

NATIONAL UNIVERSITY OF SCIENCE & TECHNOLOGY

SCHOOL OF MECHANICAL & MANUFACTURING ENGINEERING (SMME)



FUNDAMENTALS OF PROGRAMMING II

End Semester Project

CLASS:	ME-15 (Section C)
TITLE:	Project Report
SUBMITTED BY:	Rameen Fatima – 453962 Bushra Farooq - 479973 Abeer Zahra Jafari – 476474 Toseef Haider – 457249 Ali Zeeshan Rauf - 467537
SUBMITTED ON:	22/05/2024

END SEMESTER PROJECT – REPORT

❖ ABSTRACT

The goal of this project is to apply the fundamental concepts of object-oriented programming (classes and inheritance) to develop a python program capable of taking RSS feeds from various online websites and then sorting through them to display news' stories most relevant to the user on the screen as output.

❖ PROJECT PARAMETERS

For our ease the code was broken down into 11 parts. Along with a basic skeleton of the code, we were also provided with a few files with pre-written code to aid our code-writing process.

- project_test.py, a test suite that will help you check your answers
- triggers.txt, a trigger configuration file
- feedparser.py, a module that will retrieve and parse feeds for you
- project_util.py, a module that converts simple HTML fragments to plain text
- mtTkinter.py, a module that handles graphic interface

The three modules (feedparser.py, project_util.py, mtTkinter.py) are necessary for this project to work, but are outside the scope of our syllabus. The same holds true for the concept of parsing and the knowledge of RSS feed (beyond a basic know how)

❖ CODE EXPLANATION

```

class NewsStory:
    def __init__(self, guid, title, description, link, pubdate):
        self.guid = guid
        self.title = title
        self.description = description
        self.link = link
        self.pubdate = pubdate

    3 usages (3 dynamic)
    def get_guid(self):
        return self.guid

    3 usages (3 dynamic)
    def get_title(self):
        return self.title

    3 usages (3 dynamic)
    def get_description(self):
        return self.description

    1 usage (1 dynamic)
    def get_link(self):
        return self.link

    3 usages (3 dynamic)
    def get_pubdate(self):
        return self.pubdate

```

Since parsing is not part of our project, our RSS feed is already parsed. Now the first part of the problem is to create a class 'NewsStory' in which to store the parsed information as the object. Here we will be taking the globally unique identifier (GUID), link, description, title and category as the arguments and then use a default constructor to assign values to them. And finally we will create individual functions to call each of these arguments later on in this code.

```

5 usages
class Trigger(object):
    56 usages (56 dynamic)
    def evaluate(self, story):
        """
        Returns True if an alert should be generated
        for the given news item, or False otherwise.
        """
        raise NotImplementedError

```

Now we will create another class 'Triggers' to store the various triggers that will help sort the feed. This class would be an abstract class which would serve as a basis for other sub-classes. The purpose of defining the 'evaluate' function here is to make sure that all the sub-classes take the story as input and then perform some action on it, otherwise we would receive a 'not implemented error'.

```

2 usages
class PhraseTrigger(Trigger):#yeh trigger ka subclass
    def __init__(self, phrase):
        self.phrase = phrase.lower()

2 usages
def is_phrase_in(self, text):
    text = text.lower()
    for char in string.punctuation:
        text = text.replace(char, ' ') # punctua
    words = text.split() # text ko individual wo
    phrase_words = self.phrase.split() # phrase
    for i in range(len(words) - len(
        phrase_words) + 1): # if lenght of s
        phrase_found = True

    # Check each word in the phrase
    for j in range(len(phrase_words)):
        if words[i + j] != phrase_words[j]:
            # If any word does not match, the
            phrase_found = False
            break

    # If all words matched, return True
    if phrase_found:
        return True

```

Now using the concept of inheritance, we are going to create a sub-class 'PhraseTrigger'. This part of the code compares the words in the text received from RSS feed to the words in our chosen phrase to see if they match.

But the problem here is that the text received from the RSS feed is not in string format and it may contain a lot of unrequired punctuation. So once the text has been assigned a value through a constructor and made case insensitive, we use a for loop along with an in-built python library to iterate over the text and then replace the punctuations with blank spaces. Then by using the 'split' function we break both our phrase and the RSS feed text into separate words.

