NATIONAL UNIVERSITY OF SCIENCES AND TEACHNOLOGY

FOP II

PROJECT

Lab Instructor: Dr. Saqib Nazeer

Student names:

Sayed Wasi Hyder Shah (454424)

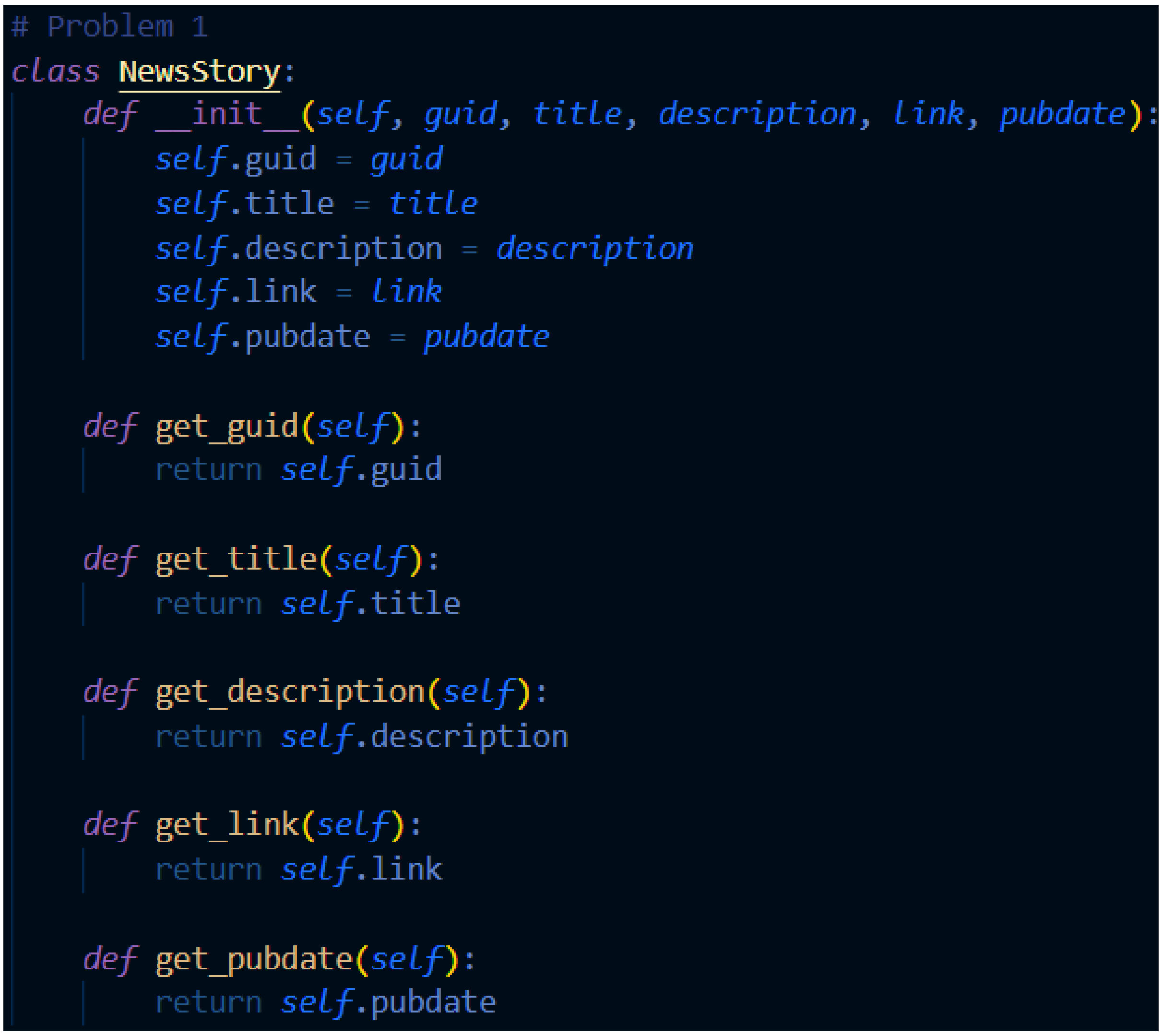
Haniyyah Abbas (481755)

Muhammad Talha Kashif (454135)

In this project, we were to use our knowledge of Object Oriented Programming to create a program that filters through news feeds all over the internet. It will alert the user of any new headline or story that fits their interests.

