Software Engineering 265
Software Development Methods
Fall 2013

Assignment 3

Due: Monday, November 18th, 3:30 pm by submission via Subversion
(no late submissions accepted)

Programming environment

For this assignment **you must ensure your work executes correctly on the Linux machines in ELW B215**. You are free to do some of your programming on your own computers; if you do this, give yourself a few days before the due date to iron out any bugs in the Python program you have uploaded to the BSEng machines.

Individual work

This assignment is to be completed by each individual student (i.e., no group work). Naturally you will want to discuss aspects of the problem with fellow students, and such discussion is encouraged. **However, sharing of code fragments is strictly forbidden without the express written permission of the course instructor (Zastre).** If you are still unsure regarding what is permitted or have other questions about what constitutes appropriate collaboration, please contact me as soon as possible. (Code-similarity analysis tools will be used to examine submitted programs.)

Objectives of this assignment

- Continue to work with Python.
- Use the Python programming language to write schedprint_plus.py accepting slightly more complex data in ICS files.
- Use regular expressions as part of your solution.
- Use Subversion to manage changes in your source code and annotate the evolution of your solution with "messages" provided during commits.
- Test your code against the 30 provided test cases.

The "iCalendar" specification: adding more into schedprint_plus.py

The iCalendar standard has existed now for over 12 years (see RFC2445 at the Internet Engineering Task Force) and is used in many calendaring services including iCal (Apple) and Google Calendar. The standard was originally proposed by Microsoft and Lotus and they did it without reference to XML (hence its somewhat odd format).

Implementing the full standard is beyond the scope of SENG 265, yet we can add more interesting features than what we have had from the first two assignments.

- *Multiple ICS files in one run*: There is no reason why we cannot have multiple sets of events in several files combined to produce one schedule.

- *Time-zone neutral*: If we store the time and date of all events in Coordinated Universal Time (UTC) then we can obtain local time and dates by specifying the offset from UTC. For example, in British Columbia we are eight hours behind UTC (i.e., "-8").

To make things a little less confusing, do not worry about conversions between daylight savings time and standard time. Put differently, you can go ahead and use the Python library routines for adjusting dates and times by a set number of hours, and allow the library to handle any daylight savings / standard time boundaries.

Exercises for this assignment

1. Within your Subversion project create an "a3" subdirectory. New test files have been provided in a3.zip (supplied with the assignment description on conneX). Look at the README file for details on command-line parameters that are used to generate the tests. You must name your script "schedprint_plus.py". Any additional Python modules that you write for your assignment must be both contained in this directory and added to Subversion control.

2. Write your program. Amongst other tasks you will need to:
   - read text input from a file, line by line
   - write output to the terminal
   - extract substrings from lines produced when reading a file
   - use regular expressions
   - create and use lists in a non-trivial way

3. If you wish, you may write your solution using Python classes. However, you must use regular expressions at some point in your solution.

4. Some simpler test files have also been provided. Use these to start your implementation. Always start with simple. Refrain from writing the program all at once, and budget time to anticipate when things go wrong.

5. For this assignment you can assume all test inputs will be well-formed (i.e., our teaching assistant will not test your submission for handling of input or for arguments containing errors).

What you must submit

- A Python source file named “schedprint_plus.py” within your subversion repository containing a solution to Assignment #3.
- All additional Python files needed to run your solution.

Evaluation

As with the first assignment, students will demonstrate their work for assignment #3 to a member of the SENG 265 teaching team. Sign-up sheets for demos will be provided close to the due-date; each demo will require 10 minutes.

Our grading scheme is relatively simple.

- “A” grade: An exceptional submission demonstrating creativity and initiative. “schedprint_plus.py” runs without any problems.
- “B” grade: A submission completing the requirements of the assignment. “schedprint_plus.py” runs without any problems.
- “C” grade: A submission completing most of the requirements of the assignment. “schedprint_plus.py” runs with some problems.
- “D” grade: A serious attempt at completing requirements for the assignment. “schedprint_plus.py” runs with quite a few problems.
- “F” grade: Either no submission given, or submission represents very little work.

schedprint, version 3

The program will be a Python program. Its name must be "schedprint_plus.py" and it must be found in the "a3" directory of your SENG 265 Subversion project.

