CSC461 Programming Languages - Fall 2016
Programming Assignment #1: File Rename Utility in Python

Scripting languages such as Python are ideally suited for writing general-purpose utility programs. This assignment will give you experience with many of the basic features of Python, including string handling, regular expressions, and the operating system module.

Write a program in Python to perform batch file renaming operations (and more). The user specifies a variety of options on the command line, which are applied to a list of filenames.

Usage: `python rename.py` *`options file1 file2 ...`*

Renaming options:

-l, --lower — convert filenames to lowercase

-u, --upper — convert filenames to uppercase

-t *n*, --trim *n* — positive *n*: trim *n* chars from the start of each filename
negative *n*: trim *n* chars from the end of each filename

-r *oldstring newstring*, --replace *oldstring newstring*
replace *oldstring* with *newstring* in filenames
strings are treated as regular expressions (and generally quoted)

-n *countstring*, --number *countstring*
rename files in sequence using *countstring*
#'s in *countstring* become numbers; e.g., ## becomes 01,02,...

Other (non-renaming) options:

-h, --help — print a help message

-v, --verbose — print old and new filenames during processing

-p, --print — only print old and new filenames, do not rename

-i, --interactive — interactive mode, ask user prior to processing each file

-d, --delete — delete files

-dt, --touch — "touch" files (update date/time stamp to current date/time)

-D *DDMMYYYY*, --date *DDMMYYYY* — change file datestamps

-T *HHMMSS*, --time *HHMMSS* — change file timestamps

Command options should be executed in the order given, and may be repeated (e.g., multiple string replacements). Only reasonable combinations of options need be tested (e.g., it makes no sense to combine file deletion with most other options). Filenames are listed after the options, and may include UNIX-style wildcards, even on Windows (hint: *globbing*). Issue appropriate warnings for incorrect usage, when files cannot be found, renaming fails, etc.

Notes

- The Regular Expression HowTo document (https://docs.python.org/3.5/howto/regex.html) has a good introduction to Python regular expressions.
- The *argparse* and *getopt* modules process command-line arguments.
- The *os*, *stat*, and *datetime* modules contain useful file-handling functionality.
- To receive full credit, your code must be readable, modular, nicely formatted, and adequately documented, as well as complete and correct. It must build and run successfully using a reasonably current Python 3 distribution (version 3.4 or later) on both Windows and Linux. If your program does not run correctly, indicate why. This will make it easier to give you partial credit.
- Modularity includes writing a multi-file Python program (not just a script of commands), with functions to accomplish logical tasks. Documentation includes docstring comments at the start of each function, plus inline comments for further explanation of the code.
- When you are finished writing, testing, and debugging your program, submit your source code using the *Submit It!* link on the MCS Department Website. Usage is self-explanatory: enter your name, choose the instructor (Weiss) and click "Select Instructor", choose the appropriate course (CSC461), browse to the filename you wish to submit, and click "Upload". Multi-file programs should be packaged in a zip or tar archive for submission.
- Submit your program by midnight on the due date (Thursday September 22) in order to receive credit for this assignment. Late programs will not be accepted for partial credit unless prior arrangements have been made with the instructor. If you have any problems with the submit program, report them to your instructor by email, and submit your program via email attachment.
- You must work in teams of two students on this assignment. Teams should make one joint submission, not individual submissions for each team member. Each group member should also submit a teamwork evaluation form, assessing both distribution of workload and team interactions.

Partners for PA#1

Wesley Adams and Savoy Schuler
Aaron Alphonsus and Alexander Iverson
Arjun Ayyangar and Krey Warshaw
Charles Bonn and Matthew De Young
Mark Buttenhoff and Micah Picasso
Hannah Carroll and Jared Johnson
John Colton and Jessica Thompson
Cheldon Coughlen and Andrew Fagrey
Lyndon Engel and Kenneth Petry
Wyatt Engel and Daniel Hodgin
Chance Haka and Johnathan Westlund
James Hinker and Cassidy Vollmer
Andrew Housh and Brady Shimp
Tyler Kinnear and Samuel Williams
Andrew Stelter and Leif Torgersen
Sierra Wahlin-Rhoades and Peter Yamaguchi

Example

Suppose you have a directory full of files from Tolkien's *Lord of the Rings* audio books:

```
1_ Book 1 - Chapter 01 - A Long-Expected Party.mp3
1_ Book 1 - Chapter 02 - The Shadow of the Past.mp3
. . .
1_ Book 1 - Chapter 12 - Flight to the Ford.mp3
2_ Book 2 - Chapter 01 - Many Meetings.mp3
. . .
3_ Book 3 - Chapter 01 - The Departure of Boromir.mp3
. . .
4_ Book 4 - Chapter 01 - The Taming of Sméagol.mp3
. . .
1_ Book 5 - Chapter 01 - Minas Tirith.mp3
. . .
2_ Book 6 - Chapter 01 - The Tower of Cirith Ungol.mp3
. . .
```

You wish to give these files compact names, so that they will be easier to manage on your portable digital audio player:

```
LR_01_01.mp3
LR_01_02.mp3
. . .
LR_01_12.mp3
LR_02_01.mp3
. . .
LR_03_01.mp3
. . .
LR_04_01.mp3
. . .
LR_05_01.mp3
. . .
```

Using your fancy-schmancy file rename utility, the following command should do the trick:

```
python rename.py –t 3 –r "Book (\d) – Chapter (\d\d).*" "LR_0\1_\2.mp3" *.mp3
```