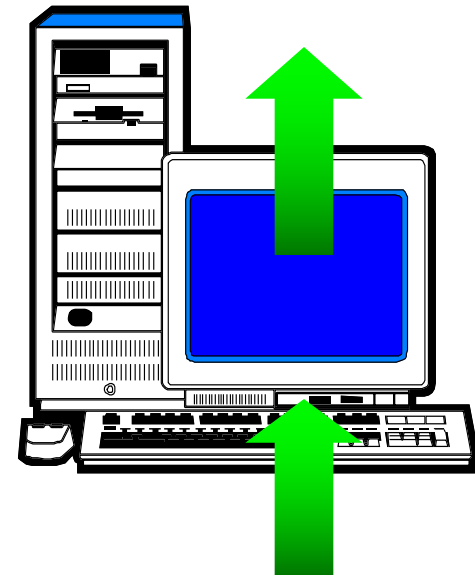


Data Storage and File Handling

Data Storage and File Handling

- **Contents**
 - **File objects**
 - **Reading files**
 - Reading tricks
 - **Writing files**
 - **Standard streams**
 - **More tricks**
 - **Random access**
 - **Python Pickle Persistence**
 - **Pickle Protocols**
 - **Shelves**
 - **Compression**
 - **JSON**
 - **XML**
 - **Database interface overview**
 - Example - SQLite from Python



New file objects

- New file objects are created with the open function
 - An alias for open is file

```
FileObject = open(filename, mode , [bufsize])
```

- Valid open modes:

'r'	open existing file for read (<u>default</u>)
'w'	open file for write, create or overwrite existing file
'a'	open file for append, create if does not exist
'r+'	open existing file for read/write
'w+'	create & truncate file for read/write
'a+'	create & append file for read/write

- File will be closed on exit, or may be closed manually

```
FileObject.close()
```

Reading files into Python

- I/O is done through method calls
 - Create a file object with open

```
infile = open('filename', 'r')
```

- Read *n* bytes
 - May return fewer bytes near end-of-file
 - If *n* is not specified, the entire file is read

```
buffer = infile.read(42)
```

- Read a line

```
line = infile.readline()
```

- The line terminator "`\n`" is read included at the end of a line
- Returns an empty string (False) at end-of-file

Reading tricks

- **Reading the whole file into a variable**
 - **Be careful of the file size**

```
lines = open('brian.txt').read()
llines= open('brian.txt').read().splitlines()

linelist = open('brian.txt').readlines()
```

- **Reading a file sequentially in a loop**
 - **Inefficient**

```
for line in open('lines.txt').readlines():
    print line,
```



- **Use the file object iterator**

```
for line in open('lines.txt') :
    print line,
```

Context Manager – With clause (≥ 2.6)

- The advantage of using a with statement is that it is guaranteed to close the file no matter *how* the nested block exits
 - break
 - exception
 - return

```
with open('text.txt', 'w') as f:  
    f.write('Hi there!')
```

Filter programs - fileinput module

- **The typical behaviour of filter programs is:**
 - **Command line arguments are names of files to process**
 - **If no arguments are given, read standard input instead**
 - **A hyphen on the command line indicates standard input as well**
 - **Examples are grep, sed, awk, wc, lpr, ...**
- **The fileinput module makes it easy to create Python filter programs**
 - **sys.argv is used for the list of filenames (omitting first element)**
 - **fileinput.input() returns each line of the files in turn**
 - **fileinput.filename() returns the current filename being read**
 - **fileinput.filelineno() returns line number in the current file**
 - **and more**

Example program: searching input files

- Python version of egrep

```
import sys, fileinput, re, operator, glob

pattern = sys.argv.pop(1)

sys.argv[1:] = glob.glob(sys.argv[1])

more_files = len(sys.argv[1:])

for line in fileinput.input():
    m = re.search(pattern, line)
    if m:
        if more_files > 1:
            print fileinput.filename()
        print line
```

```
$ egrep.py move *.py
mmove.py # Multiple-move program.
```


Writing to files from Python

- **Open a file handle with open**
 - Specifying write or append

```
output = open('myfile', 'w')  
append = open('logfile', 'a')
```

- **Write a string**
 - Append "\n" to make it a line

```
output.write("Hello\n")
```

- **Write strings from a list**

```
output.writelines(list)
```

Standard streams

- The `sys` module exposes `stdin`, `stdout`, and `stderr` as open file objects

```
import sys
sys.stdout.write("Please enter a value:")
sys.stdout.flush()
reply = sys.stdin.readline()
print "<",reply,"> was input"
```

```
Please enter a value:one
< one
> was input
```

