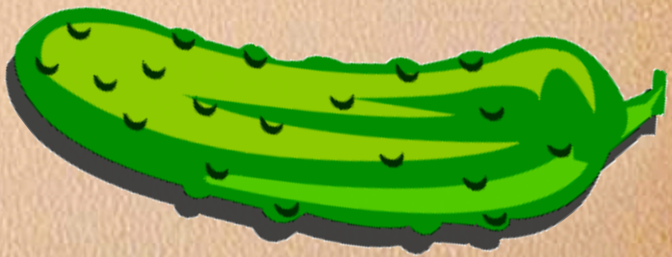
Pickling and Shelves

Dan Fleck



Coming up: What is pickling?

What is pickling?

Pickling - is the process of ***preserving food*** by anaerobic fermentation in brine (a solution of salt in water), to produce lactic acid, or marinating and storing it in an acid solution, usually vinegar (acetic acid). The resulting food is called a pickle. This procedure gives the food a salty or sour taste.

– Wikipeadia

Do we really need to know that?

No… but pickling is done to preserve food for later eating.
Pickling in Python is done to preserve data for later usage.
(And without the salty or sour taste! ☺)

Coming up: Pickling in Python

Pickling in Python

•The pickle module implements an 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

• ‘Unpickling’ is the inverse operation, whereby a byte stream is converted back into an object hierarchy.

•Alternatively known as serialization or marshalling, to avoid confusion, the terms used here are pickling and unpickling. – Python Docs

Coming up: Simply Put

Simply Put

Pickling is a way to store and retrieve data variables into and out from files.

Data variables can be lists, ints, classes, etc… There are some rules, but mostly these all work.

Coming up: How does it work Pickle-Master?

How does it work Pickle-Master?

To pickle something you must:

import pickle

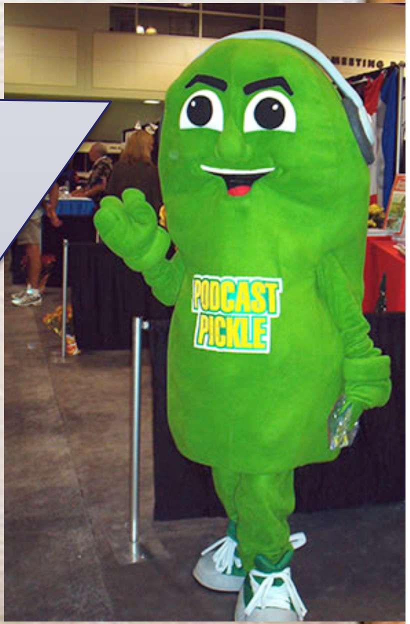
Then to write a variable to a file:

pickle.dump(myString, outfile)

Always remember, Pickle-chips, that the outfile needs to be opened first!

This will store your data into a file

It works with lists and objects also!!



Coming up: Unpickling

Unpickling

To un-pickle something (read the file back into a variable) you must:

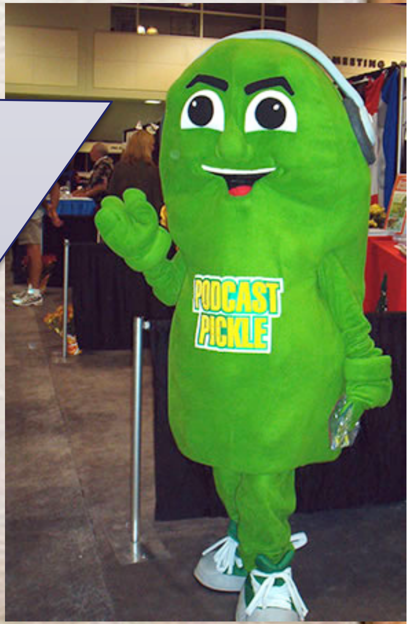
import pickle

Then to write a variable to a file:

myString = pickle.load(inputfile)

Always remember, Pickle-chips, that the inputfile needs to be opened first!

