Python Programming

by Narendra Allam

Chapter 9

Serialization

Topics Covering

- Pickle Module
- pickling built-in data structures
    - byte strings
    - binary
- xml construction and parsing
- json construction and parsing

Serialization: Serialization is the process of transforming data from one container to to another. An employee info is stored in a databse table as a record, the same is stored in a program as a tuple, structure variable or a class object. Same data can be transformed into some text representation like an xml file or json file. Data changes its container but not the structure. This process of transformation happens at various stages. The process of transforming from format A to format B is called **Serialization** and B to A is called **Deserialization**. Different technologies have different names for this process.

Encoding - Decoding
Marshalling - Unmarshalling
Pickling - Unpickling

all refer to the same process. But at present, most of the technical contexts serialization is being referred when conversion happens to and from XML and JSON formats.

In this chapter we mainly discuss 3 formats.

1. Pickling - python exclusive format
2. xml
3. json

Pickle

The pickle module implements a fundamental, but powerful algorithm 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 is converted back into an object hierarchy.

converting a python data structure into pickle format

dumps() function converts python data structure into a binary string format, later we can transfer this to a file or network.

```
In [1]:
```

```
import pickle
d = {}
d['id '] = 12345
d['name'] = 'Obama'
d['salary'] = 9000000.0
d['full_time'] = True
print(d)
```

```
{'id ': 12345, 'name': 'Obama', 'salary': 9000000.0, 'full_time': Tru
e}
```

```
In [2]:
```

```
bs = pickle.dumps(d)
```

```
In [3]:
```

```
print(bs)
```

```
b'\x80\x03}q\x00(X\x03\x00\x00\x00id q\x01M90X\x04\x00\x00\x00nameq\x0
2X\x05\x00\x00\x00Obamaq\x03X\x06\x00\x00\x00salaryq\x04GAa*\x88\x00\x
00\x00\x00X\t\x00\x00\x00full_timeq\x05\x88u.'
```

Writing into a file in binary format('wb' mode)

```
In [4]:
```

```
f = open('employ.pickle', 'wb')
f.write(bs)
f.close()
```

Checking the file content

```
In [5]:
```

```
!cat employ.pickle # !type employ.pickle
```

```
�}q(Xid qM90XnameqXObamaqXsalaryqGAa*�X        full_timeq�u.
```

Reading content from a pickle file

loads() funcion is used to load data back from a pickle file

```
In [6]:
```

```
f = open('employ.pickle', 'rb')
bs = f.read()
d1 = pickle.loads(bs)
print(d1)
f.close()
```

```
{'id ': 12345, 'name': 'Obama', 'salary': 9000000.0, 'full_time': Tru
e}
```

Creating an intermediatory binary string is not rquired all the time. We may want to directly write into a file. Python provides a straight way to do this. load() and dump() functions.

In [7]:

```
import pickle

d = {}
d['id '] = 12345
d['name'] = 'Obama'
d['salary'] = 9000000.0
d['full_time'] = True

f = open('employee.pickle', 'wb')
pickle.dump(d, f, pickle.HIGHEST_PROTOCOL)
f.close()
```

pickle.HIGHEST_PROTOCOL ensures the highest protocol to be used in pickling protocol. We can see the pure binary form of pickling.

In [8]:

```
!cat employee.pickle
```

```
��<}�(�id �M90�name��Obama��salary�GAa*��       full_time��u.
```

Using load to unpickle back to original data structure:

In [9]:

```
d = {}
with open('employee.pickle', 'rb') as f:
    d = pickle.load(f)
print (d)
```

```
{'id ': 12345, 'name': 'Obama', 'salary': 9000000.0, 'full_time': Tru
e}
```

Xml

Xml is another popular format used mainly in web based data exchange scenorios. XML is an hierarchical data format, and the most natural way to represent it is with a tree.