- Simple keyboard (`stdin`) input
 - Use `raw_input()`
 - The `"\n"` terminator is stripped out

```
reply = raw_input("Please enter a value:")
print "<",reply,"> was input"
```

```
Please enter a value:two
< two > was input
```

More tricks

- **print normally writes to stdout, but:**

```
output = open('myfile', 'w')
print >> output, "Hello"
print >> sys.stderr, "Oops, we had an error"
```

- **File writing is normally buffered**

- **To flush the buffer:**

```
output.flush()
```

- **A simple tail -f in Python:**

```
while True:
    line = fo.readline()
    if not line:
        time.sleep(1)
        fo.seek(fo.tell())
    else:
        print line,
```

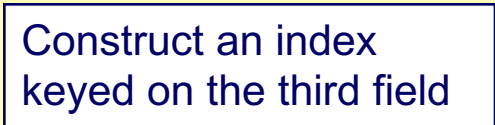
Assumes the file is
open for read

Set the file position
to EOF

Random access

- **Access directly at a position, rather than sequentially**
 - Of limited use with text files - all lines must be the same length
 - Get the current position with the **tell()** method
 - Set the position with **seek(offset[, whence])**

```
fh = open('data.txt', 'r+')
index={}
while True:
    line = fh.readline()
    if not line: break
    fields = line.split(',')
    index[fields[2].rstrip()] = fh.tell() - len(line)
key = raw_input('Enter a key:')
fh.seek(index[key])
print fh.readline()
fh.close()
```



Python pickle persistence

- **Pickling** converts Python objects into a stream of bytes
 - Usually written to a file, or across a network
 - **CPickle** module is faster than **pickle** module for large data sets
 - Functionality and operation is the same for most applications

```
import pickle

caps = {'Australia': 'Canberra', 'Eire' : 'Dublin',
        'UK'       : 'London',   'US'   : 'Washington'}

outp = open('capitals.p', 'wb')
pickle.dump(caps, outp)
outp.close()
```

Using 'b' to
indicate binary

```
import pickle

inp = open('capitals.p', 'rb')
caps = pickle.load(inp)
inp.close()
```

Pickle protocols

- **Three different formats (protocol versions) may be used**
 - 0 ASCII, backwards compatible with earlier versions of Python
 - 1 Binary format, also backwards compatible
 - 2 Added in Python 2.3. Efficient pickling of classes
 - **The pickle module has protocol attributes**
 - **HIGHEST_PROTOCOL and DEFAULT_PROTOCOL**
- ```
import pickle

outp = open('capitals.p', 'wb')
pickle.dump(caps, outp, pickle.HIGHEST_PROTOCOL)
outp.close()
```
- **None of these protocols are secure**
    - **Python pickle persistence for personal programs only please!**

# pickle and cPickle

---

- **The cPickle** module implements the same algorithm, in C instead of Python.
- It is many times faster than the Python implementation, but does not allow the user to subclass from Pickle
- Python has a more primitive serialization module called marshal, but in general pickle should always be the preferred way to serialize
- Python objects. marshal exists primarily to support Python's .pyc files

# Build some shelves

- We often wish to dump keyed structures
- A **shelve** is a keyed pickle dumped to a database
  - Looks just like an ordinary dictionary
  - Uses a simple bundled database system, usually dbm
  - You only need methods: `open()`, `sync()`, `close()`

```
import shelve
db = shelve.open('capitals')
db['UK'] = 'London'
...
db.close()
```

`close()` does a `sync()`  
(like a commit)

```
db = shelve.open('capitals')
print db['UK']
db.close()
```



# Compression

- **The standard library includes gzip**
  - **Open the file using the gzip method**
    - Same arguments as regular open
  - **Then call the usual methods on the file handle**
  - **Often used with pickles**

```
import pickle, gzip

...
outp = gzip.open('capitals.pgz', 'wb')
pickle.dump(caps, outp)
outp.close()
```

```
import pickle, gzip

inp = gzip.open('capitals.pgz', 'rb')
caps = pickle.load(inp)
inp.close()
```

# JSON (new in 2.6)

- **JSON (JavaScript Object Notation)**
  - A lightweight data interchange format inspired by JavaScript object literal syntax

```
import json

list = ['foo', {'bar': ('baz', None, 1.0, 2)}]

with open("dict.json", 'w') as d:
 json.dump(list, d)
```

```
import json

with open("dict.json", 'r') as d:
 list = json.load(d)
```

