Introduction

This is a very quick practice of some basic concepts of classes and inheritance, adapted from *here (https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-0001-introduction-to-computer-science-and-programming-in-python-fall-2016/)*.

The program that you have to implement some parts of it have to be able to filter the news, alter the user when it notices a news story that matches that user's interests.

Your solutions is better to be kept under 15-20 lines of codes or in some problems much less than that.

Start From Here:

Download minipr4.zip. This file includes all the files you need, including:

- mp4.py, a skeleton for you to fill in
- mp4_test.py, a test file that helps you to check your answers
- triggers.txt, a trigger configuration file
- feedparser.py a module that will retrieve and parse feeds for you

You use an RSS (Really Simple Syndication) feed reader instead of web browser to get data. Everything from retrieving and parsing the XML files is taken care of by feed parser.

RSS Feed Structure

Google News is an example of RSS feed. The URL for **Google News** feed is: *(http://news.google.com/?output=rss (http://news.google.com/?output=rss))* .

If you load this URL in your browser, you will see the XML code generated by the feed which is not human readable. But in facat if you have RSS feed reader, then you'll see a list of items. Each item represents a single news. In Google News feed, every item has the following fields:

- **guid**: A globally unique identifier for this news story.
- **title**: The news story's headline
- **description**: A summary of the news story
- **link**: A url link to the website
- **pubDate**: Date the news was published
- **category**: News category, such as "Health"

Check _here (https://www.w3schools.com/xml/xml_rss.asp)_ if you want to know more about RSS.

Final Goal

we want to be able to read news stories from different RSS feeds all in one place. Google News RSS is one of the RSS feeds we want to use here. Since each of the RSS feeds is structured a bit different than the others, here the goal is to come up with a unified application that aggregates several RSS feeds from different sources and handle them exactly the same.

Part1

Parsing information from the news feeds is cumbersome (as you may experienced before!). Here someone has already done this part and just has left you with some variables that contain information for a news story.

Your task in this part is to write a class (in mp4.py), *NewStory*, starting with an initializer that takes (*guid, title, description, link, pubdate*) as arguments and stores them appropriately. *NewStory* also needs to contain the following methods:

- *get_guid(self)*
- *get_title(self)*
- *get_description(self)*
- *get_link(self)*
- *get_pubdate(self)*

The solution for this part is short and straightforward.

Triggers

Alerts will be generated for a subset of a given set of news stories by your program. Users will be notified about those stories with alerts while the other stories will be silently discarded.

A trigger is a rule that is already checked over a single news story and may generate an alert. Here the collection of alerting rules is called triggers.

For example, a simple trigger could be set up to fire for all news stories whose title contain "Weather". Another trigger could fire for every news story whose description contained "California". Or in a specific case, one trigger could fire only when both of the mentioned phrases are in description.

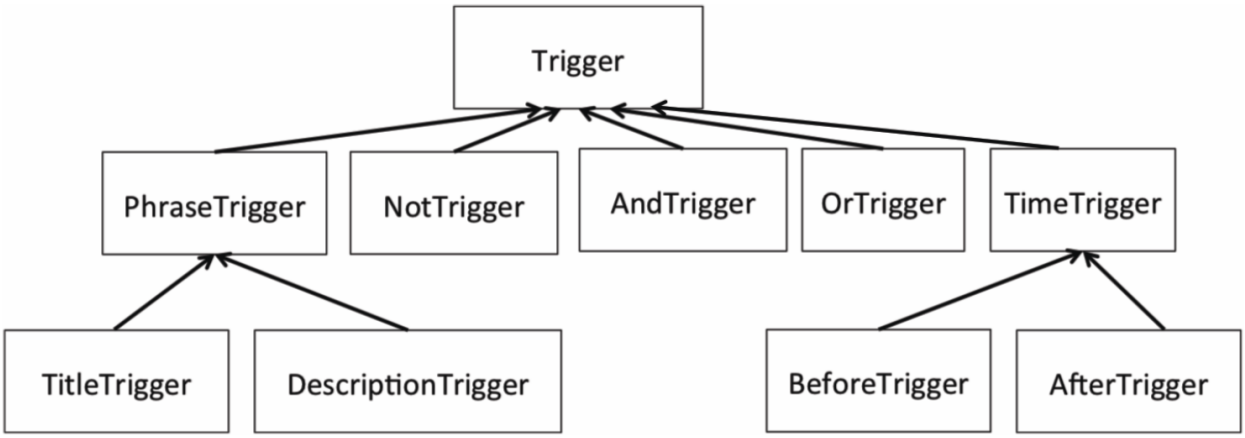
To simplify the code, we need to define a trigger interface. The trigger interface will be implemented in deifferent ways by a number of different trigger classes.

