School Of Mechanical & Manufacturing Engineering, NUST

Department of Mechanical Engineering

Fundamentals of Programing – II

Python

Lab Project

Monitoring News Feeds Over Internet

Lab Instructor: Dr. Saqib Nazir

Name	CMS ID
Muhammad Bin Ahmad	**480779**
Hussain Aun Ali	**463652**
Muhammad Abdullah Qureshi	**456523**

DATE:

22/5/24

School Of Mechanical & Manufacturing Engineering, NUST

Department of Mechanical Engineering

PROBLEMS 1-4

Muhammad Bin Ahmad

- ## Problem 1

```
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
```

This is a simple class which is designed for the sole purpose to store the information related to the news story which we get from the previous ‘**process’** function. This class has 5 attributes namely guid, title pubdate, description and link and each item is stored in separate variables. Moving on, the class also contains getter functions for each attribute.

- ## Problem 2

```
class PhraseTrigger(Trigger):
    def __init__(self, phrase):
        self.phrase = phrase.lower()

    def is_phrase_in(self, text):
        text = text.lower()
        for char in string.punctuation:
            text = text.replace(char, ' ')
        text_words = text.split()
        phrase_words = self.phrase.split()
        for i in range(len(text_words) - len(phrase_words) + 1):
            if text_words[i:i + len(phrase_words)] == phrase_words:
                return True
        return False
```

The PhraseTrigger class is a subclass of the abstract Trigger class. It is designed to determine whether a specific phrase appears within a given text, ignoring case and punctuation. The constructor method initializes a PhraseTrigger object with a specific phrase. There is only one parameter '**phrase**' which is the phrase to be checked in the texts. This method converts the input phrase to lowercase and stores it in the instance variable self.phrase. Converting to lowercase ensures that the phrase matching is case-insensitive.

The next method **is_phrase_in** checks if the phrase specified in the PhraseTrigger instance is present in the given text. It takes a single parameter **text**. The text is first converted to lowercase to ensure that it is case insensitive and then the text is iterated over and each punctuation is replaced with a whitespace. This whitespace is then utilized next to splits the text into a list of words based on whitespace. The result is stored in text_words.

phrase_words = self.phrase.split()

This line splits the stored phrase into a list of words based on whitespace. The result is stored in phrase_words. Next, we use a loop to iterate through the list of text_words. The loop variable i runs from 0 to len(text_words) - len(phrase_words), ensuring that there are enough words remaining in text_words to match phrase_words.

For each position i, it checks if the slice text_words[i:i + len(phrase_words)] matches phrase_words.

If a match is found, the method returns True. Otherwise it returns false.

In summary, the text and phrase are both converted to lowercase and any unneeded punctuations are removed so that they can be compared. If the phrase appears in the text then it returns true.

- ## Problem 3

```
class TitleTrigger(PhraseTrigger):
    def evaluate(self, story):
        return self.is_phrase_in(story.get_title())
```

The TitleTrigger class is a specialized type of PhraseTrigger. Its purpose is to determine if a specific phrase appears in the title of a news story.

TitleTrigger inherits from PhraseTrigger, which means it has access to all the methods and attributes of the PhraseTrigger class. While PhraseTrigger provides the general functionality to check for a phrase in any given text, TitleTrigger specifically applies this functionality to the title of a news story. The evaluate method determines if the phrase (specified when the TitleTrigger was created) is present in the title of the given news story. A parameter story is taken which is an instance of the NewsStory class. This class has a method get_title() which returns the title of the news story.

story.get_title()

Calls the get_title() method on the story object to retrieve the title of the news story as a string.

return self.is_phrase_in(story.get_title())

Calls the is_phrase_in method (inherited from PhraseTrigger) with the title of the story as an argument. The is_phrase_in method checks if the phrase (stored in self.phrase during the initialization of PhraseTrigger) is present in the title. This involves:

1. Converting the title to lowercase.

2. Replacing punctuation in the title with spaces.
3. Splitting the title into words.
4. Checking if the sequence of words in the phrase appears in the title.
5. If the phrase is found in the title, is_phrase_in returns True, otherwise it returns False.