**Note:** Unlike pickle and marshal, JSON is not a framed protocol so trying to serialize more objects with repeated calls to `dump()` and the same *fp* will result in an invalid JSON file

# Default JSON Encode/ Decode

## json.JSONDecoder

| JSON          | Python    |
|---------------|-----------|
| object        | dict      |
| array         | list      |
| string        | unicode   |
| number (int)  | int, long |
| number (real) | float     |
| true          | True      |
| false         | False     |
| null          | None      |

## json.JSONEncoder

| Python           | JSON   |
|------------------|--------|
| dict             | object |
| list, tuple      | array  |
| str, unicode     | string |
| int, long, float | number |
| True             | true   |
| False            | false  |
| None             | null   |

# Custom JSON Encode/ Decode

```
class Account:
 def __init__(self, initial):
 self.__balance = initial
 def deposit(self, amt):
 self.__balance = self.__balance + amt
 def withdraw(self, amt):
 self.__balance = self.__balance - amt
 def getbalance(s):
 return s.__balance
```

```
def as_account(dct):
 if 'Balance' in dct:
 return Account(dct['Balance'])
 return dct
```

```
with open('accounts.json', 'r') as d:
 accs=json.load(d,object_hook=as_account)
```

```
class AccountEncoder(json.JSONEncoder):
 def default(self, obj):
 if isinstance(obj, Account):
 return {'Balance':obj.getbalance()}
 return json.JSONEncoder.default(self, obj)
```

```
accs = [Account(123),Account(200),Account(300)]
with open("accounts.json", 'w') as d:
 json.dump(accs,d,cls=AccountEncoder)
```

# XML Parsers Overview

---

- **Two most basic and broadly used APIs to XML data are:**
  - **Simple API for XML (SAX) –**
    - Here, you register callbacks for events of interest and then let the parser proceed through the document.
    - This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk and the entire file is never stored in memory.
  - **Document Object Model (DOM) API –**
    - This is a World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

# Database interface overview

---

- **Available for most popular databases**
  - Database drivers are expected to conform to a standard
  - Import the required database driver, then call standard methods
  - Most drivers include extensions
- **Two main objects**
  - **Connection object**
    - Connect to the database
    - Create the cursor object
    - Transaction management
  - **Cursor object**
    - Execute queries on this
- **Python is shipped with SQLite**

## Example – XML DOM

```
from xml.dom.minidom import parse
import xml.dom.minidom

Open XML document using minidom parser DOMTree =
xml.dom.minidom.parse("movies.xml")
collection = DOMTree.documentElement

if collection.hasAttribute("shelf"):
 print "Root element : %s" %
collection.getAttribute("shelf")

Get all the movies in the collection
movies = collection.getElementsByTagName("movie")
```

# Example - SQLite from Python

- **Use the sqlite3 module**
  - **Bundled with the Python release**

```
import sqlite3

db = sqlite3.connect('whisky')

cur = db.cursor()

cur.execute('SELECT BRANDS.BNAME, REGION.RNAME \
 FROM BRANDS,REGION \
 WHERE REGION.REGION_ID = BRANDS.REGION_ID \
 ORDER BY BRANDS.BNAME;')

for row in cur.fetchall():
 print "%-30s %-30s" % (row[0], row[1])

db.close()
```



# Binary files - struct.pack/unpack

- **Binary file formats don't map to Python variable types**
  - Unless they were written using Python (like pickle)
  - Typically they might be written using C/C++, or similar
- **Convert to/from primitive types using pack and unpack**

```
import struct

fIn = open("bindata", "rb")
data = fIn.read(1024)
fIn.close()

clean = struct.unpack("i id 80s", data)
print clean
txt = clean[3].rstrip('\x00')
print txt
```

```
typedef struct {
 int a;
 int b;
 double c;
 char name[80];
} data;
```

```
(37, 42, 3.142, b'Hollow world!\x00\x00\x00\x00\x00...')
Hollow world!
```

# Summary

---

- **A file object is created by calling open**
- **Read from a file:**
  - Call read, readline, or readlines methods
  - Or invoke the file iterator in a for loop
- **Writing to a file:**
  - Call write or writelines methods
  - `print` can also be used
  - Good practice to close the file as soon as possible
- **Many other methods available**
- **Objects can be preserved by pickling them**

