Restrictions

The following are not permitted to be used when completing this assignment:

- `for` keyword
- `in` keyword
- `lambda` keyword
- `any()` function
- `all()` function
- `filter()` function
- `map()` function
- `eval()` function
- `global` keyword
- `import` keyword. Allowed import statements:
    - `import math`
    - `import datetime`
    - `import sys`
    - any module you have written, e.g. `import moderator`

A submission containing a file longer than 3000 lines or larger than 500kB will not be accepted. To check the file sizes of your files, use the bash command `ls -s --block-size=KB`

Assignment Description

Chat Forum Moderator Program

Your task is to create a program called moderator.py that moderates a chat forum and a program to test moderator.py. The moderator program is divided into parts involving command line arguments, reading files, writing to files, replacing text, and writing classes.

Validation rules:

Throughout this assignment, the program will need to validate a variety of inputs. Below are some rules for validating commonly found entities:

Names:

A name is permitted to contain the following:

- English characters A-Z, upper or lower case
- Space character
- 'Minus'/'dash' symbol: `-`

A name is invalid if it:

- Contains a character not in the above list
- Consists only of space characters
- Has no characters (is empty)

Datetime strings:

Whenever a datetime is represented as a string, it will be in the following format:

`YYYY-MM-DDTHH:NN:SS`

where `YYYY` is the year, `MM` is the month, `DD` is the day, `T` is the literal character 'T', `HH` is the hour, `NN` is the minute, and `SS` is the second. The length of the format is fixed for all values, such that if the month is January, it will be represented as `01`. `YYYY`, `MM`, `DD`, `HH`, `NN`, and `SS` must be digits.
You may assume that if the above format is matched, the datetime string represents a valid datetime.

An example: `1996-09-12T16:30:16`

Anything that deviates from this format is considered an invalid datetime string.

Personality scores:

A personality score must be a whole number between -10 and 10 (inclusive).

File headers:

Files for forums, people, and words all start with a file header. A file header is structured as follows:

- The first line of the file is the forum title. The forum title can contain any text (including whitespace) but may not be empty.
- The second line of the file is a blank line.

Hint: you should write a function for each of the above validations

Part 1: Testing

This part requires content taught in week 8

You must first write black box tests aimed at testing functions relevant to the Moderator program. The test suite file is called test.py and can be found in the 'Test Suite' slide.

There are two functions which must be tested: `is_valid_name()` and `is_chronological()`. The test suite program will be run against various implementations of the functions: one correct implementation and many incorrect implementations. This is to ensure your test suite can catch a wide range of bugs. You are not required to implement the functions, just to write tests.

The test suite program must print two lines to the terminal in the following format: `<function_name> has <pass_status>`. If the function being tested failed one or more of your tests, the test suite should print `<function_name> has failed.` **once only**. If the function being tested passes all tests, the test suite should print `<function_name> has passed.` **once only**.

The correct implementation must be identified by your test suite in order to receive marks for identifying the incorrect implementations. The marking script first calls your test suite on the correct version and then an incorrect version.

Example usages of test.py:

```
$ python test.py
is_valid_name has passed.
is_chronological has failed.
$
```

```
$ python test.py
is_valid_name has failed.
is_chronological has passed.
$
```

Function specifications

Function specifications

Ensure you are testing the functions, not implementing them

`is_valid_name(name: str) -> bool`

`is_valid_name()` accepts one positional argument `name` and returns a boolean. The function returns `True` if the name is valid and `False` if the name is invalid - validation rules can be found under the 'Validation Rules' section. The function returns `False` if `name` is not a string.

`is_chronological(earlier_dt: str, later_dt: str) -> bool`

`is_chronological()` accepts two positional arguments `earlier_dt` and `later_dt` and returns a boolean. The function returns `True` if `later_dt` represents a datetime that is chronologically later then `earlier_dt`, and `False` otherwise. The function returns `False` if either argument is not a string, or if the strings provided are not in the correct datetime string format.

Consider every kind and combination of arguments the functions may be given. For each kind of input, determine what the function should return and write a test to ensure that it does.

Part 2: Command line arguments

This part requires content taught in week 9

The Moderator program requires the following command line arguments:

- `-task <task>` where `<task>` is one of the following: `rank_people`, `validate_forum`, `censor_forum`, `evaluate_forum`
- `-log <filename>` where `<filename>` is the name of the file where the log should be written
- `-forum <filename>` where `<filename>` is the name of the file containing the forum messages
- `-words <filename>` where `<filename>` is the name of the file containing banned words
- `-people <filename>` where `<filename>` is the name of the file containing forum users' names and personality scores

All arguments must be provided. If an argument is not provided, the program should print `No <arg> arguments provided.` and stop immediately, where `<arg>` (without the `-`) is the first argument in the list above that is missing.
If the `<task>` argument does not match a value in the options list, the program should print `Task argument is invalid.` and stop immediately.
The arguments can be provided in any order.