- **Problem 4**

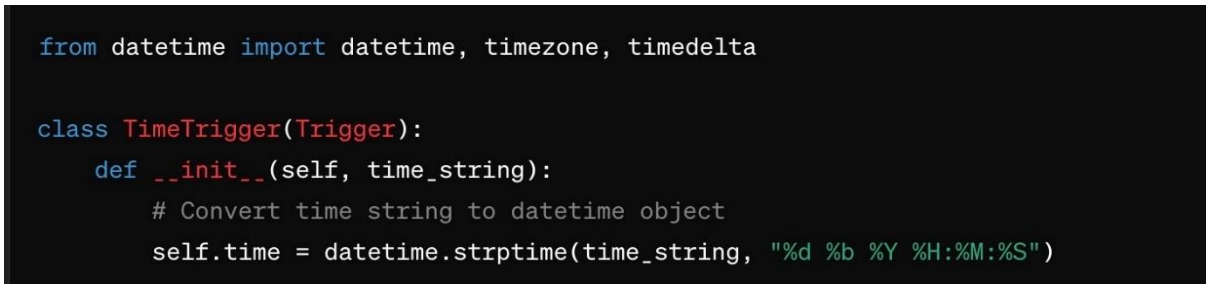
class DescriptionTrigger(PhraseTrigger):
  def evaluate(self, story):
    return self.is_phrase_in(story.get_description())

Like the class before, DescriptionTrigger is a subclass of the class PhraseTrigger. It operates in the same way as the TitleTrigger class with the only difference being that this class makes use of the descriptions instead of the title of the news stories.

PROBLEMS 4-8

Hussain Aun Ali

Problem 5



```
from datetime import datetime, timezone, timedelta

class TimeTrigger(Trigger):
    def __init__(self, time_string):
        # Convert time string to datetime object
        self.time = datetime.strptime(time_string, "%d %b %Y %H:%M:%S")
```

What This Code Does:

1. **Importing Modules**:
   - **from datetime import datetime, timezone, timedelta**: This line imports three useful classes from the **datetime** module:
     - **datetime**: This is a class for handling dates and times.
     - **timezone**: This is a class for dealing with time zones.
     - **timedelta**: This is a class for representing the difference between two dates or times.
2. **Defining a Class**:
   - **class TimeTrigger(Trigger):**: This line defines a new class called **TimeTrigger**, which inherits from another class called **Trigger**. Inheritance means that **TimeTrigger** will have all

the properties and methods of the **Trigger** class, plus any additional properties and methods you define in it.

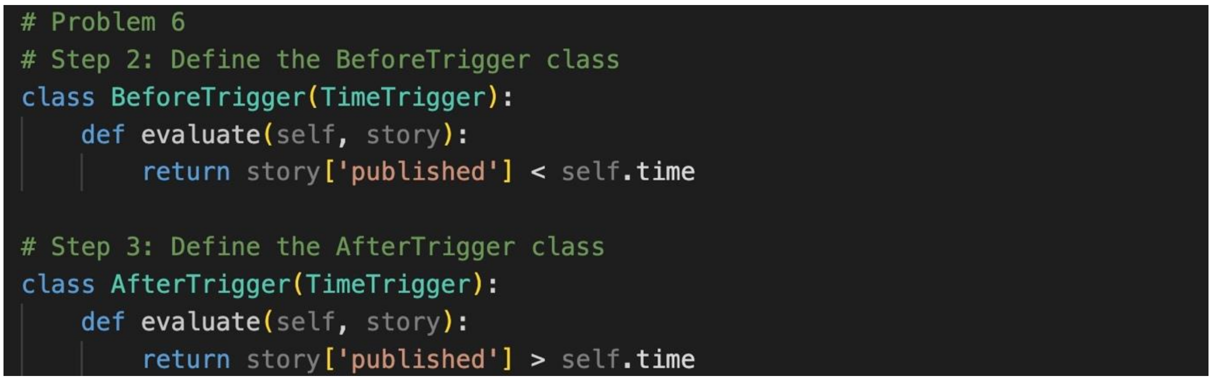
3. **Initializing the Class**:

   def __init__(self, time_string):: This line defines the constructor method for the **TimeTrigger** class. The constructor is a special method that is called when you create a new instance of the class. It takes one argument in addition to **self**, which is **time_string**.

