Software Engineering 265
Software Development Methods
Summer 2019

Assignment 2

Due: Thursday, July 4, 11:55 pm by submission via git
(no late submissions accepted)

Programming environment

For this assignment please ensure your work executes correctly on the Linux machines in ELW B238. You are welcome to use your own laptops and desktops for much of your programming; if you do this, give yourself a few days before the due date to iron out any bugs in the Python 3 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.)

When evaluating your submission, the teaching team will use the environment established by the `setSENG265` command that is available on the lab workstations.

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 language.
- Use the Python 3 programming language to write a less resource-restricted implementation of `calprint` (**but without using regular expressions or user-defined classes**).
- Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits.
- Test your code against the 15 provided test cases from assignment #1.

`calprint2.py`: Returning to the problem

For this assignment please use the description of the problem as provided at the end of this document, and also use the assignment #1 test files. Some of the limits that were placed on certain values are no longer needed (e.g., maximum number of events, maximum line length, etc.).

However, 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`" arguments. Note also that the script is named "`calprint2.py`" (and **not** "`calprint.py`").

```
./calprint2.py --start=18/6/2019 --end=18/6/2019 --file=one.ics
```

As with the first assignment, all output is to `stdout`. You must test the output of your program in the same manner as with assignment #1 (i.e., using `diff`).

However, I will place four 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 write your own classes** as this will be work for assignment #3.
3. You must **not use global variables**.
4. You must **make good use of functional decomposition**. Phrased another way, your submitted work **must not** contain one or two giant functions where all of your program logic is concentrated.

Exercises for this assignment

1. Within your Git repo ensure there is an "`a2`" subdirectory. (For testing please use the files provided for assignment #1.)Your "`calprint2.py`" script must be located in this "`a2`" directory. Note that a starter `calprint2.py` file is available for you in the `/home/zastre/seng265/a2` directory.

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 assignment #3 we will use the multiple-module and class features of Python. Please ensure you also respect all of the other constraints described earlier in this document.

4. Use the test files and listed test cases to guide your implementation effort. 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 may specify error-handling as part of the assignment.

What you must submit

- A single Python source file named "`calprint2.py`" within your Git repository containing a solution to assignment #2.
- No regular-expressions, global variables, or user-defined classes are to be used for assignment #2.

Evaluation

Our grading scheme is relatively simple.

- "A" grade: A submission completing the requirements of the assignment which is well-structured and very clearly written. All tests pass and therefore no extraneous output is produced.
- "B" grade: A submission completing the requirements of the assignment. `calprint2.py` runs without any problems; that is, all tests pass and therefore no extraneous output is produced.
- "C" grade: A submission completing most of the requirements of the assignment. `calprint2.py` runs with some problems.
- "D" grade: A serious attempt at completing requirements for the assignment. `calprint2.py` runs with quite a few problems; some non-trivial tests pass.
- "F" grade: Either no submission given, or submission represents very little work, or no tests pass.

The program will be a Python program. Its name must be “calprint2.py” and it must be found in the “a2” sub-directory of your SENG 265 git project.

Input specification:

1. All input is from ASCII test files.
2. Data lines for an “event” begin with a line “BEGIN:VEVENT” and end with a line “END:VEVENT”.
3. 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.
4. 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.
5. 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.
6. 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.
7. Repeat specification: If an event repeats, this will be indicated by a line of the format “RRULE:FREQ=<frequency>;UNTIL=<icalendardate>”. 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 (“=”).
8. Events within the input stream are not necessarily in chronological order.
9. Events may overlap in time.
10. No event will ever cross a day boundary.
11. All times are local time (i.e., no timezones will appear in a date/time string).

Output specification:

1. All output is to stdout.
2. All events which occur from 12:00 am on the --start date and to 11:59 pm on the --end date must appear in chronological order based on the event's starting time that day.
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. You may use Python's datetime module in order to create the calendar-date line.
4. Days are separated by a single blank line. There is no blank line at the start or at the end of the program's output.
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 even time. (This text may include parentheses.)
8. The event LOCATION text appears on after the SUMMARY text and is surrounded by square brackets.

Events from the same day are printed on successive lines in chronological order by starting time. 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.