If any of the `<filename>` arguments for `-forum`, `-words`, or `-people` refers to a file that does not exist (or the program cannot read for any reason) the program should print `<filename> cannot be read.` for the first `<filename>` that is not readable (in the above order) and then stop immediately. As an example: if provided `-forum bad.forum` and `-words bad.words` where both files are unreadable, the program should print `bad.forum cannot be read.`

If multiple errors in the program call are incorrect, the first error that applies from above should be printed.
If the program call is correct, the program should print `Moderator program starting...`

Examples

Examples of correct program calls:

```
$ python3 moderator.py -task censor_forum -log example.log -forum example.forum -words example.words -people
Moderator program starting...
$
```

```
$ python3 moderator.py -log example.log -people example.people -task rank_people -words example.words -forum
Moderator program starting...
$
```

Examples of incorrect program calls and the error messages displayed:

```
$ python3 moderator.py -task censor_forum -log example.log
No forum arguments provided.
$
```

```
$ python3 moderator.py -task censor_forum -log example.log -forum example.forum
No words arguments provided.
$
```

Example of a forum file that is not readable passed as an argument:

```
$ python3 moderator.py -task censor_forum -log example.log -forum bad.forum -words bad.words -people bad.peop
bad.forum cannot be read.
$
```

Part 3: Personality scores

This part requires content taught in week 9

The Moderator program must validate and interpret a people file listing users of a forum. It should extract personality scores and then rank users according to their score, writing this ranking back to the file in descending order.

An example use of the program for this part is as follows:

```
$ python3 moderator.py -task rank_people -log example.log -forum example.forum -words example.words -people .
```

In the above example, the people file is specified as `./example/example.people`

The file will contain a comma separated list of names and personality scores. The format of the file is as follows:

- The file header (format specified in the 'Validation rules' section)
- All subsequent lines are people entries.

People entries:

Each people entry will be structured like so: `<name>,<score>\n`
The `<name>` and `<score>` validity is detailed in the 'Validation rules' section under 'Names' and 'Personality score' respectively.

The program must first check that each line of the file is of the correct format.
If the file header is not in the correct format, the program should write the following to the log file: `Error: people file read. The people file header is incorrectly formatted`
If a people entry is not structured in the format `<name>,<score>\n`, the program should write the following to the log file: `Error: people file read. The people entry is invalid on line <x>` where `<x>` is the line number (the first line of the file is line 1).
If a name is invalid, the program should write the following to the log file: `Error: people file read. The user's name is invalid on line <x>` where `<x>` is the line number.
If a personality score is invalid, the program should write the following to the log file: `Error: people file read. The personality score is invalid on line <x>` where `<x>` is the line number.

If an error in formatting is encountered, the program should write to the log as specified above, and then exit immediately - the people file should **not** be modified. If multiple errors are present in the file, the error on the earliest line should be logged, i.e. an error on line 3 is reported if both line 3 and 4 contain errors.
If the file format is valid, nothing should be written to the log but the log file should still be created (yet empty).

Once the file has been validated, the program must read the personality scores, rank the people and then rewrite the file (with the same name) in descending order of personality score rank. If two people share the same personality score, their order from the original file should be kept.

Examples

An example use of the program for this part is as follows:

```
$ python3 moderator.py -task rank_people -log example.log -forum example.forum -words example.words -people e
```

A provided people file for a forum with three users:

```
Torchlight Forum

Plato,5
Socrates,10
Glaucon,-1
```

The same people file after the program has been run with `rank_people` task:

```
Torchlight Forum

Socrates,10
Plato,5
Glaucon,-1
```

Part 4: Reading the Forum

This part requires content taught in week 9

The Moderator program must validate a file containing the posts in the forum. The format of the file is as follows:

- The file header (format specified in the 'Validation rules' section)
- All subsequent lines are a part of a post.

Posts/replies

Posts consist of three lines:

- The first line of a post is a datetime string
- The second line of a post is the name of the user who wrote the post
- The third line of a post is the content/message of the post
- Every post ends with a newline character `\n`

A reply is a post that is in response to another post. In terms of format, replies are the same as posts but each line of a reply begins with a tab character `\t`. A reply must come after a post or it is not valid.

