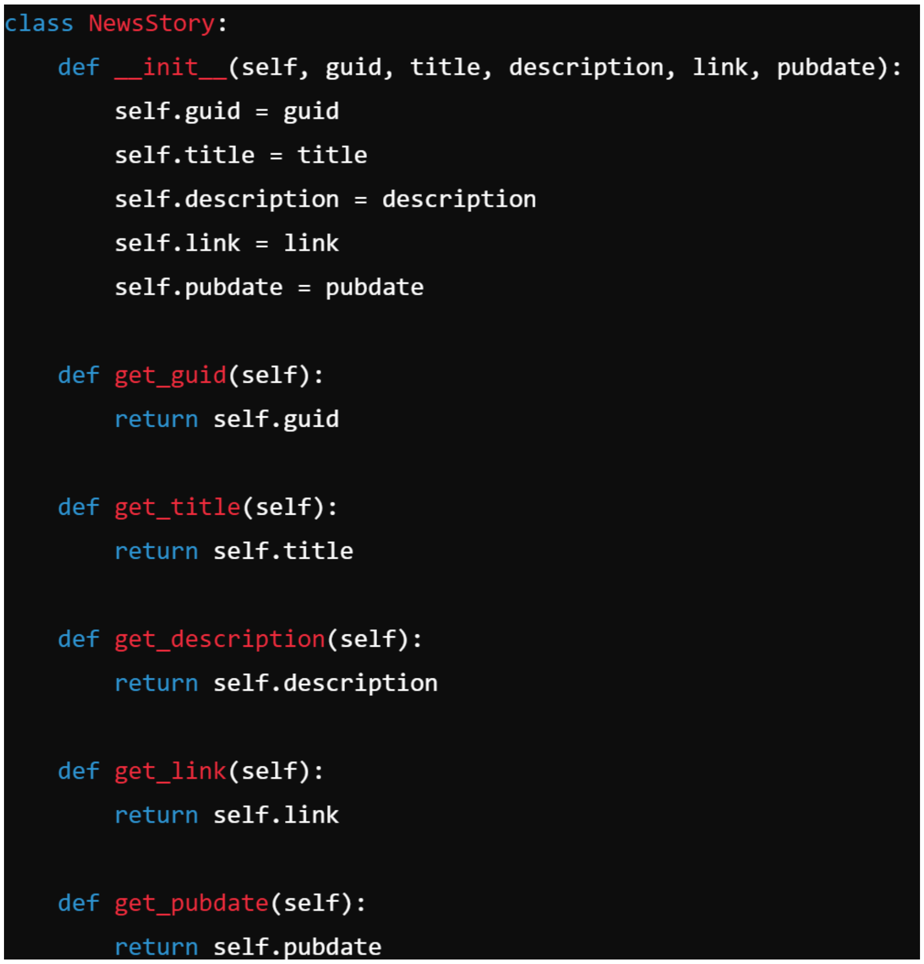
All Problem Solutions

Problem 1:

Implement the **NewsStory** class with attributes such as **guid, title,** description, link, and **pubdate**. This class represents a news story with methods to access each attribute individually.

Solution:

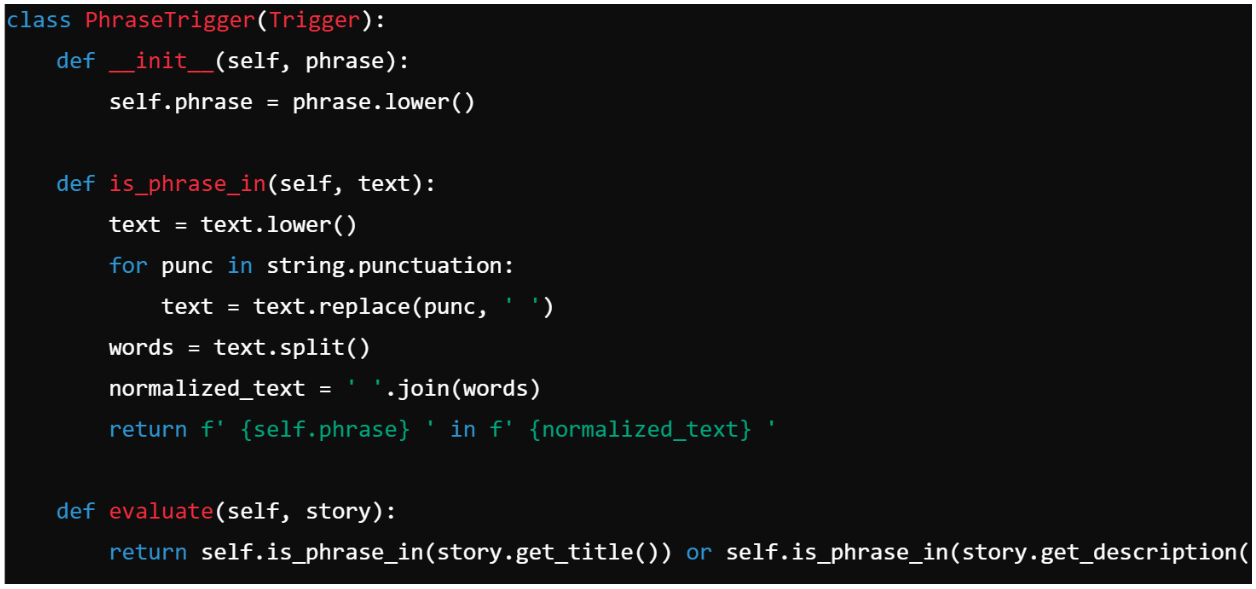


```python
class NewsStory:
    def __init__(self, guid, title, description, link, pubdate):
        self.guid = guid
        self.title = title
        self.description = description
        self.link = link
        self.pubdate = pubdate

    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
```

Problem 2:

Implement the **PhraseTrigger** class, a subclass of **Trigger**, which evaluates whether a given phrase is present in the title or description of a news story.

Solution:

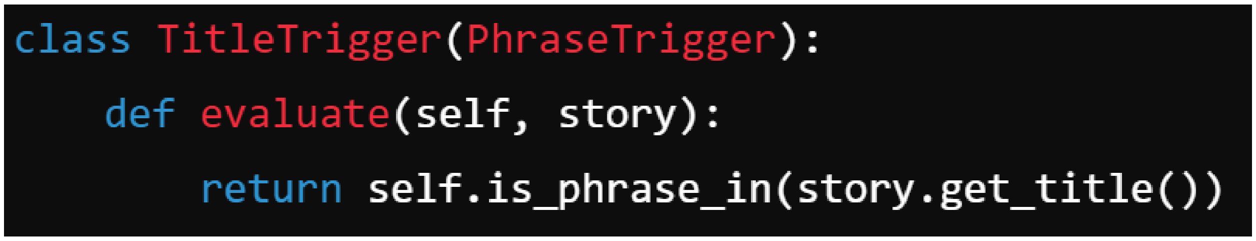


```python
class PhraseTrigger(Trigger):
    def __init__(self, phrase):
        self.phrase = phrase.lower()

    def is_phrase_in(self, text):
        text = text.lower()
        for punc in string.punctuation:
            text = text.replace(punc, ' ')
        words = text.split()
        normalized_text = ' '.join(words)
        return f' {self.phrase} ' in f' {normalized_text} '

    def evaluate(self, story):
        return self.is_phrase_in(story.get_title()) or self.is_phrase_in(story.get_description(
```

Problem 3:

Implement the **TitleTrigger** class, a subclass of **PhraseTrigger,** which evaluates whether a given phrase is present in the title of a news story.

Solution:



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

Problem 4:

Implement the **DescriptionTrigger** class, a subclass of **PhraseTrigger,** which evaluates whether a given phrase is present in the description of a news story.

Solution:

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

Problem 5:

Implement the **TimeTrigger** class, a subclass of Trigger, which evaluates whether a news story's publication date falls before or after a specified time.