Trigger Interface

The class shown below implements the *Trigger* interface (you shouldn't modify it). Each trigger class you define inherits from it and by default will have the *evaluate* method. In that case, calls to this method will go to *Trigger.evaluate( )*, which fails with *NotImplementdError*.

```
class Trigger(object):
    def evaluate(self, story):
        """
        Return True if an alert should be generated for the given
        news item, or False otherwise.
        """
        raise NotImplemetned Error
```

In the figure below, the *Trigger* class as a superclass and subclasses that we want to define by inheriting from that are shown. The arrow from *PhraseTrigger* to *Trigger* means it inherits from *Trigger*.



Phrase Triggers

We want to implement triggers that alert user about news items that contain specific phrases. A simple trigger could fire for every news item whose description contains the phrase "California".

A **phrase** is one or more words separated by a single space between the words without any punctuation. Followings are some examples of valid phrases:

- 'these are valid phrases'
- 'microsoft office'
- 'MICROSOFT OFFICE'
- 'vowwww'

followings are **NOT** valid phrases:

- 'microsoft office!!!!' (contains punctuation)
- 'microsoft office' (contains multiple spaces between words)

Trigger should fire only when each word in the phrase is present in the text in the exact format. The trigger should not be case sensitive.

Lets see when trigger should fire and when it shouldn't. A phrase trigger with the phrase 'microsoft office' should fire on the following text snippets:

- 'MICROSOFT OFFICE'
- 'microsoft office'
- 'MICROSOFT office'
- 'Microsoft!?? Office!!'
- 'microsoft@#&office'
- 'microsoft office'

But it shouldn't fire on the following text snippets:

- 'Microsoft offices'
- 'microsoft and its office'
- 'office 365'
- 'microsoftoffice'

You need to handle punctuations using string methods such as *split, replace, join, ...* as well as *lower* and *upper* method for case sensitivity.

Part2

Implement *PhraseTrigger* class. It takes a string *phrase* as an argument to the class initializer. This tigger is not case sensitive.

PhaseTrigger is a subclass of *Trigger* class. It is an abstract class, so we will not be directly instantiating it.

It has one new method, *is_phrase_in*, which has one string text argument. It turns *True* if the whole *phrase* is in the given text, *False* otherwise. This method is not case sensitive.

Part3

Implement *TitleTrigger* class that fires and generates alarm when a given phrase presents in a news item's title.

Again the *phrase* is an argument to the class initializer and the trigger is not case sensitive.

When implementing the class, think what method should be defined and what method should be inherited from the superclasses.

All subclasses that inherit from *Trigger* interface should include a working *evaluate* method.

You need to test your code using the unit tests. If your code FAILs it means it runs but the answer is wrong. But if you get ERROR it means your code crashes due to some error.

Part4

Implement *DescriptionTrigger* class that fires and generates alarm when a given phrase presents in a news item's description.

The rest of the processing is the same as Part3.

Check if your *DescriptionTrigger* can pass test suite.

Time Trigger

In this part we want to implement triggers that fire on the basis of the time a *NewsStory* was published not on its news contents. Look at the earlier diagram to follow the inheritance strucure of triggers.

Part 5

Implement *TimeTrigger* class. The class initializer takes time in EST as a string in the format of "1 Apr 2018 17:00:00".

Convert time from string to datetime before saving it as an attribute. You may use some of datetime's methods such as *strptime* and *replace* or _here (https://docs.python.org/2.7/library/datetime.html#strftime-and-strptime-behavior).

This is an abstract class, so we will not be directly instantiating any *TimeTrigger*.

Part6

Implement *BeforeTrigger* and *AfterTrigger* classes which are two subclasses of *TimeTrigger*.

BeforeTrigger generates alarm when a story is published strictly before the trigger's time and *AfterTrigger* fires when a story is published strictly after the trigger's time.

Note that their *evaluate* method shold not be more than a couple of lines of code.

