Object Serialization

The process of converting an object from python supported form to either file supported form or network supported form, is called serialization (Marshalling or pickling)

The process of converting an object from either file supported form or network supported form to python supported form is called deserialization (Unmarshalling or unpickling).

Object Serialization by using Pickle
Object Serialization by using JSON
Object Serialization by using YAML

Object Serialization by using Pickle:

We can perform serialization and deserialization of an object wrt file by using pickle module. It is Python's inbuilt module.

pickle module contains dump() function to perform Serialization(pickling).

```
pickle.dump(object,file)
```

pickle module contains load() function to perform Deserialization (unpickling).

```
object = pickle.load(file)
```

Program to perform pickling and unpickling of Employee Object:

```
1)  import pickle
2)  class Employee:
3)      def __init__(self,eno,ename,esal,eaddr):
4)          self.eno=eno
5)          self.ename=ename
6)          self.esal=esal
7)          self.eaddr=eaddr
8)      def display(self):
9)          print('ENO:{}, ENAME:{}, ESAL:{}, EADDR:{}'.format(self.eno,self.ename,self.es
    al,self.eaddr))
10)
11) e=Employee(100,'Durga',1000,'Hyderabad')
12) with open('emp.dat','wb') as f:
13)     pickle.dump(e,f)
14)     print('Pickling of Employee object completed')
15)
```


```
16) with open('emp.dat','rb') as f:
17)     obj = pickle.load(f)
18)     print('Unpickling of Employee object complected')
19)     print('Printing Employee Information:')
20)     obj.display()
```

Program for Serializing Multiple Employee objects to the file:

Based on our requirement, we can serialize any number of objects to the file.

emp.py

```
1)  class Employee:
2)      def __init__(self,eno,ename,esal,eaddr):
3)          self.eno=eno
4)          self.ename=ename
5)          self.esal=esal
6)          self.eaddr=eaddr
7)      def display(self):
8)          print('ENO:{}, ENAME:{}, ESAL:{}, EADDR:{}'.format(self.eno,self.ename,self.es
    al,self.eaddr))
```

sender.py

```
1)  #Sender is responsible to save Employee objects to the file
2)  from emp import *
3)  import pickle
4)  f=open('emp.dat','wb')
5)  while True:
6)      eno=int(input('Enter Employee Number:'))
7)      ename=input('Enter Employee Name:')
8)      esal=float(input('Enter Employee Salary:'))
9)      eaddr=input('Enter Employee Address:')
10)     e=Employee(eno,ename,esal,eaddr)
11)     pickle.dump(e,f)
12)     option=input('Do you want to serialize one more Employee object[Yes|No]:')
13)     if option.lower()=='no':
14)         break
15) print('All Employee objects serialized')
```

receiver.py

```
1)  #Receiver is responsible to deserialize Employee objects
2)  import pickle
3)  f=open('emp.dat','rb')
4)  print('Deserializing Employee objects and printing data...')
5)  while True:
6)      try:
7)          obj=pickle.load(f)
8)          obj.display()
9)      except EOFError:
10)         print('All Employees Completed')
11)         break
```

Object Serialization by using JSON

Importance of JSON:

JSON ➔ JavaScript Object Notation

Any programming language can understand json. Hence JSON is the most commonly used message format for applications irrespective of programming language and platform. It is very helpful for interoperability between applications.
It is human readable format.
It is light weight and required less memory to store data.

Eg:
Java Application sends request to Python application
Python application provide required response in json form.
Java application can understand json form and can be used based on its requirement.

What is JSON?

Python Data Types vs JavaScript Data Types

int → number
float → number
list → array
dict → object(JSON)
str → string
True → true
False → false

None → null
etc

In javascript if we want to represent a group of key value pairs, then we should go for object data type, which is nothing but json.
JSON is very similar to Python's dict object.

Why preference for JSON over XML:

1) Light weight
2) Human readable

Python's json module:

As the part of programming, it is very common requirement to convert python object into json form and from json form to python object. For these conversions (Serialization and Deserialization) Python provides inbuilt module json.

json module defines the following important functions:

For Serialization Purpose (From Python to JSON Form):

1) dumps() → It serializes python dict object to json string.
2) dump() → Converting python dict object to json and write that json to provided json file. It serializes to a file.

