FOP II PROJECT

ME-15 (A)

Group members: -

Muhammad Shehryar *461599*

Muhammad Shan e Haider *456500*

Ifra Murtaza *454546*

Hamad Arshad *481644*

```python
import feedparser
import string
import time
import threading
from project_util import translate_html
from tkinter import *
from datetime import datetime

def process(url):
    feed = feedparser.parse(url)
    entries = feed.entries
    ret = []
    for entry in entries:
        guid = entry.guid
        title = translate_html(entry.title)
        link = entry.link
```

1. feed-parser: A module that parses RSS feeds.
2. string: A module that handles typical string operations.
3. time: A module that handles time-related functions.
4. Threading is a module that allows many threads to execute concurrently.
5. project_util: A custom module that likely contains utility functions like translate_html.
6. Tkinter is a module for generating graphical user interfaces.
7. Datetime is a module for manipulating dates and times.

def process(url): Defines a function named process that accepts a single parameter, url, which is the RSS feed URL.
feed = feedparser.parse(url): Uses feedparser to extract the RSS feed from the provided URL. The resulting object contains the parsed feed data.
entries = feed.entries: Returns the list of entries (news items) from the parsed feed.
ret = []: Sets up an empty list ret to hold processed news stories.
for entry in entries: iterates through each entry in the entries list.
guid = entrance.guid: Retrieves the guid (Globally Unique Identifier) of the entry, which uniquely identifies the news item.
title = translate_html(entry.title): Extracts the entry's title and processes it with the translate_html function from the project_util module, which handles any HTML entities or special characters.
link = entry.link: Retrieves the URL link to the complete news story.

The process function accepts an RSS feed URL, parses it, then iterates over each news entry. It retrieves and processes important information like the unique

identifier (guid), the title (title), and the link to the full article. The function prepares these fields for other uses, such as filtering or displaying in a user interface. The extracted and processed data will then be used to build instances of the NewsStory class and store them in the ret list (albeit the remainder of the process function is not displayed in this excerpt).

```
        if 'description' in entry:
            description = translate_html(entry.description)
        else:
            description = ""

        if 'published' in entry:
            pubdate_str = entry.published
        elif 'published_parsed' in entry:
            pubdate_str = time.strftime('%a, %d %b %Y %H:%M:%S %Z', entry.published_parsed)
        else:
            continue

        try:
            pubdate = datetime.strptime(pubdate_str, "%a, %d %b %Y %H:%M:%S %Z")
        except ValueError:
            pubdate = datetime.strptime(pubdate_str, "%Y-%m-%dT%H:%M:%SZ")

        newsStory = NewsStory(guid, title, description, link, pubdate)
        ret.append(newsStory)
    return ret
```

DESCRIPTION PART: -

Checks whether the entry (which represents a news story from the RSS feed) contains a description field.

If it does ('description' in entry), the description is retrieved using entry.description and then converted to HTML with the translate_html function (which is most likely written elsewhere in the code).

If there is no description field (otherwise), the description variable is set to an empty string.

PUBLICATION DATE: -

Attempts to obtain the publishing date from the published field of the record (if 'published' is present).

If the published field exists, it is presumed to be a formatted string and stored in the pubdate_str variable.

If the published field is not present, it looks for published_parsed (elif 'published_parsed' in entry).

If published_parsed exists, it is considered to be a date object in a predefined format. The code formats this date object using time.strftime into a human-

readable string (%a for weekday,%d for day,%b for month, etc.) and assigns it to pubdate_str.
If neither published nor published_parsed exist (otherwise), the code skips over this step and moves on to the next one.

PARSING PUBLICATION DATE STRING: -

Attempts to convert the pubdate_str (which contains the publishing date in string format) to a datetime object using datetime.strptime.
It tries to parse using the format "%a,%d%b%Y%H:%M:%S%Z," which indicates a regular weekday, date, time, and timezone (e.g., "Fri, 24 May 2024 18:54:25 +0530").
If the parsing fails with a ValueError, it tries the alternate format "%Y-%m-%dT%H:%M:%SZ" that is often used in RSS feeds (for example, "2024-05-24T18:54:25Z").

CREATING NEWSTORY OBJECT: -

After the description and publication date have been handled, a NewsStory object is generated utilizing the extracted information (guid, title, description, link, and pubdate).