1. `xml.etree.ElementTree`
2. `ET.Element()`

ElementTree represents the whole XML document as a tree, and **Element** represents a single node in this tree. Interactions with the whole document (reading and writing to/from files) are usually done on the ElementTree level. Interactions with a single XML element and its sub-elements are done on the Element level. **Element** has four sections in it. xml.etree.ElementTree is the module that is required to import.

```
import xml.etree.ElementTree as ET
```

A simple xml tage looks like below.

```
<TagName Attr1='val' Attr2='val2' ... > TextData </TagName> TailData
```

TagName is refered as **'tag'**
Attr1 and *Attr2* are attributes refered as **'attrib'** which is a dictionary
TextData is referred as **'text'**
TailData is referred as **'tail'**

Sample xml file looks like below

```
<book>
    <title type='short'> Programming C </title>
    <author> Yeswanth Kanetkar </author>
    <author> Venu Gopal </author>
    <pages size='A5' color='white'> 230 </pages> 5
    <id>
        <isbn> 1234455 </isbn>
        <isbn13> 11313133131 </isbn13>
    </id>
</book>
```

ElementTree.Element is basic type from xml.etree.ElementTree which is required to construct a baisc XML tag.

In [10]:

```
import xml.etree.ElementTree as ET
root = ET.Element('book')

title = ET.Element('title')
title.attrib = {'type': 'short'}
title.text = 'Programming C'


author1 = ET.Element('author')
author1.text = 'Yaswanth Kanetkar'

author2 = ET.Element('author')
author2.text = 'Venu Gopal'

pages = ET.Element('pages')
pages.attrib = {'size': 'A5', 'color': 'white'}
pages.text = '230'
pages.tail = '5'

_id = ET.Element('id')

isbn = ET.Element('isbn')
isbn.text = '1234455'

isbn13 = ET.Element('isbn13')
isbn13.text = '11313133131'

_id.append(isbn)
_id.append(isbn13)

root.append(title)
root.append(author1)
root.append(author2)
root.append(pages)
root.append(_id)
```

All sub tags in root are stored as a list, we can use list indexing to access them.

In [11]:

```
root[0].text
```

Out[11]:

```
'Programming C'
```

In [12]:

```
root[4][1].text
```

Out[12]:

```
'11313133131'
```

Iterating through all tags

In [13]:

```
for item in root:
    print (item.tag, item.attrib, item.text, item.tail)
```

```
title {'type': 'short'} Programming C None
author {} Yaswanth Kanetkar None
author {} Venu Gopal None
pages {'size': 'A5', 'color': 'white'} 230 5
id {} None None
```

Converting xml tree to string

In [14]:

```
s = ET.tostring(root, encoding='unicode', method='xml')
print (s)
```

```
<book><title type="short">Programming C</title><author>Yaswanth Kanetk
ar</author><author>Venu Gopal</author><pages color="white" size="A5">2
30</pages>5<id><isbn>1234455</isbn><isbn13>11313133131</isbn13></id></
book>
```

In [15]:

```
with open('book.xml', 'w') as f:
    f.write(s)
```

In [16]:

```
!cat book.xml
```

```
<book><title type="short">Programming C</title><author>Yaswanth Kanetk
ar</author><author>Venu Gopal</author><pages color="white" size="A5">2
30</pages>5<id><isbn>1234455</isbn><isbn13>11313133131</isbn13></id></
book>
```

Excercise : construct employee.xml tree

```
<?xml version='1.0' encoding='utf8'?>
<employee>
    <id>1234</id>
    <name>John</name>
    <sal>200000</sal>
    <address>
        <street>High school street</street>
        <pin>500007</pin>
    </address>
</employee>
```

Reading and parsing external xml files:

The below is the externl file, country.xml. Let's see how we are going to parse this using xml module

country.xml

```
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein" latittude ="48">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

ET.parse() function:

In [17]:

```
import xml.etree.ElementTree as ET

tree = ET.parse('country.xml')
root = tree.getroot()

for country in root:
    print (country.tag, country.attrib, country.text, country.tail)
    for subtag in country:
        print ('    ', subtag.tag, subtag.attrib, subtag.text)
```