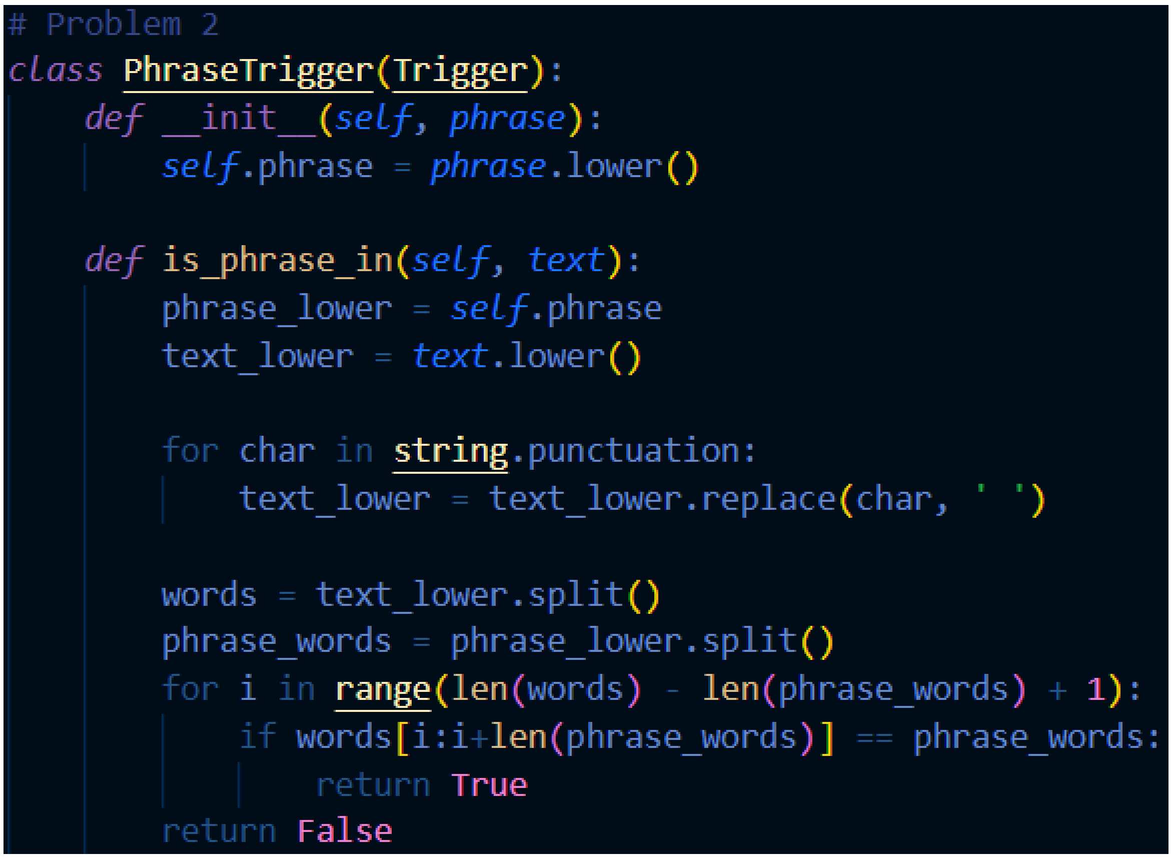
PROBLEMS 1-4

Sayed Wasi Hyder Shah



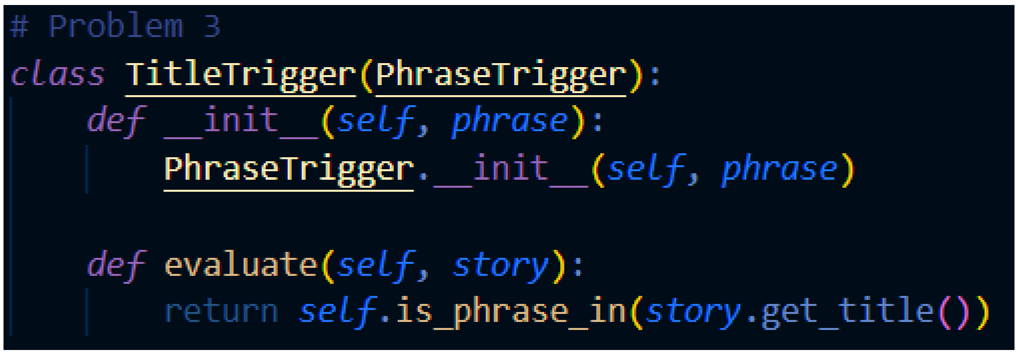
```python
# Problem 1
class NewsStory:
    def __init__(self, guid, title, description, link, pubdate):
        self.guid = guid
        self.title = title
        self.description = description
        self.link = link
        self.pubdate = pubdate

    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
```