ADD NEWSTORY TO LIST: -

After the description and publication date have been handled, a NewsStory object is generated utilizing the extracted information (guid, title, description, link, and pubdate).

This section of the code handles the description and publication date of a news story retrieved from an RSS feed item. Let us break it down step by step. This code guarantees that the software can handle the various ways in which publication dates and descriptions may be provided in RSS feeds, converting them into standard formats for further processing.

```python
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

    def get_guid(self):
        return self.guid

    def get_title(self):
        return self.title

    def get_description(self):
        return self.description

    def get_link(self):
        return self.link

    def get_pubdate(self):
        return self.pubdate
```

A class NEWSTORY is defined: -

Define __init__(self, guid, title, description, url, pubdate): This is the class's constructor method. It is invoked whenever you create a new NewsStory object. It requires five arguments:

Guid is a unique identifier for the news story.

Title: The title of the news item.

Description: An overview of the news story.

The link is the news story's URL.

The publishing date of the news piece (often a datetime object).

In the constructor (self.guid = guid, etc.):

The code assigns the supplied values to the object's instance variables. These variables are prefixed with self to indicate that they are particular to the object being formed.

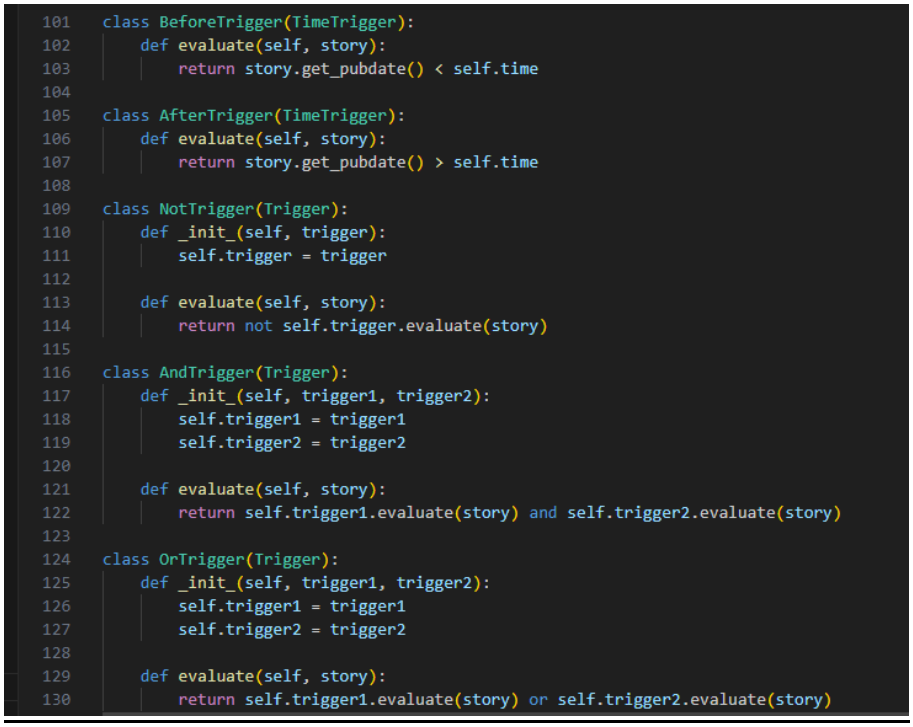
ACCESSORS METHOD: -

Define get_guid(self): - This function does not require any parameters (self already refers to the object).

It simply returns the value of the guid instance variable by calling self.guid. Similar methods (get_title, get_description, get_link, get_pubdate) have been defined to access the object's corresponding instance variables (title, description, link, and pubdate).