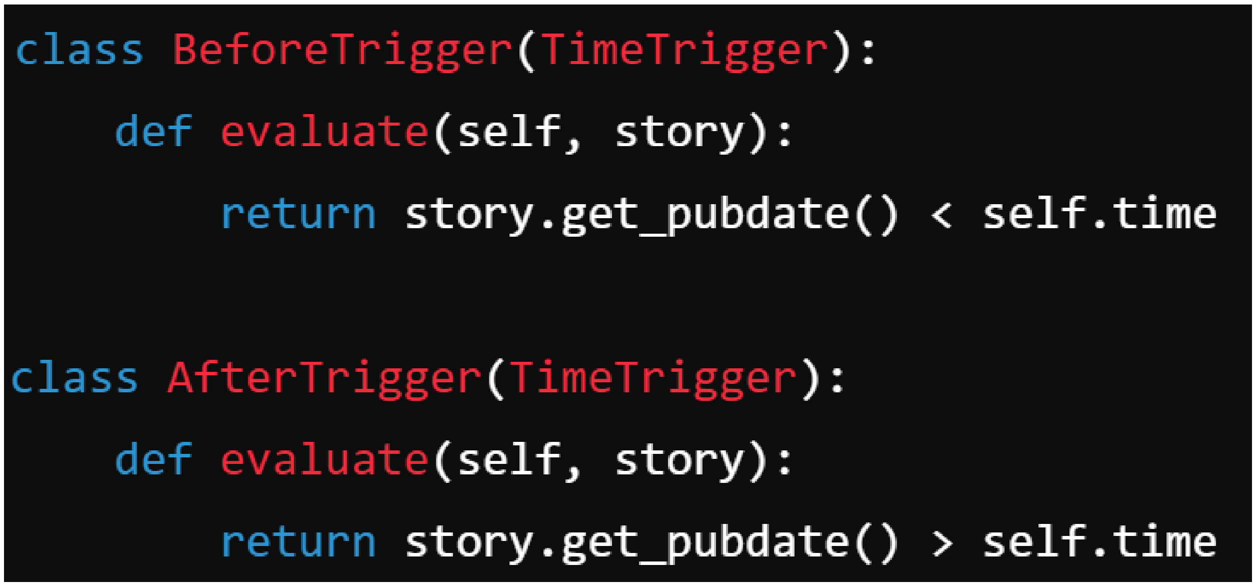
Solution:

```python
class TimeTrigger(Trigger):
    def __init__(self, time_string):
        self.time = datetime.strptime(time_string, "%d %b %Y %H:%M:%S")
        self.time = self.time.replace(tzinfo=pytz.utc)
        self.time = self.time.astimezone(pytz.timezone("US/Eastern"))

    def evaluate(self, story):
        raise NotImplementedError("This method should be implemented in subclasses.")
```

Problem 6:

Implement the **BeforeTrigger** and **AfterTrigger** classes, subclasses of **TimeTrigger,** which evaluate whether a news story's publication date falls before or after a specified time, respectively.

Solution:

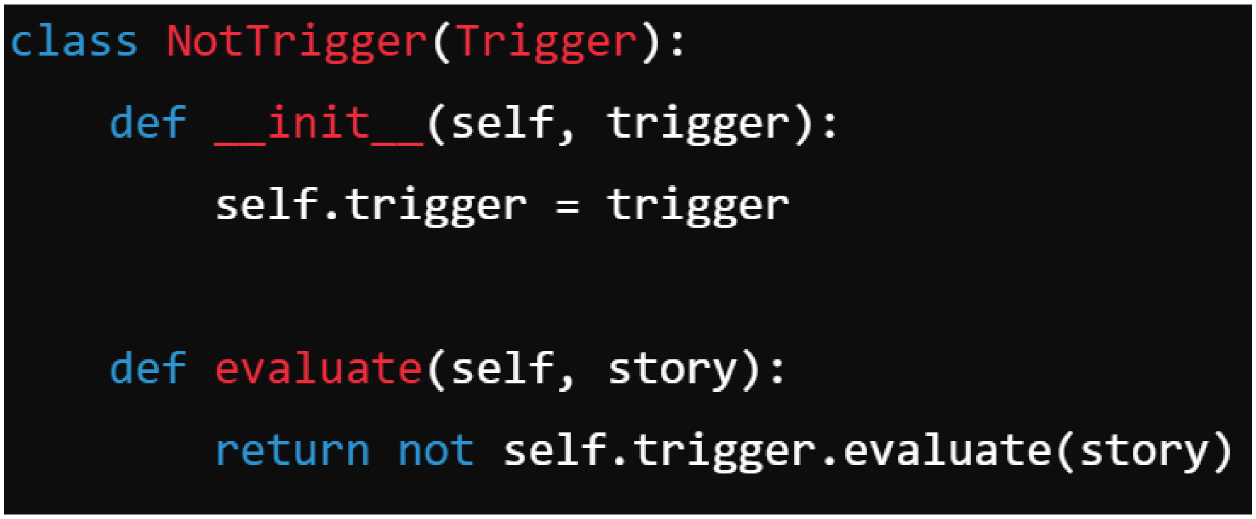


```python
class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        return story.get_pubdate() < self.time

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

Problem 7:

Implement the **NotTrigger** class, a subclass of **Trigger**, which negates the evaluation of another trigger.

Solution:



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

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

Problem 8:

Implement the **AndTrigger** class, a subclass of **Trigger**, which evaluates to true if both of its constituent triggers evaluate to true.

Solution:

```
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)
```

Problem 9:

Implement the **OrTrigger** class, a subclass of **Trigger, ** which evaluates to true if either of its constituent triggers evaluate to true.

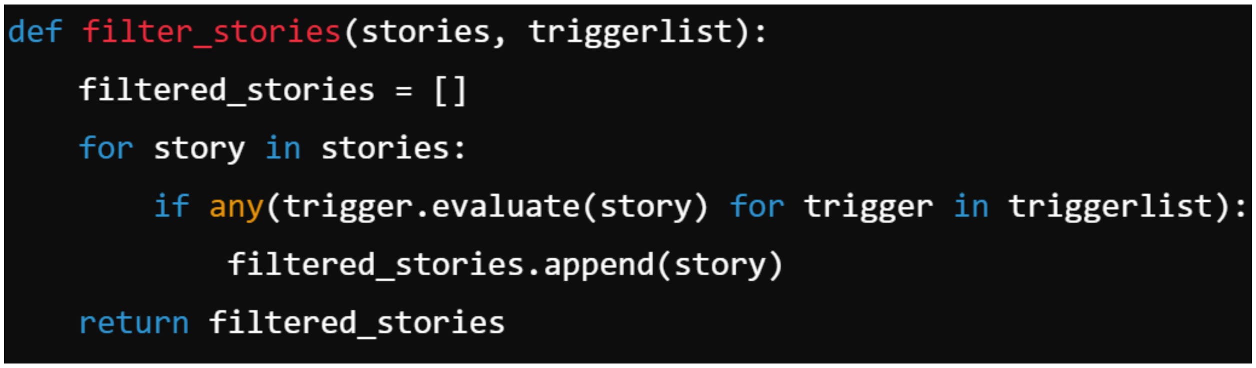
Solution:

```
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)
```

Problem 10:

Implement the **filter_stories** function, which filters a list of news stories based on a list of triggers.

Solution:



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

Problem 11:

Implement the **read_trigger_config** function, which reads a trigger configuration file and returns a list of trigger objects specified by the file.

Solution:

lines = {line.strip() for line in triiger_file if line.strip() and not line.strip()startswith(“//”)]

```python
def read_trigger_config(triggers):
    try:
        trigger_file = open(triggers, 'r')
    except FileNotFoundError:
        print(f"Error: The file {triggers} was not found.")
        return []

    lines = [line.strip() for line in trigger_file if line.strip() and not line.strip().startsw
    trigger_list = []

    for line in lines:
        parts = line.split(',')
        trigger_name = parts[0].strip()
        trigger_type = parts[1].strip()
        trigger_args = [arg.strip() for arg in parts[2:]]

        if trigger_type == 'TITLE':
            trigger = TitleTrigger(*trigger_args)
        elif trigger_type == 'DESCRIPTION':
            trigger = DescriptionTrigger(*trigger_args)
        elif trigger_type == 'AFTER':
            trigger = AfterTrigger(*trigger_args)
        elif trigger_type == 'BEFORE':
            trigger = BeforeTrigger(*trigger_args)
```

```python
        elif trigger_type == 'NOT':
            trigger = NotTrigger(triggers[trigger_args[0]])
        elif trigger_type == 'AND':
            trigger = AndTrigger(triggers[trigger_args[0]], triggers[trigger_args[1]])
        elif trigger_type == 'OR':
            trigger = OrTrigger(triggers[trigger_args[0]], triggers[trigger_args[1]])

        trigger_list.append(trigger)

    return trigger_list
```