Assignment 2

Due date: Monday, March 14, 11:59 pm
Submission via Git only

Programming environment

For this assignment you must ensure your work executes correctly on the virtual machines (i.e., Senjhalla or its analogous environment for students with M1 Mackbooks) you installed/configured as part of Assignment 0 as this is our "reference platform". This same environment will be used by the teaching team when grading the work submitted by the SENG 265 students.

All test files and sample code for this assignment are available on the SENG server in /seng265work/2022-spring/a2/ and you must use the scp command to get a copy of the files. For example:

```
scp NETLINKID@seng265.seng.uvic.ca:/seng265work/2022-spring/a2/* .
```

Any programming done outside of Senjhalla might result in lost marks or even 0 marks for the assignment. Make sure that your submitted file is named "process_cal2.py" and is inside your a2 folder of your cloned Git repository.

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. Code-similarity analysis tools are used to check submitted work for plagiarism.

Learning objectives

I. Learn or review basic features of the Python programming language.
II. Use the Python 3 programming language to write a less resource-restricted and slightly different implementation of process_cal — but without using user-defined classes.
III. Use Git to manage changes in your source code and annotate the evolution of your solution with messages provided during commits. Remember, the only acceptable way of submitting your work is using Git.
IV. Test your code against the provided test cases.

process_cal2.py: Returning to the problem

a) For this assignment, please use the problem description as provided at the end of this document and use the test files provided to test your program. Some of the limits of Assignment 1 that were placed on certain values are no longer needed (e.g., maximum number of events, maximum line length, etc.).
b) The arguments used for the Python script are similar to the ones used for Assignment 1. That is, you will indicate the range of dates to be used to generate the schedule by providing "--start" and "--end" arguments. However, the executable will now be named "process_cal2.py" (and not "process_cal.py") and you will need to provide the names (i.e., paths) of two additional files (i.e., circuits.xml and broadcasters.xml)
c) As opposed to the first assignment, the output of your program won't be stdout to avoid previous issues with the Unix diff command. Your program must generate a file called **output.yaml** that is compliant with the YAML standard.

d) Although the Unix diff command might still be used, the most reliable way to test your program is to use the provided **tester.py** file which will validate the output produced by your program in **output.yaml**, given a particular test (i.e., a set of arguments).

Restrictions

- Your program must be decomposed into easy-to-understand components. Good program decomposition is required. Methods or functions must be short and effectively parameterized.
- Unwieldy functions are not accepted.
  Do not use global variables.
- Modules (libraries) for parsing XML or YAML files **CAN NOT** be used. That is, you are not allowed to import those modules into your program (e.g., yaml or xml modules). You can use standard Python collections to parse .xml and .yaml files.

Testing your solution

- Refer to the example commands in the file "TESTS.md" for appropriate command line input. Your solution must accommodate all specified command line inputs.
- Make sure to use the test files provided as part of this assignment.
- Keep all your code in one file (process_cal2.py) for this assignment. In Assignment 4 we will use the multi-module features of Python.
- Use the test files and listed test cases to guide your implementation effort. Develop your program incrementally.
- For this assignment you can assume all test inputs are well-formed.
- Do not rely on visual inspection. Human eyes are poor at comparing white space. YAML is an indentation-based standard, thus make sure that your outputs follow the specification described at the end of this document. You can use the provided tester file (i.e., "tester.py") so you can verify the validity of your outputs in a simpler manner.

What to submit

A single Python source file named "process_cal2.py" submitted (i.e., Git push to your remote repository) to the a2 folder in your repository.

Grading

Assignment 2 grading scheme is as follows. In general, straying from the assignment requirements will result in zero marks due to automated grading.

A grade: A submission completing the requirements of the assignment and is well-structured and clearly written. Global variables are not used. process_cal runs without any problems; that is, all tests pass and therefore no extraneous output is produced. An A-grade submission is one that passes all the tests and does not have any quality issues. Outstanding solutions get an A+ (90-100 marks), solutions that are not considered outstanding by the evaluator will get an A (85-89 marks) and a solution with minor issues will be given an A- (80-84 marks).

B grade: A submission completing the requirements of the assignment. process_cal runs without any problems; that is, all tests pass and therefore no extraneous output is produced. The program is clearly written. Although all the tests pass, the solution includes significant quality issues. Depending on the number of qualitative issues, the marker may give a B+ (77-79 marks), B (73-76 marks) or a B- (70-72 marks) grade.

A submission with any one of the following cannot get a grade higher than B:

- Submission runs with warnings

- Submission has 1 or 2 large functions
- Program or file-scope variables are used

A submission with more than one of the following cannot be given a grade of higher than B-:

- Submission runs with warnings
- Submission has 1 or 2 large functions.
- Program or file-scope variables
- No documentation is provided

C grade: A submission completing most of the requirements of the assignment. process_cal runs with some problems. This is a submission that presents a proper effort but fails some tests. Depending on the number of tests passed, which tests pass and a qualitative assessment, a grade of C (60-64 marks) or C+ (65-69 marks) is given.