Next by using nested for loops we try to find the keywords in the text by comparing each word in our phrase to the words in the text. If those words are present the function returns a true value.

```

class TitleTrigger(PhraseTrigger):#eik aur subclass
    56 usages (56 dynamic)
    def evaluate(self, story):
        return self.is_phrase_in(story.get_title())

```

```

class DescriptionTrigger(PhraseTrigger):
    56 usages (56 dynamic)
    def evaluate(self, story):
        return self.is_phrase_in(story.get_description())

```

Both these triggers are a sub-class of the phrase trigger which means they inherit all the properties of the parent class. Their job is to compare the titles and the description of the RSS feed and see if they contain our keywords.\

```

class TimeTrigger(Trigger):
    def __init__(self, time_str):
        time_format = "%d %b %Y %H:%M:%S"
        est = pytz.timezone('EST')
        self.time = est.localize(datetime.strptime(time_str, time_format))

# Problem 6

```

```

class BeforeTrigger(TimeTrigger):
    56 usages (56 dynamic)
    def evaluate(self, story):
        story_pubdate = story.get_pubdate()
        if story_pubdate.tzinfo is None:
            story_pubdate = pytz.timezone('EST').localize(story_pubdate)
        return story_pubdate < self.time

```

```

class AfterTrigger(TimeTrigger):
    56 usages (56 dynamic)
    def evaluate(self, story):
        story_pubdate = story.get_pubdate()
        if story_pubdate.tzinfo is None:
            story_pubdate = pytz.timezone('EST').localize(story_pubdate)
        return story_pubdate > self.time

```

This part of the code deals with the time-envelope of the news feed. The first task is to convert the publishing date/time of our choice into a standard format.

Now the 'BeforeTime' and 'AfterTime' functions, retrieve the publishing date of the article using the functions defined in the first part and then make sure it's in the right time zone. The 'BeforeTrigger' returns a true value if the publishing date is before the user determined one, and the same goes for the 'AfterTrigger' which is true when the publishing date of the article is after the user determined one.

```
class NotTrigger(Trigger):
    def __init__(self, trigger):
        self.trigger = trigger

    56 usages (56 dynamic)
    def evaluate(self, story):
        return not self.trigger.evaluate(story)
```

```
class AndTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

    56 usages (56 dynamic)
    def evaluate(self, story):
        return self.trigger1.evaluate(story) and self.trigger2.evaluate(story)
```

```
class OrTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

    56 usages (56 dynamic)
    def evaluate(self, story):
        return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)
```

The And, OR and Not triggers are again the subclass of the class 'Triggers'. The not function filters out any story containing the phrase entered as its argument. Whereas the And and Or triggers help when we want more than phrase to present in our desired news articles.


```

def filter_stories(stories, triggerlist):
    """
    Takes in a list of NewsStory instances.
    Returns: a list of only the stories for which a trigger in triggerlist fires.
    """
    filtered_stories = []
    for story in stories:
        for trigger in triggerlist:
            if trigger.evaluate(story):
                filtered_stories.append(story)
                break
    return filtered_stories

```

This function takes a list of triggers defined below and with the help of a nested for loop and an if-else statement, isolates the stories that pass all of our triggers. Such stories are stored in the empty list 'filtered_stories' defined at the beginning of the function.

```

def read_trigger_config(filename):
    trigger_map = {
        'TITLE': TitleTrigger,
        'DESCRIPTION': DescriptionTrigger,
        'BEFORE': BeforeTrigger,
        'AFTER': AfterTrigger,
        'NOT': NotTrigger,
        'AND': AndTrigger,
        'OR': OrTrigger
    }
    trigger_file = open(filename, 'r')
    lines = []
    for line in trigger_file:
        line = line.rstrip()
        if not (len(line) == 0 or line.startswith('//')):
            lines.append(line)

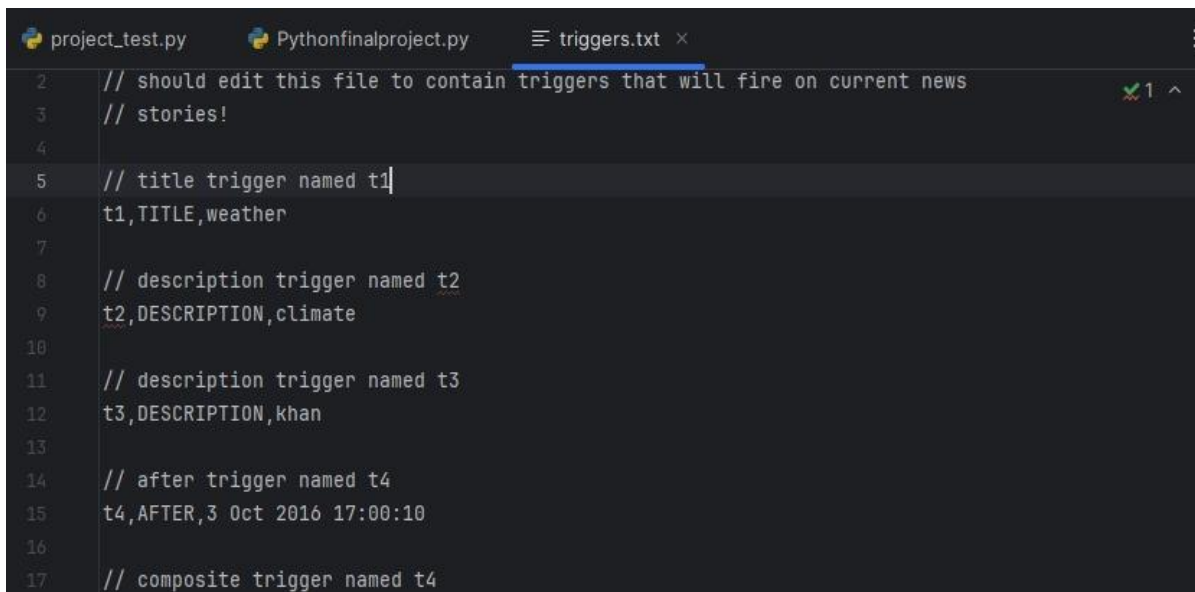
    triggers = {}
    trigger_list = []

    for line in lines:
        parts = line.split(',')
        if parts[0] == 'ADD':
            for name in parts[1:]:
                trigger_list.append(triggers[name])
        else:

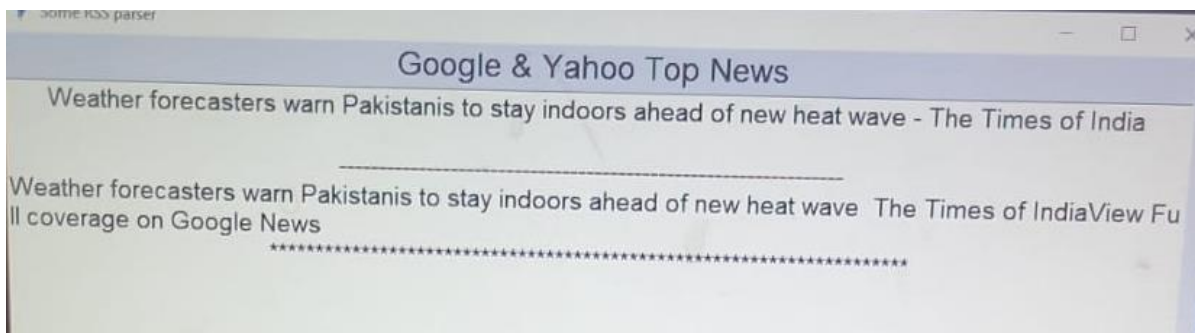
```

