

Python Course: Lecture 7

January 19, 2006

1 Input and Output

- There are many different forms of computer input: keyboard, joystick, network, camera, input from a hardware sensor.
- There are many different forms of computer output: text, graphics, print, sound, network, robot.
- The lowest common denominator is what you can put on a desktop: keyboard, mouse and screen.
- The even lower common denominator is what the state of the art hardware was fifteen years ago: a command line interface with text input from the keyboard and text output to the screen.
- Everything we do in this class will focus on this mode of UI, which is still the most common one for research software.
- For our purposes Python will be a language that takes a file as input and generates a different but related file as output.
- Python does have decent GUI support (via the TK system), but we won't get into it here.

2 Files

- Computers have files on them. It's how they save state.
- Files have names.
- Python uses the same file handling paradigm as any other computer language: you specify the name of a file to open and get back a handle. The open file has a pointer that moves forward as you read from it. When you're done you close the handle.

- You open a file with the `open` or `file` commands. The latter is newer.

```
>>> h = file('lecture06.tex')
>>> h
<open file 'lecture06.tex', mode 'r' at 0xb731d8a0>
>>> dir(h)
['__class__', '__delattr__', '__doc__', '__getattr__', '__hash__',
 '__init__', '__iter__', '__new__', '__reduce__', '__reduce_ex__',
 '__repr__', '__setattr__', '__str__', 'close', 'closed', 'encoding',
 'fileno', 'flush', 'isatty', 'mode', 'name', 'newlines', 'next', 'read',
 'readinto', 'readline', 'readlines', 'seek', 'softspace', 'tell',
 'truncate', 'write', 'writelines', 'xreadlines']
```

- If you do something wrong, Python will raise an `IOError`.
- You can do several things with a file, but the most basic are `read`, `write` and `close`.
- Calling the `read` function reads in the entire contents at once. This may be what you want, but be conscious of memory considerations for large files. You can also specify an optional number of bytes to read at a time. (This works the same way it does in C/C++.)
- Calling `readline` reads a single line from the file. It returns a string and advances the file pointer.
- Calling `readlines` makes multiple calls to `readline` and returns a list of strings.
- A file handle also works as a generator object. Putting it as the argument of a `for` loop causes it to return one line at a time. This is often the best way to handle text files of unknown size.
- Note the polymorphism: passing an open file handle to a function is the same as passing a list of strings. The tokenization functions you write for your homework can make use of this fact.
- To write to a file, open it with `'w'` mode and use the `write()` method.
- Call `close` when you're done with a handle.
- In computational linguistics you're usually dealing with text files (because you're often dealing with text). To get a representative sample of text you frequently have to deal with a large volume of data, so you frequently do have to be aware of memory issues, not reading in the whole file at once, etc.
- A useful architecture question to ask yourself is what is the smallest usable chunk of data I can read off the disk? If the answer is "a line", enumerate over lines in a file. If the answer is something larger than a line, use iterators to virtualize over file enumeration.

3 The File System

- There are lots of little details about file handling that you should read the documentation for. I'll point out a couple basic tools.
- File names are grouped hierarchically into directories.
- File names have different parts: path, root, extension.
- Python provides standard means of interacting with the file system and parsing file names.
- The `os` module contains functions that deal with basic operating system abstractions like files and processes. The paradigm and names mirror UNIX conventions. It tries to provide the functionality of sitting at a DOS/UNIX prompt.
- Some essential functions: `os.getcwd`, `os.listdir`, `os.mkdir`, `os.remove`

```
>>> os.listdir('.')
['CVS', 'lecture02.tex', 'lecture07.tex', 'lecture06.log',
 'lecture06.aux',
 'lecture01.tex', 'auto', 'lecture06.dvi', 'lecture03.tex',
 'lecture04.tex',
 'lecture05.tex', 'lecture06.tex']
```

- The `glob` module lets you do basic UNIX-style file globbing.

```
>>> glob.glob('*.tex')
['lecture02.tex', 'lecture07.tex', 'lecture01.tex', 'lecture03.tex',
 'lecture04.tex', 'lecture05.tex', 'lecture06.tex']
```

- Some high-level file and directory copying operations are the in `shutil` module: `shutil.copy`, `shutil.copytree`, `shutil.rmtree`.
- `os.path` does basic file name manipulation: `os.path.basename`, `os.path.join`, `os.path.splitext`.
- Note, `os.path` is the first example we've seen of a nested module namespace.
- Aside: there is a class of Guaranteed Solved Problems that will have implementations in any computer language you use. So if you find yourself writing code to do one of these tasks, you're making a mistake. File name manipulation is the classic example. (Others are HTML parsing and linear algebra.) Part of expert programmer experience is having a feel for what these areas are. (Generally linguistic applications are not one of them.)
- All of this illustrates Python's utility as a glue language.

