthi suzuki']
```

Assignment 3: write a function to display all the cars, in this sampleCSVFile.csv

Data Serialization

- converting the objects into byte form .
- Used for transferring the objects.
- Serialization is the process of converting a data structure or object state into a format that can be stored.
- Serialization is also called deflating or marshalling
- DeSerialization is also called Inflating or unmarshalling.

Various ways of data serialization

- Marshall -- It is primitive, and no more used.
- Pickle -- Pickle is a standard module which serializes and deserializes a python object structure.
- cPickle -- c implementation of pickle

Python 2.x has both pickle and cpickle. Whereas python 3.x has only cPickle, and it is renamed as pickle.

Pickle files has .pkl or .pickle extensions. The pickled data format is python specific.

Interview Question 1: Difference between compression and serialization ?

Ans: compression may be lossy or lossless process. Whereas Serialization is a reversible process compression is used to compress the data; whereas serialization is used for inflating or deflating an object.

In [23]:
```
#!/usr/bin/python
# workingWithPickle.py

import pickle

'''
    Purpose: Working with Pickle files
    pickling

'''

# Serialization
students = ['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 123]

f = open('ktgroupStudents.pickle', 'ab+')
pickle.dump(students, f)
f.flush()

f.close()

# Deserialization
g = open('ktgroupStudents.pickle', 'rb')
myStudents = pickle.load(g)
print "myStudents are ", myStudents
g.close()

mystud = pickle.dumps(students)
print mystud

orgStud = pickle.loads(mystud)
print orgStud
```

```
myStudents are  ['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 123]
(lp0
S'Mujeeb'
p1
aS'Harini'
p2
aS'Mamatha'
p3
aS'Ankit'
p4
aI123
a.
['Mujeeb', 'Harini', 'Mamatha', 'Ankit', 123]
```

In **conclusion**, Pickle and cpickle has their importance in interfacing with c and C++**.** As pickled data format is python specific, it is not used in interfacing with other languages**.**

In [24]:
```
def func():
    pass
```

In [25]:
```
z = func()

print z, type(z)
```

```
None <type 'NoneType'>
```

In [27]:
```
try:
    zz = pickle.dumps(z)
except pickle.PickleError as pe:
    print 'The error is ', pe
```

In [28]:
```
zz
```

Out[28]:
```
'N.'
```

In [29]:
```
del func
```

In [30]:
```
try:
    zz = pickle.dumps(z)
except pickle.PickleError as pe:
    print 'The error is ', pe
```

Interview Question 2: When a function object is deleted, but the object created with that function call is retained, will it malfunction, or work properly?

In [31]:
```
class class1():
    pass
```

In [32]:
```
b = class1()
```

In [33]:
```
pickle.dumps(b)
```

Out[33]:
```
'(i__main__\nclass1\np0\n(dp1\nb.'
```

In [34]:
```
del class1
```

In [35]:
```
print b
```

```
<__main__.class1 instance at 0x03804620>
```

Though class is deleted, its instance survivies; but, that isn't the case with functions

In [36]:
```
pickle.dumps('I am pickling')
```

Out[36]:
```
"S'I am pickling'\np0\n."
```

In [37]:
```
pickle.loads("S'I am pickling'\np0\n.")
```

Out[37]:
```
'I am pickling'
```

In [38]:
```
pickle.loads(pickle.dumps('I am pickling'))
```

Out[38]:
```
'I am pickling'
```

Limitations of Pickle

- The pickle module is not secure against erroneous or maliciously constructed data.
- The pickled data can be modfied on-the-fly, mainly by Middle-man attack.
- Never unpickle data received from an untrusted or unauthenticated source.

Shelve

- shelve is a tool that uses pickle to store python objects in an access-by-key file system. It is same as keys in dictionary.
- It is used as a simple persistent storage option for python objects when a relational database is overkill.
- The values are pickled and written to a database created and managed by anydbm.
- anydbm is a frontend interface to establish communication with database.

In [39]:
```
import shelve
```

In [40]:
```
s= shelve.open('usingShelve.db')
```

In [41]:
```
try:
    s['key1'] = {'int': 8, 'float': 8.0, 'string': '8'}
except Exception, ex:
    print ex
finally:
    s.close()
```

In [42]:
```
# To access the data again,

s = shelve.open('usingShelve.db')  # default is read only mode
try:
    myShelveContent = s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()

print 'myShelveContent = ', myShelveContent
```

```
myShelveContent =  {'int': 8, 'float': 8.0, 'string': '8'}
```

opening shelve in read-only mode.

In [43]:
```
s = shelve.open('usingShelve.db', flag = 'r')
try:
    myShelveContent = s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()

print 'myShelveContent = ', myShelveContent
```

```
myShelveContent =  {'int': 8, 'float': 8.0, 'string': '8'}
```

In [44]:
```
s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
    s['key1']['newValue'] = 'This is a new value'
    print s['key1']
except Exception, ex1:
    print ex1
finally:
    s.close()

s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()
```

```
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8'}
```

By defaults, shelve do not track modifications to volatile objects. If necessary, open the shelve file in writeback enabled

In [45]:
```
s = shelve.open('usingShelve.db', writeback=True)
try:
    print s['key1']   # accessing using the key
    s['key1']['newValue'] = 'This is a new value'
    print s['key1']
except Exception, ex1:
    print ex1
finally:
    s.close()

s = shelve.open('usingShelve.db', flag = 'r')
try:
    print s['key1']   # accessing using the key
except Exception, ex1:
    print ex1
finally:
    s.close()
```