In problem one, 'NewsStory' was written. This basically includes the information you need for every news story. It contains attributes like 'guid,', 'description', 'link' etc. 'guid' is an identifier for the story, 'title' is the title of the story, 'description' is an explanation for the story, 'link' is a link to the story, and finally 'pubdate' is the publication date of the news story. Below the mentioned attributes are the methods of the class. This helps define the behavior of the attributes that were created in the class. It is an action that the attribute will perform. It gives read-only access to the story's details.



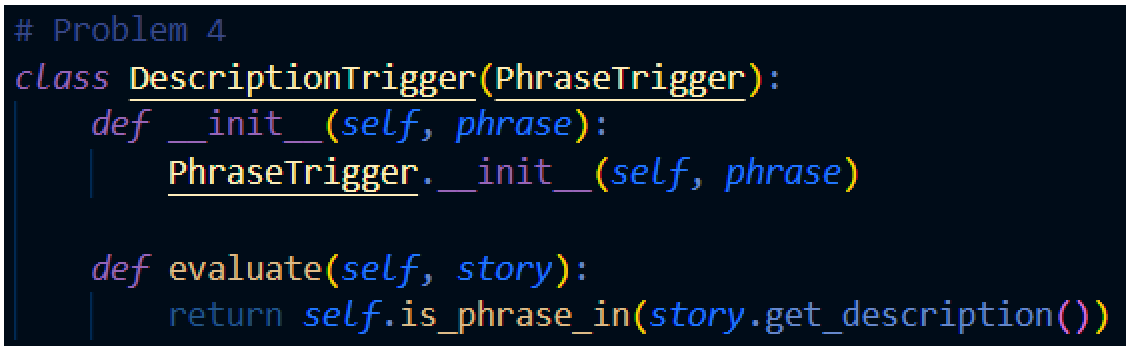
```python
# Problem 2
class PhraseTrigger(Trigger):
    def __init__(self, phrase):
        self.phrase = phrase.lower()

    def is_phrase_in(self, text):
        phrase_lower = self.phrase
        text_lower = text.lower()

        for char in string.punctuation:
            text_lower = text_lower.replace(char, ' ')

        words = text_lower.split()
        phrase_words = phrase_lower.split()
        for i in range(len(words) - len(phrase_words) + 1):
            if words[i:i+len(phrase_words)] == phrase_words:
                return True
        return False
```

In problem two, we needed to create a ‘PhaseTrigger’ class. This code was to check if a specific trigger would check a certain word given in a text or statement. ‘Phrase’ is meant to look for the phrase in said statement. The lower() function is used to ensure a case-insensitive comparison. The method in this code ‘is_phrase_in’ basically converts the phrase to all lowercase, removes punctuation from the statements and replaces it with spaces, splits the text into words, and checks if that exact sequence of words appears in the statement.



```python
# Problem 3
class TitleTrigger(PhraseTrigger):
    def __init__(self, phrase):
        PhraseTrigger.__init__(self, phrase)

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

In problem three, a ‘TitleTrigger’ class was created. Its purpose was to check if a specific phrase is found in the title of the story. It only works through the ‘PhraseTrigger’ class. There is an evaluate method which calls the ‘in_phrase_in’ method from the previous class. It returns true only if the phrase is found, otherwise it returns false.

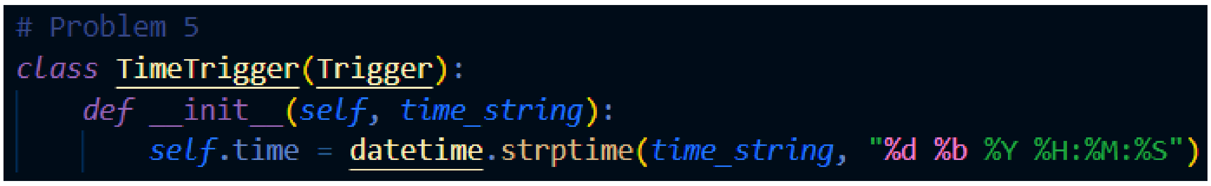


