Software Engineering 265
Software Development Methods
Fall 2013

Assignment 2

Due: ~~Monday, October 28th~~ Wednesday, October 30th, 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 welcome to work on your own laptops and desktops; if you do this, give yourself a few days before the due date to iron out any bugs in the C program you have uploaded to the BSEng machines. (Bugs in this kind of programming tend to be platform specific, and something that works perfectly at home may end up crashing on a different hardware configuration.)

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

- Learn to use basic features of the Python languages
- Use the Python programming language to write a less resource-restricted implementation of "schedprint" (but without using regular expressions).
- 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 twenty provided test cases.

Schedprint: Returning to the problem

For this assignment you are to re-use the description of the problem as given in assignment #1 along with test files provided for that assignment plus some new test files. However, some of the limits that were placed on certain values are no longer needed (e.g., maximum number of events need no longer be limited to 1000). This is because Python's ability to expand lists dynamically, along with additional features that result from dynamic typing, eliminates the requirement to state such limits in code.

The arguments used for the Python script will be slightly different. You will indicate the range of dates to be used to generate the schedule by providing "--start" and "--end" values:

```
./schedprint.py --start=01/10/2013 --end=31/10/2013 --file=one.ics
```

However, I will place three different kinds of constraints on your program.

1. For this assignment **you are not to use regular expressions**. (We will instead use these in assignment 3 in order to write a more powerful version of the program that processes more complex .ics files.)
2. You must **not** use global variables.
3. You must **make good use of functional decomposition**. Phrased another way, your submitted work must not contain one or two giant functions where all the logic is concentrated.

Exercises for this assignment

1. Within your Subversion project create an "a2" subdirectory. Use the test files in a2.zip (i.e., the ZIP file located in conneX at the assignment #2 description). Your "schedprint.py" script must be located in this 'a2" director. Ensure the subdirectory and script file are 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
   - create and use lists in a non-trivial array.

3. Keep all of your code in one file for this assignment. In later assignments we will use the multiple-module features of Python.

4. Use the test files and listed test cases to guide your implementation effort. Start with simple cases (such as those given in this writeup). 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). Assignments 3 and 4 will specify error-handling as part of the assignment.

What you must submit

- A single Python source file named "schedprint.py" within your subversion repository containing a solution to assignment #2.
- No regular-expression routines are to be used for assignment #2.

Evaluation

As with the first assignment, students will demonstrate their work for assignment #2 to a member of the SENG 265 teaching team. Sign-up sheets for demos will be provided a few days before the due-date; each demo will require from 10 to 15 minutes.

Our grading scheme is relatively simple.

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

The program will be a Python program. Its name must be "schedprint.py" and it must be found in the "a2" 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>"). (Refer to slide 65 of the Python lectures on how command-line arguments are parsed in Python.)
2. The range of dates for which schedprint will retrieve events are indicated by a command-line argument (i.e., "--start=dd/mm/yyyy --end=dd/mm/yyyy"). (Refer to the Python lectures for details on how command-line arguments are parsed in Python.)
3. Data lines for an "event" begin with a line "BEGIN:VEVENT" and end with a line "END:VEVENT".
4. 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.
5. 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.
6. 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.
7. 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.
8. Repeat specification: If an event repeats, this will be indicated by a line of the format "RRULE:FREQ=<frequency>;UNTIL=<icaldate>". The only frequencies you must account for are weekly frequencies. The date indicated by UNTIL is the last date on which the event will occur (i.e., is inclusive). Note that this line contains a colon (":") and semicolon (";") and equal signs ("=").
9. Events within the input stream are not necessarily in chronological order.
10. No two events will ever overlap in time.
11. No event will ever cross a day boundary.
12. All times are local time (i.e., no timezones will appear in a date/time string).
13. There is no limit to the number of characters in a summary.
14. There is no limit to the number of characters in a location.
15. There is no limit to the length of an .ics file (other than those imposed by underlying OS and machine).
16. 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 events occur on a particular date, then that date must be printed only once in the following format:

   <month text> <day>, <year> (<day of week>)

   Note that the line of dashes below the date must match the length of the date.
4. Days are separated by a single blank line.
5. Starting and ending times given in 12-hour format with “am” and “pm” as appropriate. For example, five minutes after midnight is represented as “12:05 am”.
6. A colon is used to separate the start/end times from the event description
7. The event SUMMARY text appears on the same line as the event time. (This text may include parentheses.)
8. The event LOCATION text appears on after the SUMMARY text and is surrounded by parentheses.

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.