4. **Converting a String to a DateTime Object**:
   - **self.time = datetime.strptime(time_string, "%d %b %Y %H:%M:%S")**: This line converts the **time_string** into a **datetime** object and assigns it to **self.time**.
   - **datetime.strptime(time_string, "%d %b %Y %H:%M:%S")** is a method that takes a date and time as a string (**time_string**) and a format string (**"%d %b %Y %H:%M:%S"**), and it returns a **datetime** object.

5. The format string **"%d %b %Y %H:%M:%S"** tells Python how to interpret the **time_string**:
   - **%d** stands for the day of the month as a zero-padded decimal number (e.g., **01** to **31**).
   - **%b** stands for the abbreviated month name (e.g., **Jan**, **Feb**, etc.).
   - **%Y** stands for the year with century as a decimal number (e.g., **2023**).
   - **%H** stands for the hour (24-hour clock) as a zero-padded decimal number (e.g., **00** to **23**).
   - **%M** stands for the minute as a zero-padded decimal number (e.g., **00** to **59**).
   - **%S** stands for the second as a zero-padded decimal number (e.g., **00** to **59**).

Problem 6



```python
# Problem 6
# Step 2: Define the BeforeTrigger class
class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        return story['published'] < self.time

# Step 3: Define the AfterTrigger class
class AfterTrigger(TimeTrigger):
    def evaluate(self, story):
        return story['published'] > self.time
```

- Both BeforeTrigger and AfterTrigger are designed to be used with a dictionary (or similar structure) representing a story, which contains a published key holding a datetime object.
- These classes allow you to create triggers based on time, where:
  - **BeforeTrigger** checks if the story was published before a specific time.
  - **AfterTrigger** checks if the story was published after a specific time

What is BeforeTrigger?
- **BeforeTrigger** is a class that inherits from the **TimeTrigger** class.
- Its purpose is to check if a given story was published before a specified time.

Components of BeforeTrigger:
1. **Class Definition**:

Define a new class **BeforeTrigger** that extends the functionality of the **TimeTrigger** class. This means **BeforeTrigger** inherits all the properties and methods of **TimeTrigger**.

2. **The evaluate Method**:

   - This method takes one parameter, **story**, which is expected to be a dictionary containing information about a story.
   - It compares the **published** time of the story (assumed to be a **datetime** object) with the **self.time** attribute (the time defined when the **BeforeTrigger** object was created).
   - The method returns **True** if the story's published time is before **self.time**, and **False** otherwise.

Step 3: Define the AfterTrigger Class

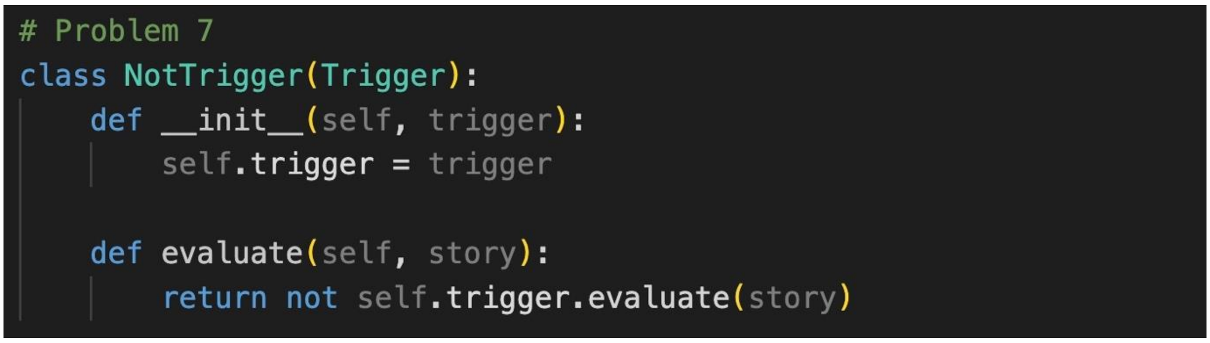
What is AfterTrigger?