The NewsStory class serves as a template for constructing objects containing information about news stories. The constructor populates the object with necessary data, and the getter functions allow you to access the stored information about a single news story object.

```
70  class Trigger(object):
71      def evaluate(self, story):
72          raise NotImplementedError
73
74  class PhraseTrigger(Trigger):
75      def _init_(self, phrase):
76          self.phrase = phrase.lower()
77
78      def is_phrase_in(self, text):
79          text = text.lower()
80          for char in string.punctuation:
81              text = text.replace(char, ' ')
82          text_words = text.split()
83          phrase_words = self.phrase.split()
84          for i in range(len(text_words) - len(phrase_words) + 1):
85              if text_words[i:i + len(phrase_words)] == phrase_words:
86                  return True
87          return False
88
89  class TitleTrigger(PhraseTrigger):
90      def evaluate(self, story):
91          return self.is_phrase_in(story.get_title())
92
93  class DescriptionTrigger(PhraseTrigger):
94      def evaluate(self, story):
95          return self.is_phrase_in(story.get_description())
96
97  class TimeTrigger(Trigger):
98      def _init_(self, time):
99          self.time = datetime.strptime(time, "%Y-%m-%dT%H:%M:%SZ")
```



```python
class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        return story.get_pubdate() < self.time

class AfterTrigger(TimeTrigger):
    def evaluate(self, story):
        return story.get_pubdate() > self.time

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

    def evaluate(self, story):
        return not self.trigger.evaluate(story)

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

    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

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

CLASSES: -

These classes (Trigger, PhraseTrigger, etc.) serve as a template for developing objects that define filtering criteria for news items.
Subclasses must implement the abstract function evaluate(self, tale) from the base class Trigger in order to establish their particular filtering mechanism.

MATCHING TEXTS: -

PhraseTrigger looks for a certain phrase (in lowercase) in a news story's title or description.
It has a method is_phrase_in(self, text) that determines whether the provided text (also lowercase and punctuation-free) contains the trigger phrase.

MATCHING TIMES: -

TimeTrigger maintains a datetime object that represents a certain time.
Subclasses such as BeforeTrigger and AfterTrigger inherit from TimeTrigger and compare the stored time to the story's publishing date.

TRIGGERS USING LOGIC STATEMENTS: -

NotTrigger negates the outcome of another trigger's examination.
AndTrigger needs that both provided triggers match for a story to pass.
To pass a tale, at least one of the stated triggers must match.

```
#======================
# Filtering
#======================

def filter_stories(stories, triggerlist):
    filtered_stories = []
    for story in stories:
        for trigger in triggerlist:
            if trigger.evaluate(story):
                filtered_stories.append(story)
                break
    return filtered_stories

#======================
# User-Specified Triggers
#======================

def read_trigger_config(filename):
    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)
    trigger_file.close()

    triggers = {}
    trigger_list = []

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

```
triggers = {}
trigger_list = []

for line in lines:
    parts = line.split(',')
    if parts[0] == 'ADD':
        for name in parts[1:]:
            if name in triggers:
                trigger_list.append(triggers[name])
    else:
        trigger_name = parts[0]
        trigger_type = parts[1]
        if trigger_type == 'TITLE':
            triggers[trigger_name] = TitleTrigger(parts[2])
        elif trigger_type == 'DESCRIPTION':
            triggers[trigger_name] = DescriptionTrigger(parts[2])
        elif trigger_type == 'AFTER':
            triggers[trigger_name] = AfterTrigger(parts[2])
        elif trigger_type == 'BEFORE':
            triggers[trigger_name] = BeforeTrigger(parts[2])
        elif trigger_type == 'NOT':
            if parts[2] in triggers:
                triggers[trigger_name] = NotTrigger(triggers[parts[2]])
        elif trigger_type == 'AND':
            if parts[2] in triggers and parts[3] in triggers:
                triggers[trigger_name] = AndTrigger(triggers[parts[2]], triggers[parts[3]])
        elif trigger_type == 'OR':
            if parts[2] in triggers and parts[3] in triggers:
                triggers[trigger_name] = OrTrigger(triggers[parts[2]], triggers[parts[3]])

return trigger_list
```

Filtering stories Function:-

The function filter__stories (filtering stories): triggerlist (a list of trigger objects) and stories (a list of stories) are the two parameters required by this function. It goes through every story in the list of stories once. Every trigger in the triggerlist is iterated over for every tale. It uses each trigger object's evaluate method to determine if the trigger evaluates to true for the current tale. The narrative gets appended to the filtered_stories list if a trigger for it evaluates to true. Ultimately, the filtered tales list—which only includes the articles that met the trigger conditions—is returned.