This will recreate the myString data from the file!



Coming up: Example

Example

```
import pickle

def pickler(name1, name2):
    output = open('pickle_jar', 'w')
    pickle.dump(name1, output)
    pickle.dump(name2, output)
    output.close()

def unpickler():
    input = open('pickle_jar', 'r')
    name1 = pickle.load(input)
    name2 = pickle.load(input)
    input.close()
    return name1, name2
```

```
def main():
    nm1 = 'Vlassic'
    nm2 = 'Heinz'

    pickler(nm1, nm2)
    nm1, nm2 = unpickler()

    print 'NM1 is %s, NM2 is %s' %(nm1,
      nm2)

main()

>>> Output: NM1 is Vlassic, NM2 is
      Heinz
```

Notice: You can add multiple variables into a pickle file, but you must read and write them in the exact same order!

Lets try it!

- See example

- Can we read in the variables from the list_pickler.py file?

Example with a list

```
import pickle

def pickler(name1, lst):
    output = open('pickle_jar', 'w')
    pickle.dump(name1, output)
    pickle.dump(lst, output)
    output.close()

def unpickler():
    input = open('pickle_jar', 'r')
    name1 = pickle.load(input)
    myList = pickle.load(input)
    input.close()
    return name1, myList

def main():
    nm1 = 'Vlassic’
    types = ['sweet', 'hot', 'dill', 99]

    pickler(nm1, types)
    nm1, types = unpickler()

    print 'NM1 is %s, types is %s' %(nm1,
      types)

main()

>>> NM1 is Vlassic, types is ['sweet',
      'hot', 'dill', 99]
```

Coming up: Pickling is great… but…

Pickling is great… but…

- Two problems with pickling:
  - Pickle files are not backward compatible
    - The format of the pickle file is not defined by you, so if you change your data (for example, you want to store an additional variable…. you must create new pickle file, all existing pickle files will be broken
  - Pickle files must be accessed in sequential order
    - You cannot say “give me the 3rd variable stored in the file”
    - The solution to this problem is ‘shelves’. I can store many pickle jars on a shelf! ☺

Shelves

- Shelves let you store and retreive variables in any order using pickling.
  - Shelves need a name to get the data. So, it works like a dictionary… given a name, lookup the data

In the pickle dictionary, under the name “shape” store this info!

```
pickles = shelve.open(‘pickle_jar’, writeback=True)
pickles[“variety”] = [“sweet”, “hot”, “dill”]
pickles[“shape”] = [“whole”,”spear”,”chip”]
    pickles.sync() # make sure data is stored to the file
    pickles.close()
```

Coming up: Shelves Example

Shelves Example

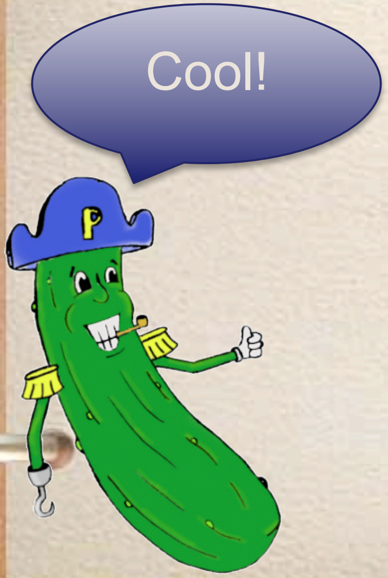
```
import pickle
import shelve

pickles = shelve.open('pickle_jar.dat’, writeback=True)
pickles['variety'] = ['sweet', 'hot', 'dill']
pickles['shape'] = ['whole','spear','chip']
pickles.sync()
pickles.close()

pickles = shelve.open('pickle_jar.dat')
shapes = pickles['shape']
pickles.close()
print 'SHAPES:', shapes

>>> SHAPES: ['whole', 'spear', 'chip']
```

In the pickle dictionary, lookup the data in under “shape”



