file-io-ajf

February 9, 2015

Contents

0.1 Session 3: File Input/Output

0.1.1 File I/O: Basic

- Data on a computer is usually stored in **files**
- From the view of the operating system, a file is just a sequence of bits that is given a name
- What data is stored in a file and how exactly it is stored in a file is defined in a **file format**
- The file format defines what the bits mean to the program that is reading/writing the file
- ***Note:*** The **file extension** (e.g. whether the name of a file ends in **.txt** or **.doc** does not determine the file format (it is just a name) – but it makes sense to name files according to their format

0.1.2 File I/O: Writing to a Text File

- A very common and useful file format is one where the sequence of bits is interpreted as sequence of characters
- This conversion is performed with respect to a character set (such as ASCII or UTF-8, but let's not worry about that here. . .)
- In Python, such **text files** can be manipulated very easily, by reading/writing their contents to/from strings
- Using the `open()` function one can obtain a reference to a **file** object that provides methods for reading and writing (e.g. `read()` and `write()`)

0.1.3 File I/O: Text Files

File I/O: Writing to a text file:

Opening a text file for writing

```
In [18]: f = open('my_first_file.txt', 'w')
```

```
In [19]: f.write('Hello, this is my first file!')
In [20]: f.close()
```

We can now read this file again:

```
In [21]: f = open('my_first_file.txt', 'r')

In [22]: line = f.readline()

In [23]: print line

Hello, this is my first file!

In [24]: f.close()
```

Write can be called multiple times to write more data:

```
In [46]: f = open("animals.txt", "w")

In [47]: for animal in ["Animal\tFood", "Sloth\tLeaves", "Chicken\tCorn", "Ant_eater\tAnts", "Penguin\t
             f.write("%s\n" % animal)

In [48]: f.close()
```

0.1.4 File I/O: Reading from a Text File:

Reading the content of a text file using the `readlines()` function: The `readlines()` function reads an entire text file into a list of strings, where each list entry corresponds to a line in the file

```
In [64]: f = open("animals.txt", "r")

In [65]: lines = f.readlines()

In [66]: print lines

['Animal\tFood\n', 'Sloth\tLeaves\n', 'Chicken\tCorn\n', 'Ant_eater\tAnts\n', 'Penguin\tFish\n', 'Armadi

In [67]: len(lines)

Out[67]: 6

In [68]: f.close()
```

Notice the difference between the readlines and readline functions

```
In [69]: f = open("animals.txt", "r")

In [70]: line = f.readline()

In [71]: print line

Animal        Food

In [72]: f.close()
```

Because the entire file is first read into memory, this can be slow or unfeasible for large files
Now print each line:

```
In [73]: for l in lines:
             print l
```

```
Animal        Food

Sloth        Leaves

Chicken        Corn

Ant_eater        Ants

Penguin        Fish

Armadillo        Ice_cream
```

```
In [74]: for l in lines:
             print l.rstrip()
```

```
Animal        Food
Sloth        Leaves
Chicken        Corn
Ant_eater        Ants
Penguin        Fish
Armadillo        Ice_cream
```

The `print` statement inserts `\n` after automatically, without removing the already present `\n` characters with `rstrip()` we end up with empty lines!

Reading the content of a text file line by line: Because processing each line in a file is such a common operation, Python provides the following simple syntax

```
In [12]: f = open("animals.txt", "r")
```

```
In [13]: for line in f:
             print line.rstrip()
```

```
Animal        Food
Sloth        Leaves
Chicken        Corn
Ant_eater        Ants
Penguin        Fish
Armadillo        Ice_cream
```

```
In [14]: f.close()
```

This iterates over the file line by line instead of reading in the whole content in the beginning!

And because python makes your life easy, here is an even shorter version:

```
In [62]: with open("animals.txt", "r") as infile:
             for line in infile:
                 print line.rstrip()
```

```
Animal        Food
Sloth        Leaves
Chicken        Corn
Ant_eater        Ants
Penguin        Fish
```

Using `with` removes the necessity to call the `close()` function on your file object!

