Introduction to Python
Persistence / Serialization

Christopher Barker

UW Continuing Education / Isilon

September 05, 2012

Table of Contents

Review of Previous Class

- doctests
- unittests
- profiling

Anyone add some doctests or unittests to his project?

Anyone time or profile their project?

Did you find (fix?) some bottlenecks??

This Class

Lightning talks today: Chris and John

Today is less about concepts

More about learning ot use a given module

So less talk, more coding

Serialization

I'm focusing on methods available in the Python standard library

Serialization is the process of putting your potentially complex (and nested) python data structures into a linear (serial) form .. i.e. a string of bytes.

The serial form can be saved to a file, pushed over the wire, etc.

Persistence

Persistence is saving your python data structure(s) to disk – so they will persist once the python process is finished.

Any serial form can provide persistence (by dumping/loading it to/from a file), but not all persistence mechanisms are serial (i.e RDBMS)

`http://wiki.python.org/moin/PersistenceTools`

Python Literals

Putting plain old python literals in your file

Gives a nice, human-editable form for config files, etc.

Don't use for untrusted sources!!!

Python Literals

Good for basic python types. (can work for your own classes, too – if you write a good `__repr__`)

In theory, `repr()` always gives a form that can be re-constructed.

Often `str()` form works too.

`pprint` (pretty print) module can make it easier to read.

Python Literal Example

```
# a list of dicts
data = [{'this':5, 'that':4}, {'spam':7, 'eggs':3.4}]

In [51]: s = repr(data) # save a string version:

In [52]: data2 = eval(s) # re-construct with eval:

In [53]: data2 == data # they are equal
Out[53]: True

In [54]: data is data2 # but not the same object
Out[54]: False
```

You can save the string to a file and even use `import`

pretty print

```
In [69]: import pprint

In [71]: repr(data)
Out[71]: "[{’this’: 5, ’that’: 4}, {’eggs’: 3.4, ’spam’: 7}, {’f

In [72]: s = pprint.pformat(data)

In [73]: print s
[{’that’: 4, ’this’: 5},
 {’eggs’: 3.4, ’spam’: 7},
 {’bar’: 4.5, ’foo’: 86},
 {’baz’: 6.5, ’fun’: 43}]
```

Pickle

Pickle is a binary format for python objects

You can essentially dump any python object to disk (or string, or socket, or...

`cPickle` is faster than pickle, but can't be customized – you usually want `cPickle`

`http://docs.python.org/library/pickle.html`

Pickle

```
In [87]: import cPickle as pickle
In [83]: data
Out[83]:
[{’that’: 4, ’this’: 5},
 {’eggs’: 3.4, ’spam’: 7},
 {’bar’: 4.5, ’foo’: 86},
 {’baz’: 6.5, ’fun’: 43}]

In [84]: pickle.dump(data, open(’data.pkl’, ’wb’))

In [85]: data2 = pickle.load(open(’data.pkl’, ’rb’))

In [86]: data2 == data
Out[86]: True
```

http://docs.python.org/library/pickle.html

Shelve

A “shelf” is a persistent, dictionary-like object

The values (not the keys!) can be essentially arbitrary Python objects (anything picklable)

NOTE: will not reflect changes in mutable objects without re-writing them to the db.
(or use writeback=True)

If less that 100s of MB – just use a dict and pickle it.

`http://docs.python.org/library/shelve.html`

Shelve

shelve presents a dict interface:

```
import shelve

d = shelve.open(filename)
d[key] = data    # store data at key
data = d[key]    # retrieve a COPY of data at key
del d[key]       # delete data stored at key
flag = d.has_key(key)    # true if the key exists

d.close()        # close it
```

http://docs.python.org/library/shelve.html

LAB

There are two datasets in the code dir:

```
add_book_data.py
add_book_data_flat.py
# load with:
from add_book_data import AddressBook
```

They have address book data – one with a nested dict, one "flat"

- Write a module that saves the data as python literals in a file — and reads it back in
- Write a module that saves the data as a pickle in a file — and reads it back in
- Write a module that saves the data in a shelve — and accesses it one by one.

Lightning Talk

Lightning Talk:

Chris

INI

INI files

(the old Windows config files)

```
[Section1]
int = 15
bool = true
float = 3.1415

[Section2]
int = 32
...
```

Good for configuration data, etc.