D grade: A serious attempt at completing requirements for the assignment (50-59 marks). process_cal runs with quite a few problems. This is a submission that passes only a few of the trivial tests.

F grade: Either no submission given, or submission represents little work or none of the tests pass (0-49 marks). No submission, 0 marks. Submissions that do not run, 0 marks. Submissions that fail all tests and show a poor to no effort (as assessed by the marker) are given 0 marks. Submissions that fail all tests, but represent a sincere effort (as assessed by the marker) may be given a few marks

Additional Criteria for Qualitative Assessment

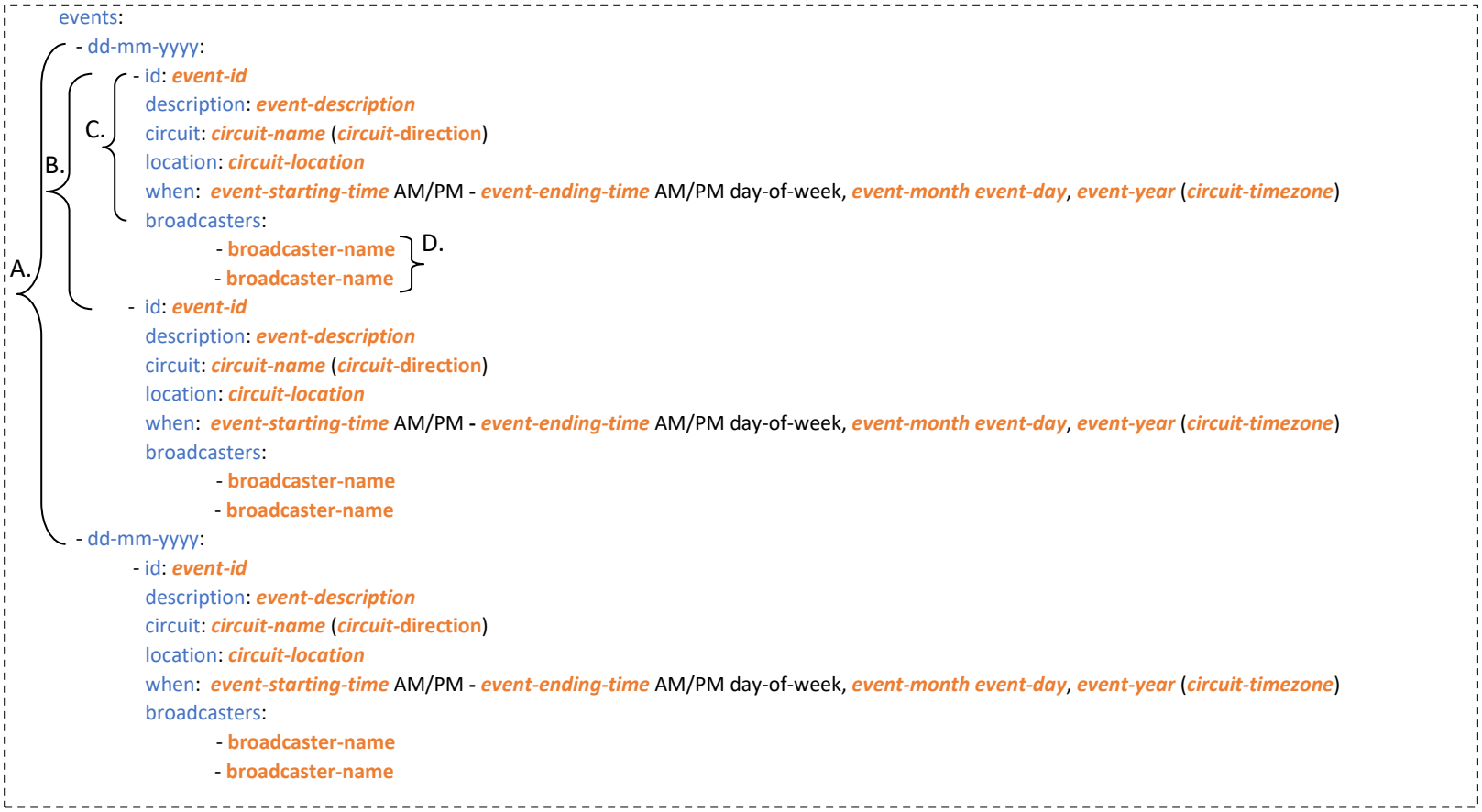
- **Documentation and commenting:** the purpose of documentation and commenting is to write information so that anyone other than yourself (with knowledge of coding) can review your program and quickly understand how it works. In terms of marking, documentation is not a large mark, but it will be part of the quality assessment.
- **Functional decomposition:** quality coding requires the good use of functions. Code that relies on few large functions to accomplish its goals is considered poor-quality code. Typically, a good program has a main function that does some basic tasks and calls other functions, that do most of the work. A solution that passes all tests, but contains all code in one or two large functions will not be given a grade better than a B.
- You must not use program-scope or file-scope variables.
- **Proper naming conventions:** You must use proper names for functions and variables. Using random or single character variables is considered improper coding and significantly reduces code readability. Single character variables as loop variables is fine.
- **Debugging/Comment artifacts:** You must submit a clean file with no residual commented lines of code or unintended text.
- **Quality of solution:** marker will access the submission for logical and functional quality of the solution. Some examples that would result in a reduction of marks: solutions that read the input files several times, solutions that represent the data in inappropriate data structures, solutions which scale unreasonably with the size of the input.