Validating the file

If the forum file is valid, the program should write nothing to the log file but the log file should still be created (yet empty).
If the forum file is invalid, the program should write the relevant violation message to the log file as detailed below. The first violation (reading from the start to the end of the file) should be logged and then the program should immediately stop.

To be considered valid the following must be true about the forum file:

- The header (first and second lines) must be correct
  - Violation message: `Error: forum file read. The forum file header is incorrectly formatted`
- The format of every post and reply must be correct
  - Violation message if any reply is placed before a post: `Error: forum file read. The reply is placed before a post on line <x>` where `<x>` is the line number (the first line of the file is line 1).
  - Violation message if the format of a post is invalid: `Error: forum file read. The post has an invalid format on line <x>` where `<x>` is the line number.
- The format of every user name must be correct
  - Violation message: `Error: forum file read. The user's name is invalid on line <x>` where `<x>` is the line number.
- The format of each datetime string must be correct
  - Violation message: `Error: forum file read. The datetime string is invalid on line <x>` where `<x>` is the line number.
- Datetime strings must be in forward chronological order:
  - A reply to a post must have a datetime later than the post it responds to **and** previous replies. Violation message: `Error: forum file read. The reply is out of chronological order on line <x>` where `<x>` is the line number.
  - A post must have a datetime later than previous posts, but replies to a previous post may have later datetimes than subsequent posts. Violation message: `Error: forum file read. The post is out of chronological order on line <x>` where `<x>` is the line number.
- The message of a post can contain any text (including no text).

Examples

An example use of the program for this part is as follows:

```
$ python3 moderator.py -task validate_forum -log example.log -forum example.forum -words example.words -peopl
```

Below are two examples of correctly formatted files:

```
Example Forum

1997-07-06T15:02:50
A
A post
    1997-07-06T15:03:32
    B
    A reply
    1997-07-06T15:03:36
    C
    Another reply
1997-07-06T15:03:20
D
A later post
    1997-07-06T15:03:54
    E
    Another reply
```

```
Torchlight Forum

1996-09-12T16:30:16
Plato
Are things real?
	1996-09-12T16:30:54
	Socrates
	Now let me show in a figure how far our nature is enlightened or unenlightened: Behold! human beings livi
	1996-09-12T16:31:12
	Glaucon
	I see.
```

The indentations in the file will be tab characters `\t`, not space characters

Part 5: Censoring the Forum

This part requires content taught in week 9

The Moderator program must read the provided words file, validate its format, and censor instances of the words in the posts/replies in the forum file.

Validating the banned words file

The words file is structured as follows:

- The file header (format specified in the 'Validation rules' section)
- All subsequent lines are banned words
    - A banned word can be any text, except for a blank line or whitespace-only text. A banned word can contain whitespace but it cannot be purely whitespace
    - Every banned word line ends with a newline character `\n`

If the words file is valid, the program should write nothing to the log file but the log file should still be created (yet empty).
If the words file is invalid, the program should write the relevant violation message to the log file as detailed below. The first violation (reading from the start to the end of the file) should be logged and then the program should immediately stop.

To be considered valid the following must be true about the words file:

- The header must be correct
    - Violation message: `Error: words file read. The words file header is incorrectly formatted`
- Every banned word line must be correct
    - Violation message: `Error: words file read. The banned word is invalid on line <x>` where `<x>` is the line number.

If the words file has passed validation, the program should validate the forum file in the same way as in Part 4.

Censoring the forum

Once all validation is complete, the program should begin the censorship functionality.

The program should rewrite the provided forum file and replace every instance of banned words with asterisks `*` of the same length. Only the message of posts should be checked for banned words, not the names of users, the file header, nor datetime strings.
An instance of a banned word is the use of a banned word surrounded by any of the following: space character, newline character `\n`, tab character `\t`, punctuation characters `,`, `.`, `'`, `"`, `!`, `?`, `(`, `)`.
A banned word is case insensitive: all case combinations of the banned word should be replaced.

If the word "hand" is banned, then the following contain instances of the banned word (the associated replacement text is also shown):

- The (hand)
    - The (****).
- ,hAnD.
    - ,****.
- This hand?
    - This ****?