4 Gzip Files

- Python has native support for gzipped files.

```
>>> import gzip
>>> print gzip.open('file.gz').read()
```

- Basically all you need to know is to call `gzip.open` in place of `open`.
- This is very helpful for dealing with large quantities of data.

5 Command Line

- Files are a standard program-to-OS channel of communication. Command line arguments are the standard program-to-user channel of communication.
- Even though the focus of this class is on writing library modules, you also have to be able to write scripts.
- When you run a program from the command line, you give it arguments.
- Ultimately this is a list of strings, which is how Python deals with them.
- The most basic Python command line parsing script.

```
import sys

print sys.argv
```

This runs like so

```
> python cmdline.py cat dog mouse
['cmdline.py', 'cat', 'dog', 'mouse']
```

- First we import the `sys` (short for “system”) module, which contains functions and values that interact strongly with the Python interpreter.
- `sys.argv` is a list of command line arguments.
- `sys.argv` is empty when running in the interpreter.
- The first element is the path to the script. The rest are the command line arguments if any.
- This is identical to the `main(argv)` variable in C/C++.

- Common idioms...
- Frequently you don't care about the name of the script, so you'll see `sys.argv[1:]`.
- Even in a script, do all your work inside a `main` function.

```
def main(args):
    print args

if __name__ == '__main__':
    import sys
    main(sys.argv[1:])
```

This allows you to work with your script code inside the interactive mode (and the debugger).

- If you know exactly how many arguments your program will use, you frequently do direct assignment.

```
arg1, arg2 = sys.argv[1:]
```

This will generate an error if the user doesn't enter the proper number of arguments.

- There is built-in support for more sophisticated command line handling (optional parameters, help printing) that we'll talk about at the end of the course.

6 I/O Streams

- Python provides support for the standard UNIX-style input and output streams. They are treated as file handles that are always open.
- `sys.stdin`, `sys.stdout`, `sys.stderr`

7 Serialization

- You can use files to save data to the disk, but the data has to be in a certain format.
- For example, say you've got consonant cluster hash table data.

```
>>> h = {'bb':{('a', 'a'):2, ('e', 'e'):1}}
```

In a lower-level language like C++, you'd have to invent a file format for your data. This is significant coding work.

- The task of defining a data format that allows in-memory objects to be written to and from the disk is called *marshalling* or *serialization*.
- Any Python object can be serialized automatically.
- Serialization is handled via the `cPickle` module. (I guess there's a food preservative metaphor at work here.)
- `cPickle.dumps` allows you to convert an arbitrary Python object to a string.

```
>>> cPickle.dumps(h)
"(dp0\nS'bb'\np1\n(dp2\n(S'a'\np3\ng3\ntp4\nI2\ns(S'e'\np5\ng5\ntp6\nI1\nss."
>>> print cPickle.dumps(h)
(dp0
S'bb'
p1
(dp2
(S'a'
p3
g3
tp4
I2
s(S'e'
p5
g5
tp6
I1
ss.
```

- This is fairly opaque, and becomes only more so as you move to more complicated data structures, but that doesn't matter, because it isn't intended to be human-readable.
- The `cPickle.loads` inverts the pickling.

```
>>> h
{'bb': {'(a', 'a'): 2, ('e', 'e'): 1}}
>>> p = cPickle.dumps(h)
>>> p
"(dp1\nS'bb'\np2\n(dp3\n(S'a'\nS'a'\ntp4\nI2\ns(S'e'\nS'e'\ntp5\nI1\nss."
>>> i = cPickle.loads(p)
>>> i
{'bb': {'(a', 'a'): 2, ('e', 'e'): 1}}
```

- The unpickled entity is a separate object.

```
>>> x = [1,2]
>>> y = cPickle.loads(cPickle.dumps(x))
>>> y
[1, 2]
>>> x[0] = 10
>>> x
[10, 2]
>>> y
[1, 2]
```

- There are choices of protocol and a binary format that can be controlled by optional arguments. See the documentation.
- You can get fancy and bundle pickle and gzip functionality into a single function call. Google “zip and pickle” and hit “I’m feeling lucky” and you get my recipe. (Though it’s not very good.)
- There’s a pure Python version called `pickle` that trades speed for some advanced functionality. Stick with `cPickle` until you know you’ve hit its limit.
- You can also write directly to and from files: `cPickle.dump`, `cPickle.load`.
- These functions are the way to archive medium complex data structures.
 - For very simple data structures (e.g. tables of text) you may be better off doing the formatting yourself.
 - For very complicated data structures you might want a full-blown database backend.
- Serialization can also be used as a form of inter-process-communication.
- Pickling objects becomes even more useful after we’ve talked about classes, because they provide an even richer concept of data structure.