```
{'int': 8, 'float': 8.0, 'string': '8'}
{'int': 8, 'float': 8.0, 'string': '8', 'newValue': 'This is a new value'}
{'int': 8, 'float': 8.0, 'string': '8', 'newValue': 'This is a new value'}
```

Note: Shelve must not be opened with writeback enabled, unless it is essential

Json

- JSON is an abbreviation for Java Script Object notation
- json is supported by almost all languages.
- official website is json.org
- stored in .json file. But, it can be stored in other format too, like .template file in AWS cloudformation, etc.

sample json

```
{
    "employees": {
        "employee": [
            {
                "id": "1",
                "firstName": "Tom",
                "lastName": "Cruise",
                "photo": "http://cdn2.gossipcenter.com/sites/default/files/imagecac
he/story_header/photos/tom-cruise-020514sp.jpg"
            },
            {
                "id": "2",
                "firstName": "Maria",
                "lastName": "Sharapova",
                "photo": "http://thewallmachine.com/files/1363603040.jpg"
            },
            {
                "id": "3",
                "firstName": "James",
                "lastName": "Bond",
                "photo": "http://georgesjournal.files.wordpress.com/2012/02/007_at_
50_ge_pierece_brosnan.jpg"
            }
        ]
    }
}
```

In [47]:
```
import json
```

In [48]:
```
students = {'batch 1' : {'Name': 'Ankit', 'regNumber': 1},
            'batch 2' : {'Name': 'Mujeeb', 'regNumber': 2}
           }
```

In [49]:
```
print json.dumps(students)  # dictionary to json
```

```
{"batch 2": {"regNumber": 2, "Name": "Mujeeb"}, "batch 1": {"regNumber": 1,
 "Name": "Ankit"}}
```

In [51]:
```
print json.dumps(students, sort_keys = True)
```

```
{"batch 1": {"Name": "Ankit", "regNumber": 1}, "batch 2": {"Name": "Mujeeb",
 "regNumber": 2}}
```

tuple to json array

In [54]:
```
tup1 = ('Red', 'Black', 'white', True, None)
json.dumps(tup1)
```

Out[54]:
```
'["Red", "Black", "white", true, null]'
```

Observe that True is changed to true; and None to null

In [55]:
```
string = 'This is a string'
```

In [56]:
```
json.dumps(string)
```

Out[56]:
```
'"This is a string"'
```

In [57]:
```
b = True
json.dumps(b)
```

Out[57]:
```
'true'
```

In [58]:
```
a = -123
b = -2.34
z = 1.3e-10
```

In [59]:
```
json.dumps(a)
```

Out[59]:
```
'-123'
```

In [60]:
```
json.dumps(b)
```

Out[60]:
```
'-2.34'
```

In [61]:
```
json.dumps(z)
```

Out[61]:
```
'1.3e-10'
```

Interview Question 3: How different is python dictionary, from that of json?

json is different from dictionary.

```
- All the content within the dictionary must be in key-value pairs; whereas there c
an be arrays in dictionary.
- And, json data types are different from that of python data types.
```

json doesn't cover all the datatypes of python, mainly tuples and bytes.

Decoding the json data, back to python data types can be achieved using json.loads correspondingly.

Assignment 4: Use json.loads to correspondingly decode it

json to python object conversion pairs:

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

There are various online json lints to validate a written json file; and online json to other format convertors, like code Beautify (http://codebeautify.org/jsonviewer/), json formatter (https://jsonformatter.curiousconcept.com/), etc

Assignment 5: Go through this Yahoo weblink (https://developer.yahoo.com/python/python-rest.html) and have an insight on accessing web content using python.

In [62]:
```
import urllib

url = 'https://developer.yahoo.com/'
u = urllib.urlopen(url)
# u is a file-like object
data = u.read()
f = open('yahooData.txt', 'ab+')

f.write(data)
f.close()
```

Both urllib and urllib serve the same purpose; But, urllib2 module can handle HTTP Errors better than urllib

In [63]:
```
import urllib2

try:
        data = urllib2.urlopen(url).read()
except urllib2.HTTPError, e:
        print "HTTP error: %d" % e.code
except urllib2.URLError, e:
        print "Network error: %s" % e.reason.args[1]

f = open('yahooData1.txt', 'ab+')

f.write(data)
f.close()
```