```python
# Problem 4
class DescriptionTrigger(PhraseTrigger):
    def __init__(self, phrase):
        PhraseTrigger.__init__(self, phrase)

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

In problem four, we wrote the ‘DescriptionTrigger’ class. This also works through ‘PhraseTrigger’ and its purpose is to find a specific phrase from the description of the story. There is an evaluate method again in this class, which calls the ‘in_phrase_in’ method from ‘PhraseTrigger’ to check the description of a story. It returns true if the description is found, otherwise it reads false.

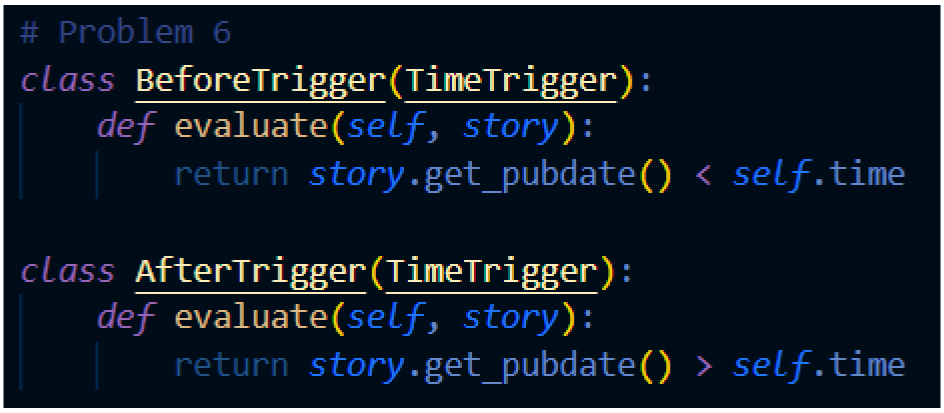
PROBLEMS 5-8

Haniyyah Abbas



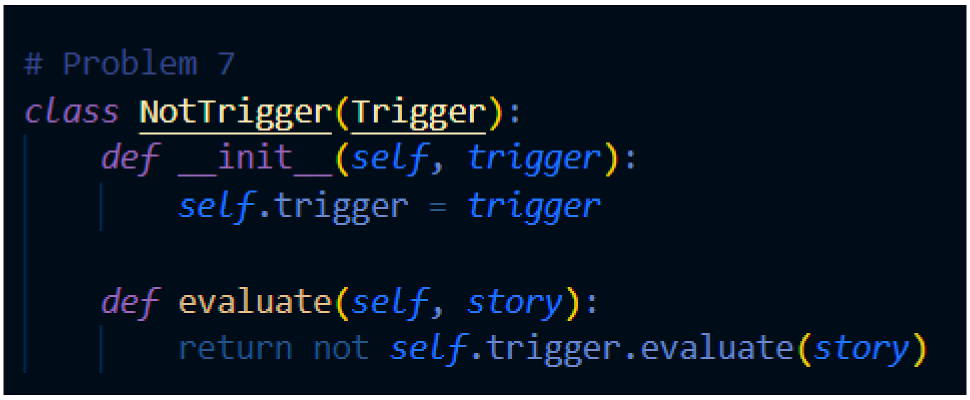
```python
# Problem 5
class TimeTrigger(Trigger):
    def __init__(self, time_string):
        self.time = datetime.strptime(time_string, "%d %b %Y %H:%M:%S")
```

In problem five, the ‘TimeTrigger’ base class was created. This is a base class from the time-based triggers. A base class provides behaviors and attributes to the other classes, such as ‘BeforeTrigger’ or ‘AfterTrigger’. A ‘datetime’ object that represents a specific time is created by parsing a time string. The ‘__init__’ method takes a ‘time_string” parameter. It converts that string to a ‘datetime’ object and uses the ‘strptime’ method from the ‘datetime’ module. The format of the ‘time_string’ is "%d %b %Y %H:%M:%S" which means day, month, year, hour, minute, and second.