The following are **not** instances of the banned word:

- The handler
- hand-over
- hand/palm

Examples

An example use of the program for this part is as follows:

```
$ python3 moderator.py -task censor_forum -log example.log -forum example.forum -words example.words -people
```

Valid words file:

```
Torchlight Forum

things
nature
show
i see
```

Valid forum file:

```
Torchlight Forum

1996-09-12T16:30:16
Plato
Are things real?
    1996-09-12T16:30:54
    Socrates
    Now let me show in a figure how far our nature is enlightened or unenlightened: Behold! human beings livi
    1996-09-12T16:31:12
    Glaucon
    I see.
```

Censored forum file:

```
Torchlight Forum

1996-09-12T16:30:16
Plato
Are ****** real?
    1996-09-12T16:30:54
    Socrates
    Now let me **** in a figure how far our ****** is enlightened or unenlightened: Behold! human beings livi
    1996-09-12T16:31:12
    Glaucon
    *****.
```

Part 6: Advanced Personalities

This part requires content taught in week 10

The User class

The Moderator program must include a class representing users in the forum, called `User`. This class will be used to track various qualities of the forum users in order to modify their personality scores. The User class can contain any attributes and methods of your choosing, but **must** feature the following instance attributes/methods:

Attributes:

- `name: str` - given in the constructor and cannot change
- `engagement: float` - has a value of 0 at object construction
- `expressiveness: float` - has a value of 0 at object construction
- `offensiveness: float` - has a value of 0 at object construction

Methods:

`__init__(self: User, name: str)`

- `__init__()` constructs the `User` object. This method should set the value of the instance attribute `name` to the value of the parameter `name`. No validation is required inside this method. This method should also set the value of the instance attributes `engagement`, `expressiveness`, and `offensiveness` to 0.

`process_message(self: User, message: str, banned_words: list) -> bool`

- `process_message()` accepts a string `message` and a list `banned_words`. The method modifies the instance attributes of the user object according to the rules outlined below in the 'Personality score modification' subsection. If the `message` argument is not a string the method should return `False`, otherwise it should return `True`.

`calculate_personality_score(self: User) -> int`

- `calculate_personality_score()` uses the current values of `engagement`, `expressiveness`, and `offensiveness` to calculate and return a personality score for the user object. The value returned by the method must be an integer, and should round the calculated score towards zero (Hint: the `int()` cast function uses this rounding method).
  Note: the method **is permitted** to return a value smaller than -10 or larger than 10

  The rules for calculating the score are as follows:
  - score = `expressiveness` - `offensiveness`
  - The return value of the method cannot be greater than `engagement`. For example, if `engagement` is 3, `expressiveness` is 5, and `offensiveness` is 1, the score would first come to 5 - 1 = 4, but the return value would be capped at 3.

Personality score modification

The program must first validate the provided forum file the same way as outlined in Part 4. If the forum file is valid, the program must then validate the provided words file the same way as outlined in Part 5. If the words file is valid, the program must then validate the provided people file as outlined in Part 3. If all files are valid, the program can continue.
The program must read through the forum file and modify the attributes of each user based on the message of their posts (not the name or datetime string). The rules for attribute modification are as follows:

- `engagement` increases by 1 for each message
- `expressiveness` increases by 1 if a message contains a `!` character and increases by 2 if a message contains both a `!` character and a `?` character. If the message does not contain either character, the value decreases by 1.
- `offensiveness` increases by 1 for every message that contains one or more banned words, as outlined in the provided words file (**Note:** the `process_message()` method should not access the words file directly). An instance of a banned word is classified in the same way as outlined in Part 5.

All of the above rules have a 1.5 times multiplier if the message is contained in a post that is not a reply, i.e. `engagement` increases by 1 for a reply and 1.5 for a non-reply post.

Once the entire forum file has been read and every post message has been processed, the program must modify the personality scores in the provided people file. The people file must be rewritten using the new personality scores and the user entries must be reordered so that users are written in descending order by personality score, in the same way as outlined in Part 3. The personality scores must be modified according to the following rules:

- The new personality score is the old personality score added to the calculated personality score for the forum.
- The new personality score must be a whole number between -10 and 10, inclusive. If the new personality score is outside this range, the score is set to the closest value within the range, i.e. if the old personality score is 0 and the calculated personality score for the forum is 15, the score recorded in the person file will be 10.

Examples:

An example use of the program for this part is as follows:

```
$ python3 moderator.py -task evaluate_forum -log example.log -forum example.forum -words example.words -peopl
```

words file:

```
Forum

test
```

Original people file:

```
Forum

a,0
```

Forum file:

```
Forum

2000-01-01T10:10:10
a
cool!?
```

Given the above files, the program should modify the people file to the following:

```
Forum

a,1
```