User Specified Trigger Function:-

The function read_trigger_config (user-specified triggers): A filename is entered into this function. Line by line, the content of the specified file is read while it is opened in read mode. Lines that begin with "//" (comments) and are empty are ignored. To hold non-empty and non-comment lines, it keeps a list named lines. The triggers dictionary, which stores trigger objects, and the trigger_list dictionary, which stores the final list of triggers, are initialized. Every line in the lines list that is not a comment is iterated over. It extracts distinct sections of the trigger configuration for each line by splitting it up using commas. The trigger object (e.g., Title Trigger, Description Trigger, After Trigger, etc.) is created in accordance with the trigger type that is given in the configuration. The generated trigger object is added to the triggers dictionary. When a trigger type is "ADD," previously defined triggers are added to the trigger_list. Ultimately, it yields the trigger_list, including every trigger object listed in the configuration file.

```
SLEEPTIME = 120  # seconds

def main_thread(master, keywords):
    try:
        triggerlist = []
        if keywords:
            for keyword in keywords:
                triggerlist.append(OrTrigger(TitleTrigger(keyword), DescriptionTrigger(keyword)))

        frame = Frame(master)
        frame.pack(side=BOTTOM)
        scrollbar = Scrollbar(master)
        scrollbar.pack(side=RIGHT, fill=Y)

        t = "Google & Yahoo Top News"
        title = StringVar()
        title.set(t)
        ttl = Label(master, textvariable=title, font=("Helvetica", 18))
        ttl.pack(side=TOP)
        cont = Text(master, font=("Helvetica", 14), yscrollcommand=scrollbar.set)
        cont.pack(side=BOTTOM)
        cont.tag_config("title", justify='center')
        button = Button(frame, text="Exit", command=master.destroy)
        button.pack(side=BOTTOM)
        guidShown = []
```

- **EXPLANATION:**
- **Importing Libraries** : The code presumably starts by importing necessary libraries such as `tkinter`,whixh is a python library used for constructed basic graphical user interface
- **SLEEPTIME:** This constant, set to 120 seconds, might be used to define the interval between updates or checks for new news stories.
- **main_thread Function:** This function initializes the main part of the GUI, sets up triggers for filtering news articles based on keywords, and creates the user interface components.
- **Trigger List:** This part initializes an empty list called `triggerlist` and populates it with triggers based on the provided keywords. These triggers are used to filter news stories.
- **Custom triggers:** `OrTrigger`, `TitleTrigger`, and `DescriptionTrigger` are custom trigger classes that probably evaluate whether a news story's title or description contains any of the keywords.
- **Frame and Scrollbar:** These lines set up the main frame and scrollbar for the text widget that will display the news stories.
- **Title Label:** This part sets up a label at the top of the window to display the title "Google & Yahoo Top News."
- **Content Text Box:** A text widget is created to display the news stories, with the scrollbar linked to it.

- **Exit Button:** A button is added to the frame to allow the user to close the application.
- **GUID Shown List:** This list keeps track of GUIDs (Globally Unique Identifiers) of the displayed news stories to avoid showing duplicates.
- **get_cont Function:** This function, which is defined but not fully shown in the snippet, likely handles retrieving and displaying the content of new stories. It may involve fetching data from news sources, filtering the stories using the triggers, and updating the content text widget.
- **OUTPUT**
- When this code is run:
- A GUI window appears with the title "Google & Yahoo Top News."
- The window includes a text area for displaying news stories and a scrollbar for navigation.
- An "Exit" button at the bottom allows the user to close the application.
- The application initializes triggers based on provided keywords to filter news stories.
- The function `get_cont` (if fully implemented) would fetch and display news stories in the text widget, filtering them according to the specified keywords and ensuring no duplicate stories are shown.

```
        def get_cont(newstory):
            if newstory.get_guid() not in guidShown:
                cont.insert(END, newstory.get_title() + "\n", "title")
                cont.insert(END, "\n----------------------------------------------------------------------\n")
                cont.insert(END, newstory.get_description())
                cont.insert(END, "\n*\n", "title")
                guidShown.append(newstory.get_guid())

        while True:
            print("Polling...")
            stories = process("http://news.google.com/news?output=rss")
            stories.extend(process("http://news.yahoo.com/rss/topstories"))

            stories = filter_stories(stories, triggerlist)

            list(map(get_cont, stories))
            scrollbar.config(command=cont.yview)

            print(f"No keywords provided. Continuing to poll...")
            time.sleep(SLEEPTIME)

    except Exception as e:
        print(f"Error occurred: {e}")


        def get_cont(newstory):
            if newstory.get_guid() not in guidShown:
                cont.insert(END, newstory.get_title() + "\n", "title")
                cont.insert(END, "\n----------------------------------------------------------------------\n")
                cont.insert(END, newstory.get_description())
                cont.insert(END, "\n*\n", "title")
                guidShown.append(newstory.get_guid())
```

1. Setup the GUI:

- Creates a window with a title "Google & Yahoo Top News".