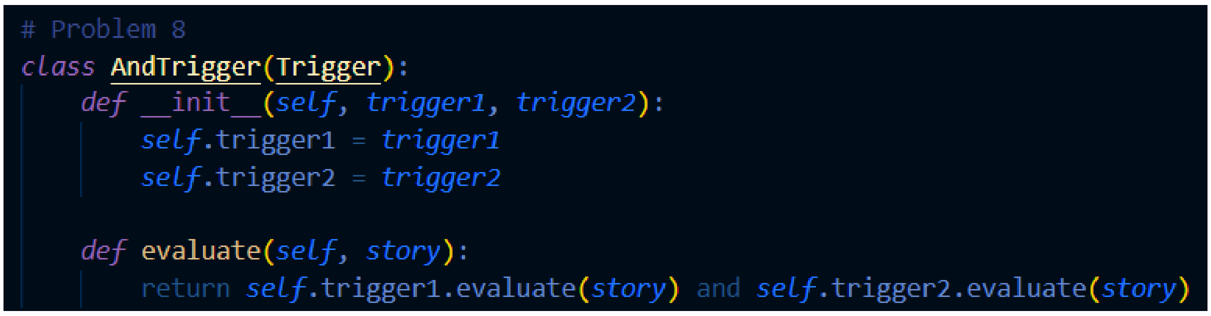
```python
# Problem 6
class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        return story.get_pubdate() < self.time

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

In problem six, the ‘BeforeTrigger’ and ‘AfterTrigger’ classes were written. These two classes depend upon the ‘TimeTrigger’ class. There is an evaluate method that checks if a story was published before a specific time. It takes ‘story’ as an argument, and then compares the publication of that story with the ‘self.time’ attribute. In the ‘AfterTrigger’ class, a method is implemented to check if a story was published after a certain time. Once again, it takes ‘story’, compares the date of the story with the ‘self.time’ attribute and returns true if all is well. Otherwise, it comes out as false.



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

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

The ‘NotTrigger’ class is a logical NOT operator for triggers. It wraps another trigger and inverts its result. This is useful when you need to filter out stories that match a certain condition and only keep those that do not match.

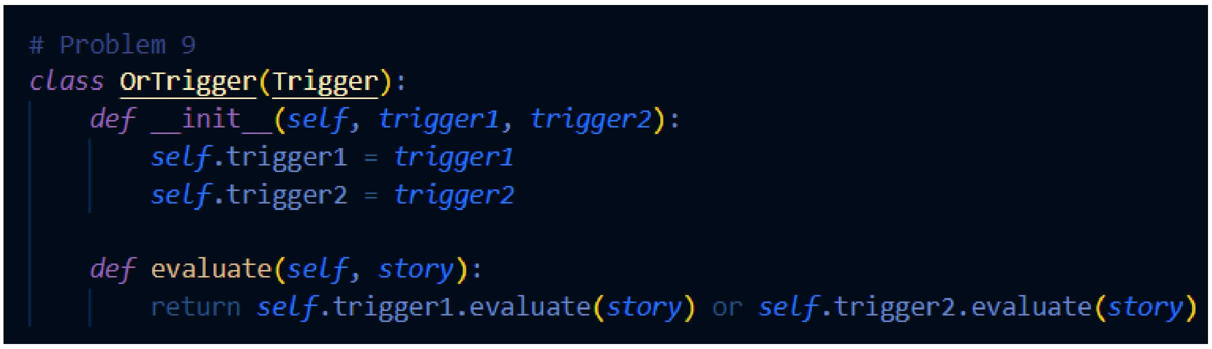


```python
# Problem 8
class AndTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

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

The ‘AndTrigger’ class is a logical AND operator for triggers. It combines two other triggers and only fires when both of the combined triggers fire. This is useful when you need to filter stories based on multiple conditions that must all be true simultaneously.