ConfigParser

Writing ini files:

```
import ConfigParser
config = ConfigParser.ConfigParser()

config.add_section('Section1')
config.set('Section1', 'int', '15')
config.set('Section1', 'bool', 'true')
config.set('Section1', 'float', '3.1415')

# Writing our configuration file to 'example.cfg'
config.write( open('example.cfg', 'wb') )
```

Note: all keys and values are strings

ConfigParser

Reading ini files:

```
>>> config = ConfigParser.ConfigParser()
>>> config.read('example.cfg')
>>> config.sections()
['Section1', 'Section2']

>>> config.get('Section1', 'float')
'3.1415'

>>> config.items('Section1')
[('int', '15'), ('bool', 'true'), ('float', '3.1415')]
```

http://docs.python.org/library/configparser.html

CSV

CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.

No real standard – the Python csv package more or less follows MS Excel standard
(with other "dialects" available)

Can use delimiters other than commas...
(I like tabs better)

Most useful for simple tabular data

CSV module

Reading CSV files:

```
>>> import csv
>>> spamReader = csv.reader( open(’eggs.csv’, ’rb’) )
>>> for row in spamReader:
...     print ’, ’.join(row)
Spam, Spam, Spam, Spam, Spam, Baked Beans
Spam, Lovely Spam, Wonderful Spam
```

csv module takes care of string quoting, etc. for you

`http://docs.python.org/library/csv.html`

CSV module

Writing CSV files:

```
>>> import csv
>>> spamWriter = csv.writer(open(’eggs.csv’, ’wb’),
                            quoting=csv.QUOTE_MINIMAL)
>>> spamWriter.writerow([’Spam’] * 5 + [’Baked Beans’])
>>> spamWriter.writerow([’Spam’, ’Lovely Spam’, ’Wonderful
```

csv module takes care of string quoting, etc for you

`http://docs.python.org/library/csv.html`

JSON

JSON (JavaScript Object Notation) is a subset of JavaScript syntax used as a lightweight data interchange format.

Python module has an interface similar to pickle

Can handle the standard Python data types

Specializable encoding/decoding for other types – but I wouldn't do that!

Presents a similar interface as `pickle`

`http://www.json.org/`

`http://docs.python.org/library/json.html`

Python json module

```
In [94]: s = json.dumps(data)

Out[95]: ’[{"this": 5, "that": 4}, {"eggs": 3.4, "spam": 7},
           {"foo": 86, "bar": 4.5}, {"fun": 43, "baz": 6.5}]’
    # looks a lot like python literals...
In [96]: data2 = json.loads(s)

Out[97]:
[{u’that’: 4, u’this’: 5},
 {u’eggs’: 3.4, u’spam’: 7},
...
In [98]: data2 == data
Out[98]: True # they are the same
```

(also `json.dump()` and `json.load()` for files)

XML

XML is a standardized version of SGML, designed for use as a data storage/interchange format.

NOTE: HTML is also SGML, and modern versions conform to the XML standard.

XML in the python std lib

`xml.dom`:

`xml.sax`:

`xml.parsers.expat`:

`xml.etree`:
http://docs.python.org/library/xml.etree.elementtree.html

elementtree

The Element type is a flexible container object, designed to store hierarchical data structures in memory.

Essentially an in-memory XML – can be read from written-to XML

an `ElementTree` is an entire XML doc

an `Element` is a node in that tree

`http://docs.python.org/library/xml.etree.elementtree.html`

LAB

```
# load with:
from add_book_data import AddressBook
```

They have address book data – one with a nested dict, one "flat"

- Write a module that saves the data as an INI file
  — and reads it back in
- Write a module that saves the data as a CSV file
  — and reads it back in
- Write a module that saves the data in JSON
  — and reads it back in
- Write a module that saves the data in XML
  — and reads it back in
  — this gets ugly!