0.1.5 File I/O: Transforming a File:

- When working with data provided by other programs (and/or other people), it is often necessary to convert data from one format to another

The file that we wrote contained columns separated by tabs; what if we need commas?

```
In [68]: import os
         with open("animals.txt", "r") as infile:
             with open("animals.csv", "w") as outfile:
                     for line in infile:
                         outfile.write(",".join(line.split()))
                         outfile.write(os.linesep)  # Writes \n for us!
```

Lets check everything worked…

```
In [69]: with open("animals.csv", "r") as infile:
                 for line in infile:
                     print line.rstrip()
```

```
Animal,Food
Sloth,Leaves
Chicken,Corn
Ant_eater,Ants
Penguin,Fish
```

Looking good!

0.1.6 File I/O Pickling:

- Text files are convenient when data needs to be exchanged with other programs
- However, getting the data in/out of text files can be tedious
- If we know we only need the data within Python, there is a very easy way to write arbitrary Python data structures to compact binary files
- This is generally referred to as **serialization**, but in Python-lingo it's called **pickling**
- The **pickle** module and it's more efficient **cPickle** version provide two functions, `dump()` and `load()`, that allow writing and reading arbitrary Python objects

```
In [70]: from cPickle import dump, load
```

```
In [71]: l = ["a", "list", "with", "stuff", [42, 23, 3.14], True]
```

```
In [74]: with open("my_list.pkl", "wb") as f:
             dump(l, f)
```

```
In [77]: with open("my_list.pkl", "rb") as f:
             l = load(f)
         l
```

```
Out[77]: ['a', 'list', 'with', 'stuff', [42, 23, 3.14], True]
```

0.1.7 File I/O Checking for Existence:

- Sometimes a program needs to check whether a file exists
- The `os.path` module provides the `exists()` function

```
In [75]: from os.path import exists
In [77]: if exists("my_first_file.txt"):
             print "my first file exists!"
         else:
             print "No first file found!"
```

```
my first file exists!
```

```
In [82]: if exists("lockfile"):
             print "Lockfile exists!"
         else:
             print "No lockfile found!"
```

```
No lockfile found!
```

In general, the `os` and `os.path` modules provide functions for manipulating the file systems. Don't try to reinvent the wheel - most things exist already in the Python standard library!

0.1.8 File I/O: Reading from the Web:

- In Python, there are several other objects that behave just like text files
- One particularly useful one provides file-like access to resources on the web: the `urlopen()` method in the `urllib2` module

```
In [78]: import urllib2
```

```
In [79]: URL = "http://www.gutenberg.org/cache/epub/28885/pg28885.txt"
```

```
In [80]: if not exists("alice.txt"):
             f = urllib2.urlopen(URL)
             with open("alice.txt", "wb") as outfile:
                 outfile.write(f.read())
```

```
In [83]: print open("alice.txt").readlines()[970]
```

```
middle of one! There ought to be a book written about me, that there
```

```
In [95]: print ''.join(open("alice.txt").readlines()[970:975])
```

```
middle of one! There ought to be a book written about me, that there
ought! And when I grow up, I'll write one--but I'm grown up now," she
added in a sorrowful tone; "at least there's no room to grow up any more
_here_."
```

```
In [101]: with open("alice.txt", "rb") as infile:
              book = infile.readlines()
              print "".join(book[1000:1005])
```

```
hand, and made a snatch in the air. She did not get hold of anything,
but she heard a little shriek and a fall, and a crash of broken glass,
from which she concluded that it was just possible it had fallen into a
cucumber-frame, or something of the sort.
```

0.1.9 File I/O Multiple Files:

The `glob` module provides an easy way to find all files with certain names (e.g. all files with names that end in `.txt`)

```
In [103]: import glob

In [106]: text_files = glob.glob("*.txt")

In [107]: for t in text_files:
              print t

alice.txt
animals.txt
multiplication_table.txt
my_first_file.txt
```

0.1.10 File I/O Terminal streams:

- The terminal input/output streams can also be accessed like files using the `stdin` and `stdout` objects from the `sys` module

```
In [84]: import sys

In [85]: sys.stdout.write("Another way to print!\n")

Another way to print!

In [ ]:
```