For Deserialization Purpose (From JSON form to Python form):

1) loads() → Converting JSON string to python dict. It deserializes to a string.
2) load() → Reading json from a file and converting to python dict object. Deserializes from a json file.

Demo program for Serialization

```
1)  import json
2)  employee={'name':'durga',
3)            'age':35,
4)           'salary':1000.0,
5)            'ismarried':True,
6)            'ishavinggirlfriend':None
7)           }
8)  json_string = json.dumps(employee,indent=4,sort_keys=True)
9)  print(json_string)
10)
11) with open('emp.json','w') as f:
```


```
12)    json.dump(employee,f,indent=4)
```

Demo Program for Deserialization from json String

```
1)  import json
2)  json_string='''{
3)     "name": "durga",
4)     "age": 35,
5)     "salary": 1000.0,
6)     "ismarried": true,
7)     "ishavinggirlfriend": null
8)  }'''
9)  emp_dict=json.loads(json_string)
10) print(type(emp_dict))
11) print('Employee Name:',emp_dict['name'])
12) print('Employee Age:',emp_dict['age'])
13) print('Employee Salary:',emp_dict['salary'])
14) print('Is Employee Married:',emp_dict['ismarried'])
15) print('Is Employee Has GF:',emp_dict['ishavinggirlfriend'])
16)
17) for k,v in emp_dict.items():
18)     print('{} : {}'.format(k,v))
```

Demo Program for Deserialization from json file

```
1)  import json
2)  with open('emp.json','r') as f:
3)     emp_dict=json.load(f)
4)
5)  print(type(emp_dict))
6)  print('Employee Name:',emp_dict['name'])
7)  print('Employee Age:',emp_dict['age'])
8)  print('Employee Salary:',emp_dict['salary'])
9)  print('Is Employee Married:',emp_dict['ismarried'])
10) print('Is Employee Has GF:',emp_dict['ishavinggirlfriend'])
11)
12) for k,v in emp_dict.items():
13)     print('{} : {}'.format(k,v))
```

FAQs from json module:

Q1. What is the difference between dump() and load() functions of json module?
Q2. What is the difference between dump() and dumps() functions of json module?
Q3. What is the difference between load() and loads() functions of json module?

Communicate with coindesk application to get bitcoin price:

If we send http request to coindesk application it will provide bitcoin current price information.

We can send http request from python application by using requests module. We have to install this module separately.

pip install requests

We can send request to coindesk application by using the following url:

https://api.coindesk.com/v1/bpi/currentprice.json

It will provide the following response in json format.

{
"time":{"updated":"Sep 18, 2013 17:27:00 UTC","updatedISO":"2013-09-18T17:27:00+00:00"},

"disclaimer":"This data was produced from the CoinDesk Bitcoin Price Index. Non-USD currency data converted using hourly conversion rate from openexchangerates.org",

"bpi":{"USD":{"code":"USD","symbol":"$","rate":"126.5235","description":"United States Dollar","rate_float":126.5235},

"GBP":{"code":"GBP","symbol":"£","rate":"79.2495","description":"British Pound Sterling","rate_float":79.2495},

"EUR":{"code":"EUR","symbol":"€","rate":"94.7398","description":"Euro","rate_float":94.7398}}}

Demo Program

```
1) import requests
2) response=requests.get('https://api.coindesk.com/v1/bpi/currentprice.json')
3) binfo=response.json() # provides python's dict object
4) #print(type(binfo))
```

```
#print(binfo)
print('Bitcoin Price as on',binfo['time']['updated'])
print('1 BitCoin = $',binfo['bpi']['USD']['rate'])
```

How to perform serialization and deserialization of customized class objects:

json.dumps() → python dict to json string
json.dump() → python dict to json file
dump(),dumps() functions will work only for python dict objects, and we cannot use for our customized class objects like Employee, Customer etc.

json.loads() → json string to python dict
json.load() → json file to python dict

load() and loads() functions will always provide python dict objects as return type and we won't get our customized class objects directly.

The required conversions we have to take care explicitly.