- **AfterTrigger** is another class that also inherits from the **TimeTrigger** class.
- Its purpose is to check if a given story was published after a specified time.

Components of AfterTrigger:

1. **Class Definition**:
   A new class **AfterTrigger** is defined that extends the functionality of the **TimeTrigger** class. It inherits all the properties and methods of **TimeTrigger**.
2. **The evaluate Method**:
   - This method is similar to the one in **BeforeTrigger**, but it checks if the story's **published** time is after **self.time**.
   - It returns **True** if the story's published time is after **self.time**, and **False** otherwise.

Problem 7



```python
# Problem 7
class NotTrigger(Trigger):
    def __init__(self, trigger):
        self.trigger = trigger

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

What is NotTrigger?

- **NotTrigger** is a class that inherits from the **Trigger** class.
- It is designed to invert the evaluation of another trigger. This means if the original trigger evaluates to **True**, **NotTrigger** will evaluate to **False**, and vice versa.

Components of NotTrigger:

1. **Class Definition**:

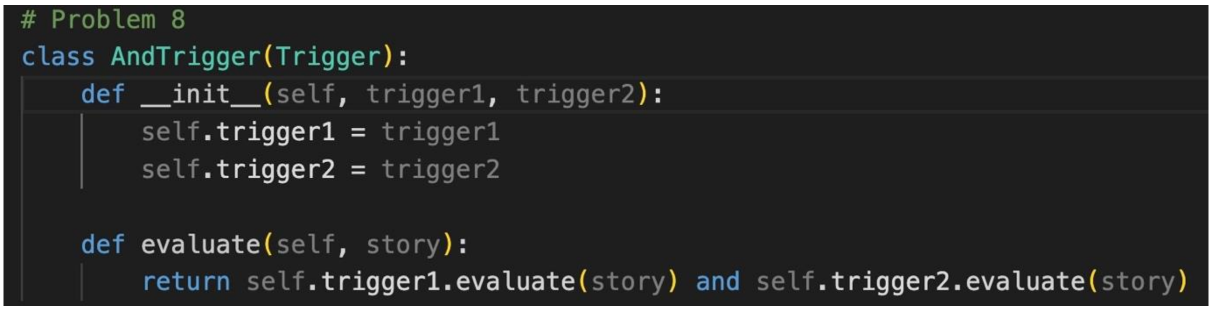
   A new class **NotTrigger** is defined that extends the functionality of the **Trigger** class. This means **NotTrigger** inherits all the properties and methods of **Trigger**.

2. **Constructor (__init__ Method)**:
   - This method is called when a new instance of **NotTrigger** is created.
   - It takes one parameter in addition to **self**: **trigger**. This is an instance of another trigger class (such as **BeforeTrigger**, **AfterTrigger**, or any other class that inherits from **Trigger**).
   - **self.trigger = trigger** stores the provided trigger instance in the **self.trigger** attribute for later use.

3. **The evaluate Method**:

- This method takes one parameter, **story**, which is expected to be a dictionary containing information about a story.
- It calls the **evaluate** method of the stored trigger (**self.trigger.evaluate(story)**) and negates the result using the **not** operator.
- If **self.trigger.evaluate(story)** returns **True**, **not self.trigger.evaluate(story)** will return **False**.
- If **self.trigger.evaluate(story)** returns **False**, **not self.trigger.evaluate(story)** will return **True**.

Problem 8