This last problem creates a dictionary to store all the triggers. And then stores this in a file. Most of our code depends upon taking input from the user regarding the phrases they want to see in their feed and the ones they don't. This part of the code deals with determining those phrases and then integrating them into the rest of the program.

❖ OUTPUT

A screenshot of a code editor with three tabs: 'project_test.py', 'Pythonfinalproject.py', and 'triggers.txt'. The 'triggers.txt' tab is active and shows a list of triggers. The code is as follows:

```
2 // should edit this file to contain triggers that will fire on current news
3 // stories!
4
5 // title trigger named t1
6 t1,TITLE,weather
7
8 // description trigger named t2
9 t2,DESCRIPTION,climate
10
11 // description trigger named t3
12 t3,DESCRIPTION,khan
13
14 // after trigger named t4
15 t4,AFTER,3 Oct 2016 17:00:10
16
17 // composite trigger named t4
```



❖ FUTURE IMPLICATIONS

This project has significant future implications for enhancing personalized news consumption. By refining and expanding filtering criteria, the system can offer highly tailored news feeds, adapting over time based on user behavior. Incorporating machine learning algorithms could enable more sophisticated

content analysis, improving the relevance of news stories presented. Enhancements for real-time updates and notifications could ensure users are immediately informed of breaking news.

Developing mobile and web versions would increase accessibility, broadening the user base. Integrating with social media platforms could facilitate sharing and discussion of news stories, fostering community engagement. Aggregating and analyzing user data could provide valuable insights into news consumption patterns, useful for targeted advertising and informing content creators.

Leveraging natural language processing techniques could enhance content analysis accuracy, understanding context and sentiment. The project also has potential for monetization through subscription models, premium features, or partnerships with news outlets, providing a sustainable business model. Overall, future developments could lead to a more intelligent, versatile, and widely-used news filtering tool.

❖ CONCLUSION

In conclusion, this project successfully demonstrates a powerful tool for personalized news consumption by implementing a system that filters and delivers relevant news stories based on user-defined criteria. By leveraging various types of triggers, the project showcases the potential for enhancing the way individual's access and interact with news content.

The system's ability to process multiple RSS feeds and apply complex filtering rules ensures that users receive tailored news updates, aligning with their interests and preferences. As news consumption habits continue to evolve, the RSS Feed Filter project positions itself as a forward-thinking solution that can adapt and grow to meet the demands of modern users.

❖ REFERENCES

- <https://rss.com/blog/how-do-rss-feeds-work/>
- <https://pythontutor.com/>