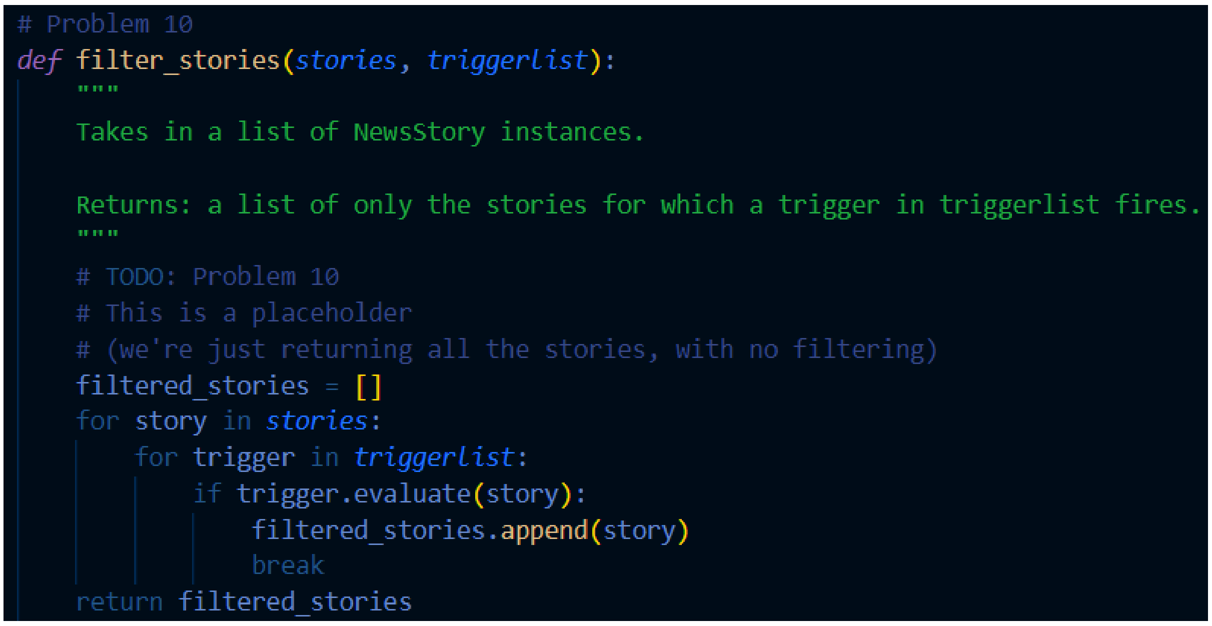
PROBLEMS 9-11

Muhammad Talha Kashif



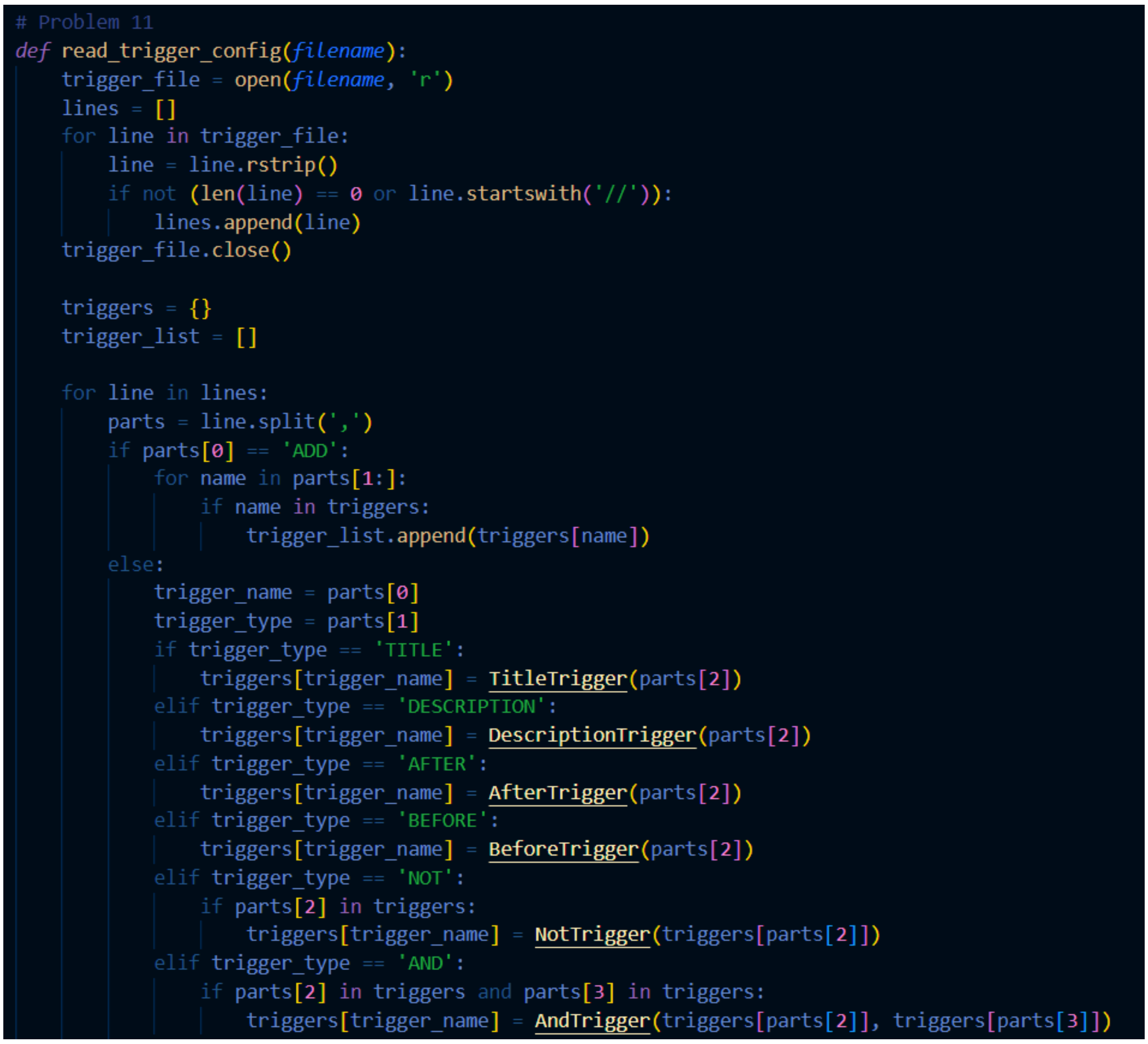
```python
# Problem 9
class OrTrigger(Trigger):
    def __init__(self, trigger1, trigger2):
        self.trigger1 = trigger1
        self.trigger2 = trigger2

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

The ‘OrTrigger’ class is a logical OR operator for triggers. It combines two other triggers and fires if at least one of the combined triggers fires. This is useful when you need to filter stories based on multiple conditions where any one condition being true is sufficient.

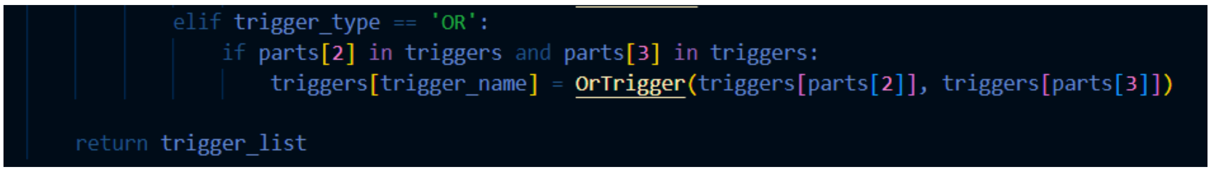


```python
# Problem 10
def filter_stories(stories, triggerlist):
    """
    Takes in a list of NewsStory instances.

    Returns: a list of only the stories for which a trigger in triggerlist fires.
    """
    # TODO: Problem 10
    # This is a placeholder
    # (we're just returning all the stories, with no filtering)
    filtered_stories = []
    for story in stories:
        for trigger in triggerlist:
            if trigger.evaluate(story):
                filtered_stories.append(story)
                break
    return filtered_stories
```

The ‘filter_stories’ function filters a list of news stories, returning only those for which at least one trigger in the provided trigger list fires. It ensures that each story is evaluated against all triggers, but as soon as one trigger fires for a story, that story is added to the results, and the function moves on to the next story.



```python
# Problem 11
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]
            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]])
```



```python
            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
```

- read_trigger_config: Reads a configuration file to create and return a list of triggers.
- main_thread: Sets up a GUI, continuously polls for news stories, filters them based on triggers, and displays the relevant stories.

These functions work together to create a system that monitors news feeds, applies user-defined triggers to filter stories, and presents the filtered stories in a user-friendly manner.