```python
# Problem 8
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)
```

What is AndTrigger?

- **AndTrigger** is a class that inherits from the **Trigger** class.
- It is designed to combine two other triggers using a logical AND operation. This means that **AndTrigger** will only evaluate to **True** if both of the combined triggers evaluate to **True**.
-

Components of AndTrigger:

1. **Class Definition**:

A new class **AndTrigger** is defined that extends the functionality of the **Trigger** class. This means **AndTrigger** inherits all the properties and methods of **Trigger**.

2. **Constructor (__init__ Method)**:

- This method is called when a new instance of **AndTrigger** is created.
- It takes two parameters in addition to **self**: **trigger1** and **trigger2**. These are instances of other trigger classes (such as **BeforeTrigger**, **AfterTrigger**, or any other class that inherits from **Trigger**).
- **self.trigger1 = trigger1** and **self.trigger2 = trigger2** store the provided trigger instances in the **self.trigger1** and **self.trigger2** attributes for later use.
-

3. **The evaluate Method**:

- This method takes one parameter, **story**, which is expected to be a dictionary containing information about a story.
- It calls the **evaluate** method of both stored triggers (**self.trigger1.evaluate(story)** and **self.trigger2.evaluate(story)**) and combines their results using the **and** operator.
- The method returns **True** only if both **self.trigger1.evaluate(story)** and **self.trigger2.evaluate(story)** return **True**.
- If either of the evaluations returns **False**, the method returns **False**.

PROBLEMS 9-11

Muhammad Abdullah Qureshi

Problem 9: OrTrigger Class

```
class OrTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1  # First trigger
        self.trigger2 = trigger2  # Second trigger

    def evaluate(self, story):
        # Return True if either of the triggers evaluate to True
        return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)
```

Implementation: The **OrTrigger** class is designed to combine two other triggers and fire if either of the component triggers fire. It achieves this by implementing the **evaluate** method, which checks both triggers.

- **__init__** method: Initializes the **OrTrigger** with two triggers (**trigger1** and **trigger2**).

- **evaluate** method: Evaluates both triggers on the given story. If either trigger evaluates to **True**, the method returns **True**.

Problem 10: filter_stories Function

```
def filter_stories(stories, triggers):
    filtered_stories = []  # List to hold stories that match the triggers
    for story in stories:
        for trigger in triggers:
            if trigger.evaluate(story):  # Check if the trigger fires for the story
                filtered_stories.append(story)
                break  # Move to the next story after the first matching trigger
    return filtered_stories  # Return the list of filtered stories
```

The **filter_stories** function iterates through the provided list of news stories and triggers. It checks each story against all the triggers and includes the story in the result list if any trigger fires.

- The function takes two arguments: **stories** (a list of **NewsStory** objects) and **triggers** (a list of trigger objects).
- It initializes an empty list **filtered_stories** to store stories that match any trigger.
- It iterates over each story and each trigger, and if a trigger fires (**evaluate** returns **True**), it adds the story to **filtered_stories** and breaks out of the inner loop to check the next story.

Problem 11: User-Specified Triggers

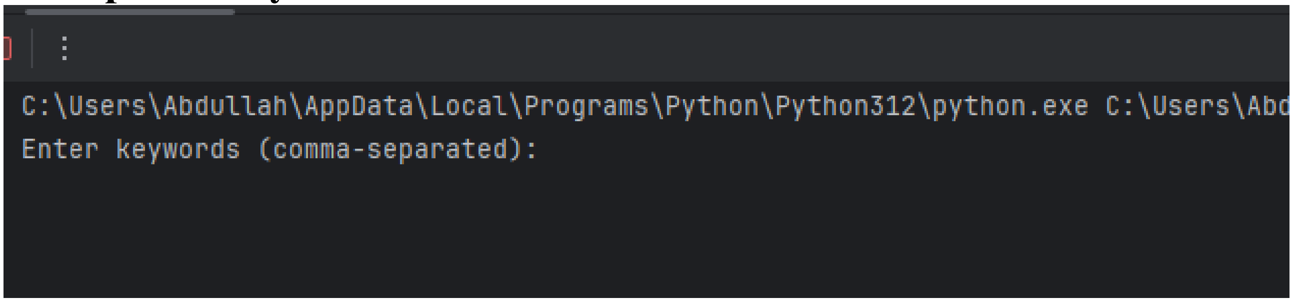
```
def read_trigger_config(filename):
    trigger_map = {
        'TITLE': TitleTrigger,
        'DESCRIPTION': DescriptionTrigger,
        'BEFORE': BeforeTrigger,
        'AFTER': AfterTrigger,
        'NOT': NotTrigger,
        'AND': AndTrigger,
        'OR': OrTrigger
    }
    triggers = []
    trigger_objects = {}