Shelve Writeback

- shelve.open(*filename*, *writeback=True*) :
  - *By default, when you open a shelf any changes to the dictionary you make are NOT written back into the file.*
  - **writeback=True** means that when you close the file, all changes WILL be written back into the file
  - **shelve.sync()** means writeback data from the cache to the file immediately. This is automatically called during the close() operation

Coming up: Shelve.open(file, flag='c')

Shelve.open(file, flag=‘c’)

- The optional *flag argument:*
  - *'c' to create the database if it doesn’t exist (default)*
  - *'r' to open an existing database for reading only,*
  - *'w' to open an existing database for reading and writing,*
  - *'n', which will always create a new empty database.*

Coming up: What is pickling really doing?

What is pickling really doing?

- The real power in the pickling is converting data into a *formatted string* automatically.
- This is what you did manaully converting things into:
  - john~smith~G001212~12~124
- Pickling just standardizes this for Python

Coming up: Why?

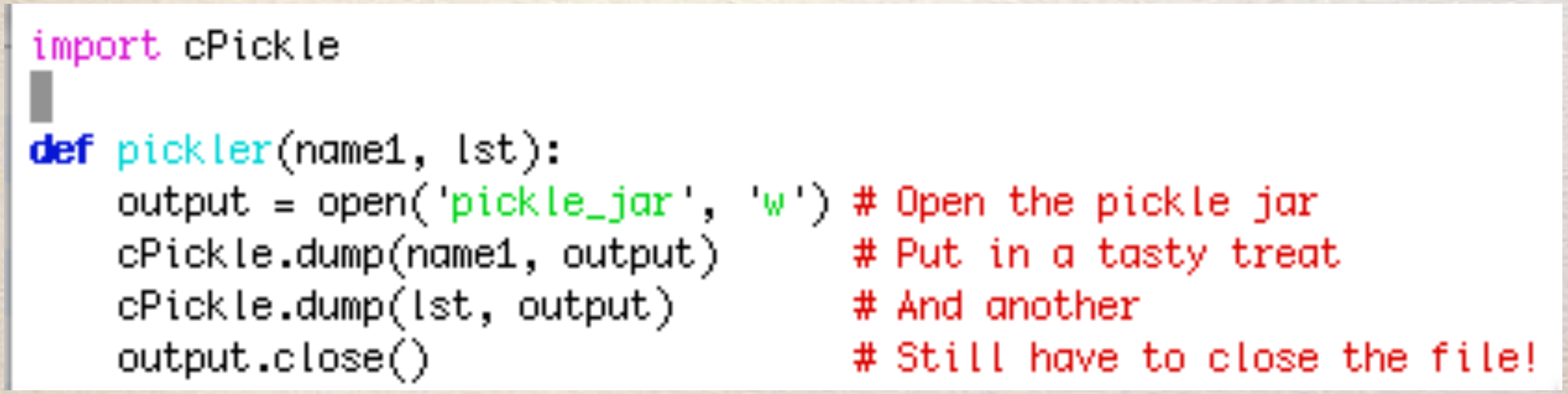
Why?

- The most common use for a string representation of a data structure is to store and retrieve it in files
- However you can also convert it to a string and store it in a database or send it across a network
- All of these help support persistance

Coming up: cPickle or Pickle?

cPickle or Pickle?

- cPickle is a version of the Pickle module written in “C” instead of Python.
- You can use it in the same way, and it is much faster – thus, always use it!



```python
import cPickle

def pickler(name1, lst):
    output = open('pickle_jar', 'w') # Open the pickle jar
    cPickle.dump(name1, output)      # Put in a tasty treat
    cPickle.dump(lst, output)        # And another
    output.close()                   # Still have to close the file!
```

