Physics Without Frontiers

The Abdus Salam International Centre for Theoretical Physics

Practical Programming
in Python

Inspired by 'Practical Programming' by Paul Gries, Jennifer Campbell, Jason Montojo

Lecture 10: Summary & Exercises
Reading & Writing Files

Reading Files, Files from the Internet, Writing Files, File Formats, Parsing Files

---

"Keep knowledge in plain text."
– The Pragmatic Programmer

Lecture 10: Summary

In this lecture you learned the following:

- When files are opened and read, their contents are commonly stored in lists of strings.
- Data stored in files is usually formatted in one of a small number of ways, from one value per line to multi-line records with explicit end-of-record markers. Each format can be processed in a stereotypical way.
- Data processing programs should be broken into input, processing, and output stages so that each can be reused independently.
- Files can be read (content retrieved), written to (content replaced), and added to (new content appended). When a file is opened in writing mode and it doesn't exist, a new file is created.
- Data files come in many different formats, so custom code is often required, but we can reuse as much as possible by writing helper functions.

Example & Data

You can find the example data and Python modules mentioned in the exercises in the directory `~/practical_programming`.

Lecture 10: Exercises

When writing code, only use Python concepts that have been introduced in the lectures already.

Exercise 1:

Write a program that makes a backup of a file. Your program should prompt the user for the name of the file to copy and then write a new file with the same contents but with .bak as the file extension.

Exercise 2:

Suppose the file `alkaline_metals.txt` contains the name, atomic number, and atomic weight of the alkaline earth metals:

```
beryllium 4 9.012
magnesium 12 24.305
calcium 20 20.078
strontium 38 87.62
barium 56 137.327
radium 88 226
```

Write a `for` loop to read the contents of `alkaline_metals.txt` and store it in a list of lists, with each inner list containing the name, atomic number, and atomic weight for an element. (Hint: Use string.split.)

Exercise 3:

All of the file-reading functions we have seen in this chapter read forward through the file from the first character or line to the last. How could you write a function that would read backward through a file?

Exercise 4:

To process whitespace-delimited data from `lynx.dat`, we used the "For Line in File" technique to process data line by line, breaking it into pieces using string method split. Rewrite function `process_file` in a copy of `tsdl.py` to skip the header as normal but then use the Read technique to read all the data at once.

Exercise 5:

Modify the function `process_file` in a copy of `tsdl.py` so that it can handle files with no data after the header.

Exercise 6:

Modify the PDB file reader from the Multiline Records slides, such that it can deal with missing end markers (all lines with `END` are removed).

Exercise 7:

Modify the PDB file reader from the Multiline Records slides, such that it ignores blank lines and comment lines in PDB files. A blank line is one that contains only space and tab characters (that is, one that looks empty when viewed). A comment is any line beginning with the keyword `CMNT`.

Exercise 8:

Modify the PDB file reader to check that the serial numbers on atoms start at 1 and increase by 1. What should the modified function do if it finds a file that doesn't obey this rule?

Exercise 9:

Create a module pdb.py that provides functions to read PDB files. The functions should be able to handle PDB files with or without end markers as well as empty lines and comments (as defined in the previous exercises).

Exercise 10:

Change the module from the previous exercise such it can be executed as a program from the command line. The program should take the path to the PDB file as a command line argument and print the molecule names and the number of atoms in each molecule.