Lightning Talk

Lightning Talk:

John

anydbm

`anydbm` is a generic interface to variants of the DBM database

Suitable for storing data that fits well into a python dict with strings as both keys and values

Note: anydbm will use the dbm system that works on your system – this may be different on different systems – so the db files may NOT be compatible! `whichdb` will try to figure it out, but it's not guaranteed

`http://docs.python.org/library/anydbm.html`

anydbm module

Writing data:

```
#creating a dbm file:
anydbm.open(filename, 'n')
```

flag options are:

'r'	Open existing database for reading only (default)
'w'	Open existing database for reading and writing
'c'	Open database for reading and writing, creating it if it doesnt exist
'n'	Always create a new, empty database, open for reading and writing

http://docs.python.org/library/anydbm.html

anydbm module

dbm provides dict-like interace:

```
db = dbm.open("dbm", "c")

db["first"] = "bruce"
db["second"] = "micheal"
db["third"] = "fred"
db["second"] = "john" #overwrite
db.close()

# read it:
db = dbm.open("dbm", "r")
for key in db.keys():
    print key, db[key]
```

sqlite

SQLite: C library provides a lightweight disk-based single-file database

Nonstandard variant of the SQL query language

Very broadly used as as an embedded databases for storing application-specific data etc.

Firefox plug-in:
`https://addons.mozilla.org/en-US/firefox/addon/sqlite-manager/`

python sqlite module

`sqlite3` Python module wraps C lib – provides standard DB-API interface

Allows (and require SQL queries

Can provide high performance, flexible, portable storage for your app

`http://docs.python.org/library/sqlite3.html`

python sqlite module

Example:

```
import sqlite3
# open a connection to a db file:
conn = sqlite3.connect('example.db')

# or build one in-memory
conn = sqlite3.connect(':memory:')

# create a cursor
c = conn.cursor()
```

http://docs.python.org/library/sqlite3.html

python sqlite module

Execute SQL with the cursor:

```
# Create table
c.execute(’’’CREATE TABLE stocks
             (date text, trans text, symbol text, qty real

# Insert a row of data
c.execute("INSERT INTO stocks VALUES (’2006-01-05’,’BUY’,’

# Save (commit) the changes
conn.commit()

# Close the cursor if we are done with it
c.close()
```

python sqlite module

SELECT creates an cursor that can be iterated:

```
>>> for row in c.execute('SELECT * FROM stocks ORDER BY pr
        print row

(u'2006-01-05', u'BUY', u'RHAT', 100, 35.14)
(u'2006-03-28', u'BUY', u'IBM', 1000, 45.0)
...
```

Or you can get the rows one by one or in a list:

```
c.fetchone()
c.fetchall()
```

python sqlite module

Good idea to use the DB-APIs parameter substitution:

```
t = (symbol,)
c.execute(’SELECT * FROM stocks WHERE symbol=?’, t)
print c.fetchone()

# Larger example that inserts many records at a time
purchases = [(’2006-03-28’, ’BUY’, ’IBM’, 1000, 45.00),
             (’2006-04-05’, ’BUY’, ’MSFT’, 1000, 72.00),
             (’2006-04-06’, ’SELL’, ’IBM’, 500, 53.00),
            ]
c.executemany(’INSERT INTO stocks VALUES (?,?,?,?,?)’, purc
```

http://xkcd.com/327/

DB-API

The DB-API spec (PEP 249) is a specification for interaction between Python and Relational Databases.

Support for a large number of third-party Database drivers:

- MySQL
- PostgreSQL
- Oracle
- MSSQL (?)
- .....

`http://www.python.org/dev/peps/pep-0249`

Object-Relation Mappers

Systems for mapping Python objects to tables

Saves you writing that glue code (and the SQL)

Usually deal with mapping to variety of back-ends:
– test with SQLite, deploy with PostreSQL

SQL Alchemy
– `http://www.sqlalchemy.org/`

Django ORM
`https://docs.djangoproject.com/en/dev/topics/db/`

Object Databases

Directly store and retrieve Python Objects.

Kind of like `shelve`, but more flexible, and give you searching, etc.

ZODB:
(`http://www.zodb.org/`)

Durus:
(`https://www.mems-exchange.org/software/DurusWorks/`)

NoSQL

Map-Reduce, etc.

....Big deal for "Big Data": Amazon, Google, etc.

Document-Oriented Storage

- MongoDB (BSON interface, JSON documents)
- CouchDB (Apache):
  - JSON documents
  - Javascript querying (MapReduce)
  - HTTP API

Evaluations

I need to submit evaluations to UW

We'll so that now – then the last LAB

LAB

```
# load with:
from add_book_data import AddressBook
```

- Write a module that saves the data in a dbm datbase — and reads it back in
- Write a module that saves the data in an SQLItE datbase — and reads it back in — helps to know SQL here...

Homework

Send me a copy of your project: due next Sunday

Keep learning about and using Python