- Adds a text area to display news stories and a scrollbar to scroll through the stories.

- Adds an "Exit" button to close the window.

2. Main Loop:

- Continuously fetches the latest news stories every 120 seconds.
- Filters these stories based on the keywords you provided.
- Displays the filtered stories in the text area of the window.

- **cont:** A text area where the news stories will be shown.
- scrollbar: A scrollbar to scroll through the news stories.
- ```
- **get_cont:** A function that displays a new story if it hasn't been shown yet.
- **newstory.get_guid():** Checks if the story has a unique identifier (GUID) that hasn't been shown.
- **cont.insert():** Inserts the story's title, a separator line, and the story's description into the text area.
- **guidShown.append():** Adds the story's GUID to the list of shown stories to avoid duplicates.

- while True: A loop that runs forever, continuously checking for new stories.
- **process(url):** Fetches news stories from the given URL.
- **stories.extend():** Combines stories from both Google and Yahoo.
- **filter_stories(stories, triggerlist):** Filters the fetched stories based on the keywords (triggers).
- **list(map(get_cont, stories)):** For each story in the filtered list, it calls `get_cont` to display it in the text area.
- **time.sleep(SLEEPTIME):** Waits for 120 seconds before checking for new stories again.

OUTPUT

- When you run this code:
- A window will appear with the title "Google & Yahoo Top News".
- The program will continuously fetch news stories from Google and Yahoo every 120 seconds.
- It will filter these stories based on the keywords you provided.
- The filtered stories will be displayed in the text area of the window.
- Stories will include their title, a separator, and their description.

```
        while True:
            print("Polling...")
            stories = process("http://news.google.com/news?output=rss")
            stories.extend(process("http://news.yahoo.com/rss/topstories"))

            stories = filter_stories(stories, triggerlist)

            list(map(get_cont, stories))
            scrollbar.config(command=cont.yview)

            print(f"No keywords provided. Continuing to poll...")
            time.sleep(SLEEPTIME)

    except Exception as e:
        print(f"Error occurred: {e}")

if name == 'main':
    root = Tk()
    root.title("RSS Feed Filter")

    keywords = input("Enter keywords (comma-seperated): ").strip().split(',')
    keywords = [keyword.strip() for keyword in keywords if keyword.strip()]

    t = threading.Thread(target=main_thread, args=(root, keywords))
    t.start()

    root.mainloop()
```

- If you provided keywords, only stories matching those keywords will be shown. If no keywords are provided, all fetched stories will be shown.
- The "Exit" button will allow you to close the window.

Polling: This prints a message to the console indicating that the application is fetching new stories.

process("http://news.google.com/news?output=rss"): This function fetches news stories from Google's RSS feed.

stories.extend(process("http://news.yahoo.com/rss/topstories")): This line fetches news stories from Yahoo's RSS feed and adds them to the list of stories.

Filter_stories(stories, triggerlist): This function filters the fetched stories based on the keywords the user provided. Only stories that match these keywords are kept.

list(map(get_cont, stories): This line calls the `get_cont` function for each filtered story, adding the story to the text area in the GUI if it hasn't been shown already.

scrollbar.config(command=cont.yview): This updates the scrollbar so it works correctly with the text area.

time.sleep(SLEEPTIME): This pauses the loop for a specified time (120 seconds) before fetching new stories again.

Error Handling: If any error occurs during the fetching or processing of stories, it is caught here, and an error message is printed to the console.

if __name__ == '__main__': Ensures that this block of code runs only if the script is executed directly, not if it is imported as a module.

root = Tk(): Initializes the main window for the Tkinter GUI.

root.title("RSS Feed Filter"): Sets the title of the window to "RSS Feed Filter".

keywords = input("Enter keywords (comma-separated): ").strip().split(','): Prompts the user to enter keywords for filtering news stories. The keywords are separated by commas and any extra spaces are removed.

t = threading.Thread(target=main_thread, args=(root, keywords)): Creates a new thread to run the `main_thread` function, passing the main window (`root`) and the list of keywords as arguments.

t.start(): Starts the new thread.

root.mainloop(): Starts the Tkinter main event loop, which handles user interactions with the GUI.