```
country {'name': 'Liechtenstein', 'latittude': '48'} 


     rank {} 1
     year {} 2008
     gdppc {} 141100
     neighbor {'name': 'Austria', 'direction': 'E'} None
     neighbor {'name': 'Switzerland', 'direction': 'W'} None
country {'name': 'Singapore'} 


     rank {} 4
     year {} 2011
     gdppc {} 59900
     neighbor {'name': 'Malaysia', 'direction': 'N'} None
country {'name': 'Panama'} 


     rank {} 68
     year {} 2011
     gdppc {} 13600
     neighbor {'name': 'Costa Rica', 'direction': 'W'} None
     neighbor {'name': 'Colombia', 'direction': 'E'} None
```

ET.fromstring(): Reading driectly from a string

In [18]:

```
country_data_as_string = '''
<data>
    <country name="Liechtenstein" latittude ="48">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
'''

root = ET.fromstring(country_data_as_string)
```

Finding a tag:

In [19]:

```
cs = root.find('country') # Finds first occurance
print (cs.tag, cs.attrib, cs.text)
```

```
country {'name': 'Liechtenstein', 'latittude': '48'}
```

Finding all tags:

This finds tags which are direct children of the current element

In [20]:

```
for country in root.findall('country'):
    rank = country.find('rank').text
    name = country.get('name')
    print (name, rank)
```

```
Liechtenstein 1
Singapore 4
Panama 68
```

Finding Interested tags:

Iterates recursively over all the sub-tree below it

In [21]:

```
for neighbor in root.iter('neighbor'):
    print (neighbor.attrib)
```

```
{'name': 'Austria', 'direction': 'E'}
{'name': 'Switzerland', 'direction': 'W'}
{'name': 'Malaysia', 'direction': 'N'}
{'name': 'Costa Rica', 'direction': 'W'}
{'name': 'Colombia', 'direction': 'E'}
```

Updating a all tags:

Let's say, we want to add one to each country's rank, and add an updated attribute to the rank element:

In [22]:

```
for rank in root.iter('rank'):
    new_rank = int(rank.text) + 1
    rank.text = str(new_rank)
    rank.set('updated', 'yes')
```

Updating single tag:

Updating singapore tag

In [23]:

```
for country in root.iter('country'):
    if country.get('name') == 'Singapore':
        country.set('updated', 'Yes')
        country.find('rank').text = '5'
        break

print (ET.tostring(root, encoding='unicode', method='xml'))
```

```
<data>
    <country latittude="48" name="Liechtenstein">
        <rank updated="yes">2</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor direction="E" name="Austria" />
        <neighbor direction="W" name="Switzerland" />
    </country>
    <country name="Singapore" updated="Yes">
        <rank updated="yes">5</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor direction="N" name="Malaysia" />
    </country>
    <country name="Panama">
        <rank updated="yes">69</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor direction="W" name="Costa Rica" />
        <neighbor direction="E" name="Colombia" />
    </country>
</data>
```

Removing a tag:

In [24]:

```
for country in root.iter('country'):
    if country.get('name') == 'Singapore':
        root.remove(country)
        break

print (ET.tostring(root, encoding='utf8', method='xml'))
```

```
b'<?xml version=\'1.0\' encoding=\'utf8\'?>\n<data>\n    <country lati
ttude="48" name="Liechtenstein">\n        <rank updated="yes">2</rank>
\n        <year>2008</year>\n        <gdppc>141100</gdppc>\n        <n
eighbor direction="E" name="Austria" />\n        <neighbor direction
="W" name="Switzerland" />\n    </country>\n    <country name="Panam
a">\n        <rank updated="yes">69</rank>\n        <year>2011</year>
\n        <gdppc>13600</gdppc>\n        <neighbor direction="W" name
="Costa Rica" />\n        <neighbor direction="E" name="Colombia" />\n
    </country>\n</data>'
```

JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is based on a subset of the JavaScript Programming Language. JSON is a text format that is completely language independent. Python makes it simple to work with JSON files. The module used for this purpose is the json module.

JSON can only store the following objects:

- character strings
- numbers
- booleans (True/False)
- None
- lists
- dictionaries with character string keys

Every object that’s not one of these must be converted.

The following table maps from the names of Java script types to their analogous types in Python:

JavaScript	Python
string	string
number	int/float
object	dict
array	list
boolean	bool
null	None

Serializing data to json file:

In [25]:

```
import json

book = {'title': 'Let Us C',
        'type': 'short',
        'Authors' : ['Yaswanth Kanetkar', 'sahani'],
        'Pages': 230,
        'price': 560.0,
        'published': True,
        'solution_booklet': None}

s = json.dumps(book)
with open('book.json', 'w') as f:
    f.write(s)
```

In [26]:

```
s
```

Out[26]:

```
'{"title": "Let Us C", "type": "short", "Authors": ["Yaswanth Kanetka
r", "sahani"], "Pages": 230, "price": 560.0, "published": true, "solut
ion_booklet": null}'
```

In [27]:

```
!cat book.json
```

```
{"title": "Let Us C", "type": "short", "Authors": ["Yaswanth Kanetka
r", "sahani"], "Pages": 230, "price": 560.0, "published": true, "solut
ion_booklet": null}
```

Deserializing data from json file:

In [28]:

```
import json
b = {}
with open('book.json', 'r') as f:
    s = f.read()
    b = json.loads(s)
print (b)
```

```
{'title': 'Let Us C', 'type': 'short', 'Authors': ['Yaswanth Kanetka
r', 'sahani'], 'Pages': 230, 'price': 560.0, 'published': True, 'solut
ion_booklet': None}
```

Directly dumping into json file, without intermediatery string format:

In [29]:

```
import json
book = [
        { 'title': 'Let Us C',
         'type': 'short',
         'Authors' : {'author1' : 'Yaswanth Kanetkar',
                      'author2' : 'sahani'},
         'publisher': ['bpb', 'wrox', 'pearson', 'appress'],
         'Pages': 230,
         'price': 560.0,
         'published': True,
         'solution_booklet': None},
        { 'title': 'Python Programming',
         'type': 'long',
         'Authors' : {'author1' : 'Narendra Allam'},
         'publisher': ['bpb', 'wrox', 'pearson', 'appress'],
         'Pages': 650,
         'price': 750.0,
         'published': False,
         'solution_booklet': None}
       ]

f = open('books.json', 'w')
json.dump(book, f)
f.close()
```

In [30]:

```
!cat books.json
```

```
[{"title": "Let Us C", "type": "short", "Authors": {"author1": "Yaswan
th Kanetkar", "author2": "sahani"}, "publisher": ["bpb", "wrox", "pear
son", "appress"], "Pages": 230, "price": 560.0, "published": true, "so
lution_booklet": null}, {"title": "Python Programming", "type": "lon
g", "Authors": {"author1": "Narendra Allam"}, "publisher": ["bpb", "wr
ox", "pearson", "appress"], "Pages": 650, "price": 750.0, "published":
false, "solution_booklet": null}]
```

Loading from json file

In [31]:

```
import json
f = open('books.json', 'r')
d = json.load(f)
print (d)
```

```
[{'title': 'Let Us C', 'type': 'short', 'Authors': {'author1': 'Yaswan
th Kanetkar', 'author2': 'sahani'}, 'publisher': ['bpb', 'wrox', 'pear
son', 'appress'], 'Pages': 230, 'price': 560.0, 'published': True, 'so
lution_booklet': None}, {'title': 'Python Programming', 'type': 'lon
g', 'Authors': {'author1': 'Narendra Allam'}, 'publisher': ['bpb', 'wr
ox', 'pearson', 'appress'], 'Pages': 650, 'price': 750.0, 'published':
False, 'solution_booklet': None}]
```