Demo Program

```
import json
class Employee:
    def __init__(self,eno,ename,esal,eaddr):
        self.eno=eno
        self.ename=ename
        self.esal=esal
        self.eaddr=eaddr
    def display(self):
        print('ENO:{}, ENAME:{}, ESAL:{}, EADDR:{}'.format(self.eno,self.ename,self.esal,self.eaddr))

e=Employee(100,'Durga',1000.0,'Hyderabad')

#emp_dict={'eno':e.eno,'ename':e.ename,'esal':e.esal,'eaddr':e.eaddr}

emp_dict=e.__dict__

with open('emp.json','w') as f:
    json.dump(emp_dict,f,indent=4)

with open('emp.json','r') as f:
```

```
21)    edict = json.load(f)
22) #print(type(edict))
23)
24) newE=Employee(edict['eno'],edict['ename'],edict['esal'],edict['eaddr'])
25)
26) newE.display()
```

Note: In the above program, we converted Employee object to dict object explicitly to perform serialization, because dump() function will always accept dict type only.

load() function always returns dict type only and hence we have to convert that dict object to Employee object explicitly.

Short-cut:

```
e=Employee(100,'Durga',1000.0,'Hyderabad')

with open('emp.json','w') as f:
        json.dump(e.__dict__,f,indent=4)
```

jsonpickle module:

By using jsonpickle module we can serialize our custom class objects directly to json and we can deserialize json to our custom class objects directly.

jsonpickle module is not available bydefault and we have to install explicitly.

```
pip install jsonpickle
```

This module contains

1. encode() → To convert any object to json_string directly
2. decode() → To convert json_string to our original object

Demo Program for serialization and deserialization by using jsonpickle

```
1)  import jsonpickle
2)  class Employee:
3)      def __init__(self,eno,ename,esal,eaddr,isMarried):
4)          self.eno=eno
5)          self.ename=ename
6)          self.esal=esal
7)          self.eaddr=eaddr
8)          self.isMarried=isMarried
```


```
9)     def display(self):
10)        print('ENO:{}, ENAME:{}, ESAL:{}, EADDR:{}, Is Married:{}'.format(self.eno,self.
       ename,self.esal,self.eaddr,self.isMarried))
11)
12) #Serialization to String
13) e=Employee(100,'Durga',1000.0,'Hyderabad',True)
14) json_string = jsonpickle.encode(e)
15) print(json_string)
16)
17) #Serialization to file
18) with open('emp.json','w') as f:
19)     f.write(json_string)
20)
21) #Deserialization
22) newEmp=jsonpickle.decode(json_string)
23) #print(type(newEmp))
24) newEmp.display()
25)
26) #Deserialization From the file
27) with open('emp.json','r') as f:
28)     json_string=f.readline()
29) newEmp=jsonpickle.decode(json_string)
30) newEmp.display()
```

Object Serialization with YAML

YAML: A retronym for YAML Ain't Markup Language that meant originally Yet Another Markup Language.

It is alternative to JSON.
It is also light weight and human understandable form.
It is more readable than JSON.

To serialize and deserialize our python data to yaml, we have to go for pyaml libray. This library by default not available and we have to install separately.

pip install pyaml

pyaml library contains yaml module.

yaml module contains the following functions to perform serialization and deserialization.

For Serialization

1.dump() → To serialize python dict object to yaml string or yaml file.

For Deserialization

2. load() → To deserialize from yaml string or yaml file to python dict object

Note: load() is deprecated and we have to use safe_load() function.

Demo Program for serialization and deserialization by using yaml

```
1)  from pyaml import yaml
2)  emp_dict={'name':'Durga','age':35,'salary':1000.0,'isMarried':True}
3)
4)  #Serialization to yaml string
5)  yaml_string=yaml.dump(emp_dict)
6)  print(yaml_string)
7)
8)  #Serialization to yaml file
9)  with open('emp.yaml','w') as f:
10)     yaml.dump(emp_dict,f)
11)
12) #Deserialization from yaml string
13) ed=yaml.safe_load(yaml_string)
14) print(ed)
15) for k,v in ed.items():
16)     print(k,':',v)
17)
18) #Deserialization from yaml file
19) with open('emp.yaml','r') as f:
20)     edf=yaml.safe_load(f)
21) print(edf)
```