Once you are done, the *BeforeAndAfterTrigger* unit tests in test suite should pass.

Copmosite Triggers

We want to compose the previous triggers to make more interesting and powerful triggers. For example, we may want to raise an alert when both "Microsoft office" and "stock" are present in the news item. Using just PhraseTrigger, we cannot get to this.

Part7

Implement *NotTrigger* class. It takes other trigger as an argument and produces it's output by inverting the output of that other trigger.

After it is done, the *NotTrigger* unit tests should pass.

Part8

Implement *AndTrigger*.

This trigger takes two triggers as arguments to its initializer and should raise alert on a new stroy only if both of the inputted triggers would raise alert on the given item.

After it is done, *AndTrigger* unit tests should pass.

Part9

Implement *OrTrigger*.

This trigger takes two triggers as arguments to its initializer and should raise alert on a new stroy if either one (or both) of the inputted triggers would raise alert on the given item.

After it is done, *OrTrigger* unit tests should pass.

Filtering

At this point, mp4.py can fetch and display all Google and Yahoo news items in a pop up window. But the goal is to filter out the stories we wanted.

Part10

Write a function, *filter_stories(stories, triggerlist)* that takes in two arguments: a list of news stories and a list of triggers and returns a list of the stories for which a trigger fires.

After completing this part, you can try running mp4.py and you'll see a pop up window containing different RSS news items. The triggers for these items are already defined in the code but you can change them to what is currently in the news now. The code runs in an infinite loop and checks the RSS feeds for news story every 120 seconds.

User Specified Triggers

Currently, your triggers are specified in your Python code and each time you want to change them, you have to edit your program. Instead you can read your trigger configuration from triggers.txt file everytime your application starts.

Consider the following example of configuration file:

// description trigger named t1

t1, DESCRIPTION, Presidential Election

// title trigger named t2

t2, TILTE, HILLARY CLINTON

// description trigger named t3

t3, DESCRIPTION, Donald Trump

// composite trigger named t4

t4, AND,t2,t3

// the trigger list contains t1 and t4

ADD,t1,t4

The example file shows that four triggers are created and two of those are added to the trigger list:

- t1: A trigger that fires when the description contains the phrase 'Presidential Election'
- t4: A trigger that fires when the description contains 'Donald Trump' and the title contains 'HILLARY CLINTON'.

The two other triggers (t2, t3) are used as an arguments for the AND trigger (t4) and are not added directly to the trigger liist.

The following is what each line does in the file:

Blank lines:

These lines consist only of whitespaces and are ignored.

Comment lines:

These lines begin with // and are ignored.

Trigger definitions:

The first element in a trigger definition is either the keyword ADD or the name of the trigger. Named triggers are defiend in lines that do not begin with ADD Keyword. The name can be any combination of letter or numbers (e.g., t2). The second element of a trigger definition is a keyword (e.g., TITLE, DESCRIPTIION, AND, etc.) that specifies the type of the trigger being defined. The remaining elements of the trigger definition are the trigger arguments which depend on the trigger type you already defined:

- **TITLE**: one phrase
- **DESCRIPTION**: one phrase
- **AFTER**: one correctly formatted time string
- **BEFORE**: one correctly formatted time string
- **NOT**: the name of the treigger that will be NOT'd
- **AND**: the names of the two triggers that will be AND'd
- **OR**: the names of the two triggers that will be OR'd

Trigger addition:

A trigger definition creates a trigger name and assigns a name to it but it doesn't automatically add it to the list. We need to define one or more ADD lines in the trigger congfiguration file to specify which one of the triggers should be in the trigger list. These lines start with ADD keyword following the names of one or more triggers defined previously separated by commas.

Part11

The function *read_trigger_config(filename)* is written to open the file and clean up all blank lines and comments. It should return the list of triggers specified by the configuration file. Finish the implementation.

After that's done, you need to modify the code with *main_thread* to use the trigger list specified in your configuration file, instead the one already defined in the code.

After implementing read_trigger_config, uncomment this line:

#triggerlist = read_trigger_config('triggers.txt')

After completing this part, you can run mp4.py and depending on your triggers.txt file various RSS news items will pop up.

Hint1: You may find using a dictionary where the keys are trigger names to be helpful here.

Hint2: You can define helper function if you want but it is not required.

Hint3: If no stories pop up, change the triggers to ones that reflect current news.