Input specification:

1. All input is from ASCII test files. The test file is indicated by a command-line argument (i.e., "--file=<filename>"). Multiple files may be provided by separating the filenames with commas (but no spaces), e.g., "--file=one.ics,many.ics".
2. The range of dates for which schedprint_plus.py will retrieve events are indicated by command-line arguments (i.e., "--start=dd/mm/yyyy --end=dd/mm/yyyy").
3. The offset from UTC is provided on the command line by using "--tz=<offset" (e.g., Vancouver would be "--tz=-8"; Berlin would be "--tz=1"). <u>An offset must be given.</u> This means that an event with a start datetime of "20131101T021500Z" with "--tz=-8" specified must result in a datetime of "20131031T181500".
4. Data lines for an "event" begin with a line "BEGIN:VEVENT" and end with a line "END:VEVENT".
5. Starting time: An event's starting date and time is contained on a line of the format "DTSTART:<icalendardate>" where the characters following the colon comprise the date/time in icalendar format.
6. Ending time: An event's ending date and time is contained on a line of the format "DTEND:<icalendardate>" where the characters following the colon comprise the date/time in icalendar format.
7. Event location: An event's location is contained on a line of the format "LOCATION:<string>" where the characters following the colon comprise the string describing the event location. These strings will never contain the ":" character. The location may be blank or missing from an event.
8. Event description: An event's description is contained on a line of the format "SUMMARY:<string>" where the characters following the colon comprise the string describing the event's nature. These strings will never contain the ":" character.
9. Repeat specification: If an event repeats, this will be indicated by a line of the format "RRULE:FREQ=<frequency>;UNTIL=<icaldate>". Note that these lines consist of combinations of colon (":") and semicolon (";") and equal signs ("=").
10. Ignore all other ICS fields.
11. Events within the input stream are not necessarily in chronological order.
12. Events may overlap in time.
13. Events may cross a day boundary.
14. All times are in Coordinated Universal Time (indicated by the "Z" added to the end of the datetime).
15. There is no limit to the number of characters in a summary.
16. There is no limit to the number of characters in a location.
17. There is no limit to the length of an .ics file (other than those imposed by underlying OS and machine).

18. There is no limit to the number of events indicated by an .ics file (other than those imposed by the underlying OS and machine).

Output specification:

1. All output is to stdout.
2. All events which occur on the days indicated on or between --start and --end must appear in chronological order.
3. If two events start at the same day and time, their order will be lexicographic-ascending with respect to the event end datetime, then location, then summary.
4. If events occur on a particular date, then that date must be printed only once in the following format:

   ```
   <month text> <day>, <year> (<short weekday>)
   --------------------------------------------
   ```

   Note that the line of dashes below the date must match the length of the date.
5. Days are separated by a single blank line.
6. Starting and ending times are in 12-hour format with “am” and “pm” as appropriate. For example, five minutes after midnight is represented as “12:05 am”.
7. A straight line (“|”) is used to separate the start/end times from the event description.
8. The event LOCATION and SUMMARY text appears on the same line as the event time. (This text may include parentheses.)
9. If the start and end times of the event are on different days, a “+” character will immediately follow the end time.
10. The event LOCATION text appears before the SUMMARY text and each are separated with hyphens (“---“). However, if the LOCATION is either blank or missing from the event, then only the SUMMARY is printed.

Events from the same day are printed on successive lines in chronological order. Do not use blank lines to separate the event lines within the same day.

In the case of tests provided by the instructor, the Unix “diff” utility will be used to compare your program’s output with what is expected for that test. Significant differences reported by “diff” may result in grade reductions.