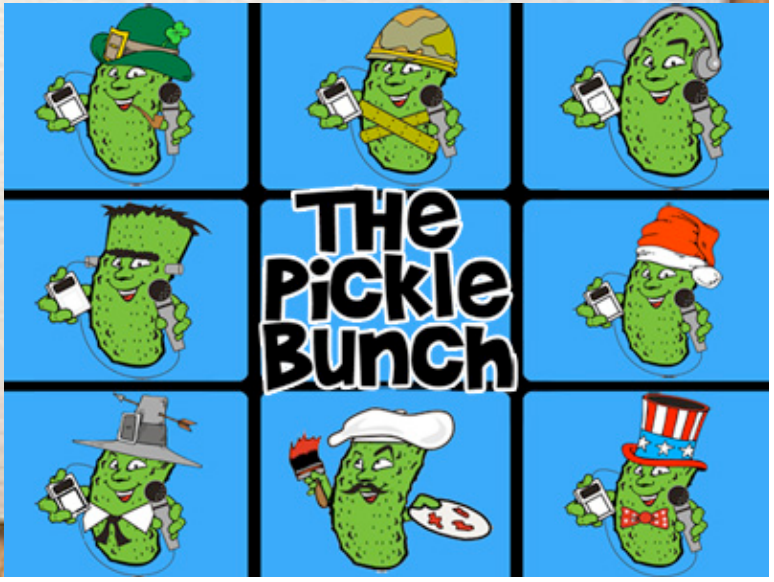
Persistence

- In computer science, persistence refers to the characteristic of data that outlives the execution of the program that created it. Without this capability, data only exists in memory, and will be lost when the memory loses power, such as on computer shutdown. – Wikipedia

- So, pickling supports persistence by allowing you to easily write data to a persistent data store (e.g. a file, or database)

Why is Python so cool?

- Because it uses fun words like pickling.
- Other languages do the same thing as pickling but call it
  - object serialization and deserialization
  - marshalling and unmarshalling

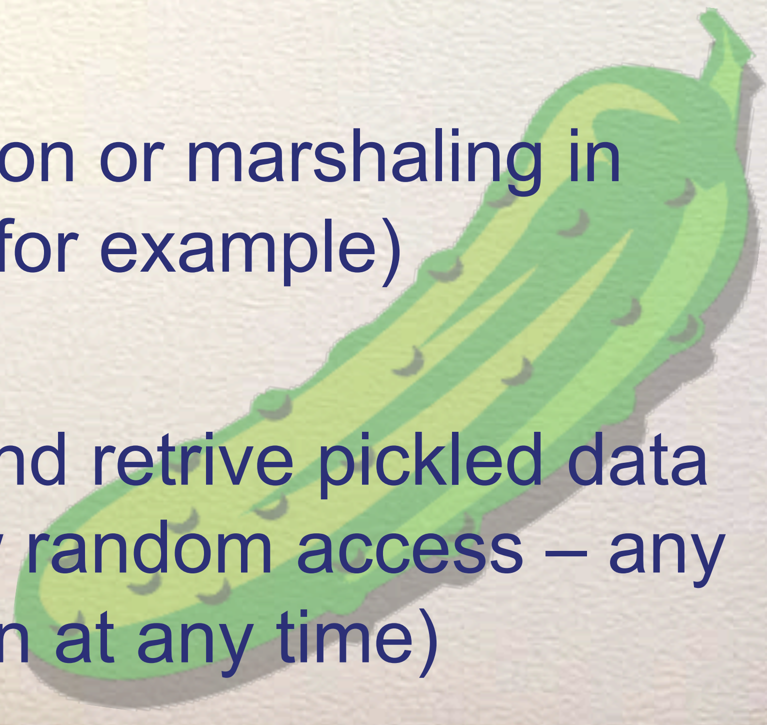


Summary

- Pickling and unpickling let you store/retreive data (lists, int, string, objects, etc..) from files. This is helps you persist data.

- This is called serialization or marshaling in other languages (Java for example)

- Shelves let you store and retrive pickled data in any order (they allow random access – any data can be read/written at any time)



End of presentation