Input specification

- All input test files are in ASCII-format
- ***2022-season-testing.xml & 2022-f1-races-americas.xml***
  - Data lines for an "event" are contained within the block defined by the tags <event></event>
  - ***event-id:*** An event's id is contained on a line including the tag <id>
  - ***event-description:*** An event's description is contained on a line including the tag <description>
  - ***event-starting-time:*** An event's starting date and time is contained on a line including the tag <start>
  - ***event-ending-time:*** An event's ending date and time is contained on a including the tag <end>
  - ***event-location:*** An event's location is contained on a line including the tag <location>. This time, the information inside this tag refers to an ID linked to a particular circuit described in **circuits.xml (c.f., *circuit-id*)**
  - ***event-id:*** An event's broadcaster is contained on a line including the tag <broadcaster>. The information inside this tag refers to an ID linked to a particular circuit described in **broadcasters.xml (c.f., *broadcaster-id*)**
  - ***event-date:*** An event's date information is contained on the lines including the following tags:
    - ***event-day:*** <day> (numeric),
    - ***event-month:*** <month> (numeric)
    - ***event-year:*** <year> (numeric)
  - Events within the input stream are not necessarily in chronological order
  - Events may overlap in time
  - No event will ever cross a day boundary
  - The order of the tags might vary from event to event
- ***circuits.xml***
  - Data lines for a "circuit" are contained within the block defined by the tags <circuit></circuit>
  - ***circuit-id***: An identification code for the circuit is contained on a line including the tag <id>
  - ***circuit-name***: The official name of the circuit is contained on a line including the tag <name>
  - ***circuit-location***: The city that hosts the circuit is contained on a line including the tag <location>
  - ***circuit-timezone***: The time zone of the city that hosts the circuit is contained on a line including the tag <timezone>
  - ***circuit*-direction**: The direction of the circuit (i.e., clockwise or anti-clockwise) is contained on a line including the tag <direction>
- ***broadcasters.xml***
  - Data lines for a "broadcaster" are contained within the block defined by the tags <broadcaster></broadcaster>
  - ***broadcaster-id*:** An identification code for the broadcaster is contained on a line including the tag <id>
  - ***broadcaster-name*:** The official name of the broadcaster is contained on a line including the tag <name>
  - ***broadcaster-cost*:** The yearly cost of a broadcaster is contained on a line including the tag <cost>

Output specification

- After its execution, your program must produce\create an output file called **output.yaml** compliant with the YAML standard
- YAML is a spacing-sensitive standard, thus make sure your file contains the appropriate indentation.
- The **output.yaml** file to be produced by your program must follow the sample format described below (examples of the required **output.yaml** files for each test are provided in the test files):
  A. List of individual, unique dates
  B. List of events happening in a particular date
  C. List of event attributes
  D. List of broadcasters of an event


events:
- dd-mm-yyyy:
- id: event-id
description: event-description
circuit: circuit-name (circuit-direction)
location: circuit-location
when: event-starting-time AM/PM - event-ending-time AM/PM day-of-week, event-month event-day, event-year (circuit-timezone)
broadcasters:
- broadcaster-name
- broadcaster-name
- id: event-id
description: event-description
circuit: circuit-name (circuit-direction)
location: circuit-location
when: event-starting-time AM/PM - event-ending-time AM/PM day-of-week, event-month event-day, event-year (circuit-timezone)
broadcasters:
- broadcaster-name
- broadcaster-name
- dd-mm-yyyy:
- id: event-id
description: event-description
circuit: circuit-name (circuit-direction)
location: circuit-location
when: event-starting-time AM/PM - event-ending-time AM/PM day-of-week, event-month event-day, event-year (circuit-timezone)
broadcasters:
- broadcaster-name
- broadcaster-name
A.
B.
C.
D.