    with open(filename, 'r') as f:
        for line in f:
            parts = line.strip().split(',')
            if parts[0] == 'ADD':
                triggers.extend([trigger_objects[part] for part in parts[1:]])
            else:
                trigger_name = parts[0]
                trigger_type = parts[1]
                args = parts[2:]

                if trigger_type in ['AND', 'OR']:
                    trigger_objects[trigger_name] = trigger_map[trigger_type](
                        trigger_objects[args[0]],

                        trigger_objects[args[1]]
                    )
                else:
                    trigger_objects[trigger_name] = trigger_map[trigger_type](*args)

    return triggers
```

The **read_trigger_config** function reads a configuration file that defines which triggers to create and how to combine them. This function dynamically creates trigger instances based on the file's contents.

- **trigger_map**: A dictionary mapping trigger type names to their corresponding classes.
- **triggers**: A list to store the final set of triggers to be used.
- **trigger_objects**: A dictionary to store instances of created triggers by their names.
- The function opens and reads the **filename** line by line. Each line defines a trigger or adds triggers to the main list:
    - Lines starting with **ADD** specify which triggers to add to the main **triggers** list.
    - Other lines define individual triggers with their type and arguments.
    - For **AND** and **OR** triggers, it creates compound triggers using already created triggers.
    - For other triggers, it directly creates instances using the provided arguments.

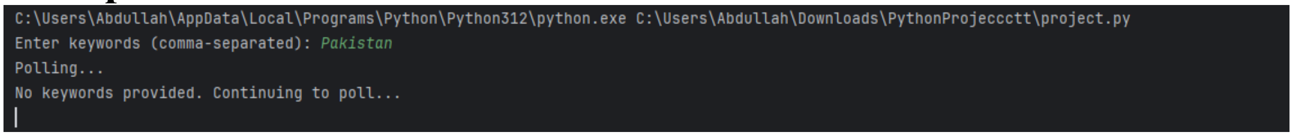
WORKING OF CODE:

Prompt for Keyword:



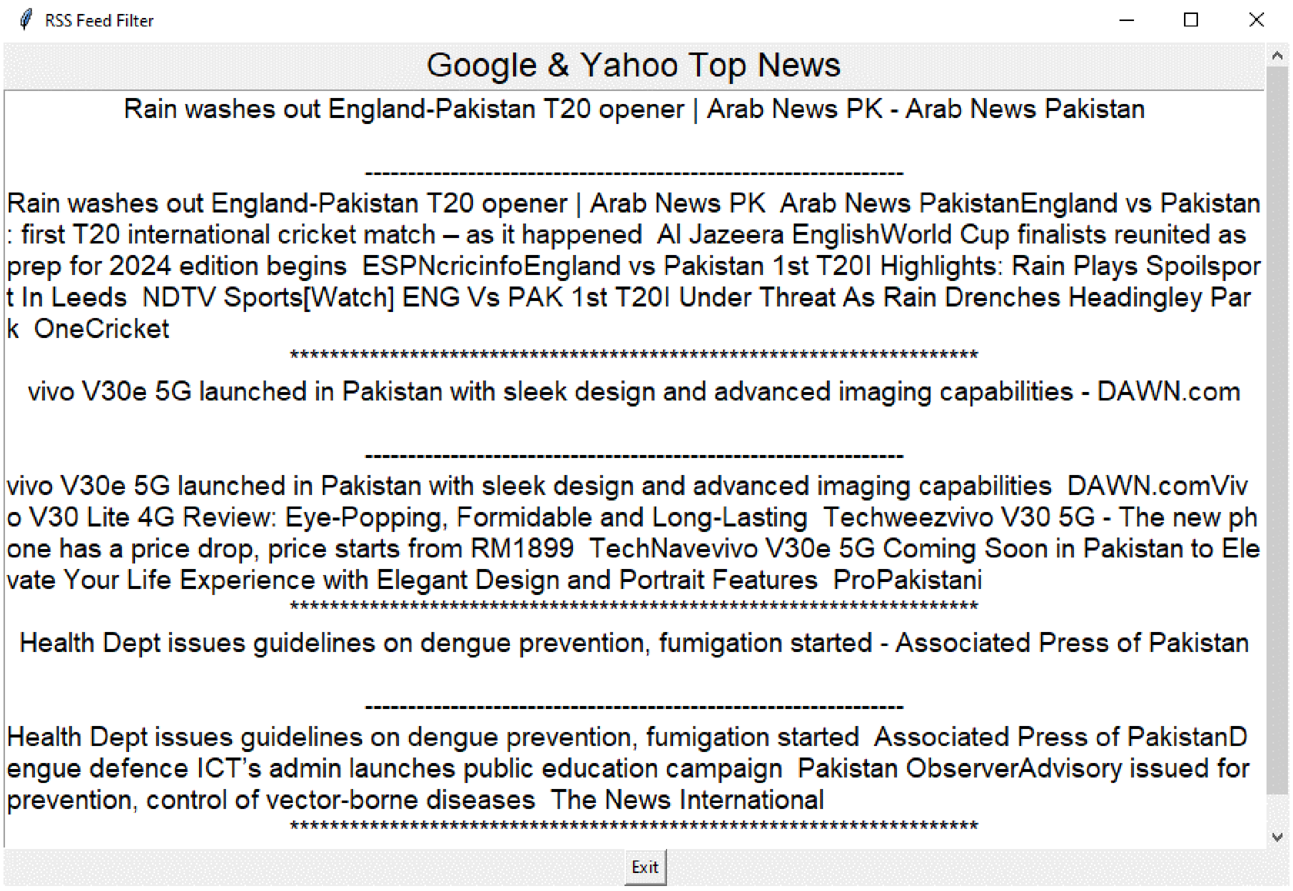
```
C:\Users\Abdullah\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Abd
Enter keywords (comma-separated):
```

Prompt Entered:



```
C:\Users\Abdullah\AppData\Local\Programs\Python\Python312\python.exe C:\Users\Abdullah\Downloads\PythonProjeccctt\project.py
Enter keywords (comma-separated): Pakistan
Polling...
No keywords provided. Continuing to poll...
```

Results:



```
RSS Feed Filter

Google & Yahoo Top News

Rain washes out England-Pakistan T20 opener | Arab News PK - Arab News Pakistan

--------------------------------------------------------------
Rain washes out England-Pakistan T20 opener | Arab News PK  Arab News PakistanEngland vs Pakistan
: first T20 international cricket match – as it happened  Al Jazeera EnglishWorld Cup finalists reunited as
prep for 2024 edition begins  ESPNcricinfoEngland vs Pakistan 1st T20I Highlights: Rain Plays Spoilspor
t In Leeds  NDTV Sports[Watch] ENG Vs PAK 1st T20I Under Threat As Rain Drenches Headingley Par
k  OneCricket
********************************************************************************
vivo V30e 5G launched in Pakistan with sleek design and advanced imaging capabilities - DAWN.com

--------------------------------------------------------------
vivo V30e 5G launched in Pakistan with sleek design and advanced imaging capabilities  DAWN.comViv
o V30 Lite 4G Review: Eye-Popping, Formidable and Long-Lasting  Techweezvivo V30 5G - The new ph
one has a price drop, price starts from RM1899  TechNavevivo V30e 5G Coming Soon in Pakistan to Ele
vate Your Life Experience with Elegant Design and Portrait Features  ProPakistani
********************************************************************************
Health Dept issues guidelines on dengue prevention, fumigation started - Associated Press of Pakistan

--------------------------------------------------------------
Health Dept issues guidelines on dengue prevention, fumigation started  Associated Press of PakistanD
engue defence ICT's admin launches public education campaign  Pakistan ObserverAdvisory issued for
prevention, control of vector-borne diseases  The News International
********************************************************************************
Exit
```