
# *NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY*

*SCHOOL OF MECHANICAL AND MANUFACTURING ENGINEERING*

*SEMESTER # 02* *CLASS: - ME-15* [*SEC A]*

## END SEMESTER PROJECT

**FUNDAMENTALS OF PROGG II**

DATE OF SUBMISSION: 22 May, 2024

SUBMITTED TO: Dr. Saqib Nazir

**Submitted by:**

**Talha Mateen**

**Cms:**

**454713**

| GROUP MEMBERS | | |
|---|---|---|
| Serial NO. | NAME | ROLL NO. |
| 01 | ABDULLAH JAMIL TUNG | 478459 |
| 02 | KASHIF NADEEM KAYANI | 456466 |
| 03 | SYED ASAD RAZA ZAIDI | 464239 |
| 04 | TALHA MATEEN | 454713 |

## 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
```

### Class Definition:

Defines the NewsStory class to store information about a news story.

**Constructor Method (__init__):**

Initializes instances with attributes: guid, title, description, link, and pubdate. Parameters used are guid, title, description, link, pubdate. Assigns values passed as arguments to the corresponding attributes.

**Getter Methods:**

Provides methods to access attribute values externally. Getter methods: get_guid(), get_title(), get_description(), get_link(), get_pubdate(). Returns the value of the respective attribute.

**Underscore Prefix in Method Names**

Method names with single underscores are used, possibly as typos (should be double underscores). Correcting these underscores ensures consistency with Python conventions.

## Problem 2

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

    def is_phrase_in(self, text):
        text = text.lower()
        for p in string.punctuation:
```

```
            text = text.replace(p, ' ')
        words = text.split()
        phrase_words = self.phrase.split()
        for i in range(len(words) - len(phrase_words) + 1):
            if words[i:i + len(phrase_words)] == phrase_words:
                return True
        return False
```

### Class Definition:

Defines the PhraseTrigger class to trigger alerts based on specific phrases in news stories. Inherits from the Trigger class and ABC (Abstract Base Class) to define an abstract base class.

**Constructor Method (__init__):**

Initializes instances with a phrase attribute storing the phrase to search for, converted to lowercase. Parameters: phrase (the phrase to search for).

**is_phrase_in Method:**

Checks if the stored phrase appears in a given text.

Parameters: text (text to search for the phrase).

**Steps:**

Converts the text to lowercase.

Replaces punctuation with spaces to separate words.

Splits text into words and the phrase into words.

Iterates through text segments to find an exact match with the phrase.

Returns True if a match is found, else False.

**evaluate Method:**

Abstract method declared with @abstractmethod decorator. Indicates that subclasses must implement their specific evaluation logic. Suggests that PhraseTrigger serves as a base class for subclasses to define custom evaluation methods.

## Problem 3

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

### Class Definition

Defines the TitleTrigger class as a subclass of PhraseTrigger.

**evaluate Method**

Overrides the evaluate method inherited from PhraseTrigger. Takes a story parameter representing a news story object.

**Inside evaluate**

Retrieves the title of the news story using get_title.

Calls the is_phrase_in method (inherited from PhraseTrigger) with the story title.

Returns True if the phrase is found in the title, else False.

**Usage of Inheritance**

Inherits functionality from PhraseTrigger, such as the is_phrase_in method.

Promotes code reuse and maintains a clear hierarchy, specializing in evaluating titles of news stories.

## Problem 4

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

### Class Definition

Declares the DescriptionTrigger class as a subclass of PhraseTrigger.

**evaluate Method**

Overrides the evaluate method inherited from PhraseTrigger. Takes a story parameter representing a news story object.

Inside evaluate:

Retrieves the description of the news story using get_description. Calls the is_phrase_in method (inherited from PhraseTrigger) with the story description. Returns True if the phrase is found in the description, else False.

**Inheritance Relationship**

Inherits functionality from PhraseTrigger, such as the is_phrase_in method. Enables reuse of phrase-matching functionality, maintaining a clear hierarchy. Specializes in evaluating descriptions of news stories.

## Problem 5

```
class TimeTrigger(Trigger):
    def __init__(self, time):
        self.time = datetime.strptime(time, "%d %b %Y %H:%M:%S")
        self.time = self.time.replace(tzinfo=pytz.timezone("EST"))
```

### Class Definition

Declares the TimeTrigger class as a subclass of Trigger.

**TODO Placeholder**

Contains a pass statement, indicating incomplete implementation. No additional code is provided within the class definition.

**Constructor Documentation**

Describes expected constructor parameters and format for time input. Specifies the need to convert time to a datetime object in EST.

**Purpose of TimeTrigger**

Intended for triggering alerts based on specific times. Users can specify alert conditions based on news story publication times. Likely to include methods for evaluating if a story meets time-based trigger criteria.

**Problem 6**

```
class BeforeTrigger(TimeTrigger):
    def evaluate(self, story):
        pub_date = (story.get_pubdate()).replace(
            tzinfo=pytz.timezone("EST"))
        return pub_date < self.time

class AfterTrigger(TimeTrigger):
    def evaluate(self, story):
        pub_date = (story.get_pubdate()).replace(
            tzinfo=pytz.timezone("EST"))
        return pub_date > self.time
```

### Class Definitions

Introduces BeforeTrigger and AfterTrigger classes inheriting from TimeTrigger.

**Constructor**

Both classes have constructor initializing instances with a specified time. Time strings are converted to datetime objects using strptime.

**evaluate Method**

Determines if a news story satisfies trigger conditions. Compares story publication date with trigger time. BeforeTrigger returns True if story is published before trigger time. AfterTrigger returns True if story is published after trigger time.

**Purpose**

Facilitates trigger conditions based on temporal constraints. Enables specifying trigger conditions like pre- and post-publication times. Handles logic for evaluating temporal trigger conditions within the news alert system.

**Problem 7**

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

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

### Class Definition

Introduces the NotTrigger class inheriting from Trigger.

**Constructor**

Initializes instances with a trigger to be negated.

**evaluate Method**

Determines if a news story satisfies the trigger condition. Invokes the stored trigger's evaluate function and negates its result.

**Purpose**

Facilitates creation of composite triggers by negating other triggers' results. Allows for specifying trigger conditions based on the absence of certain criteria. Enhances the flexibility and expressiveness of the trigger system within the news alert application.

**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)
```

### Class Definition

Introduces the AndTrigger class inheriting from Trigger.

**Constructor**

Initializes instances with two triggers (t2 and t3) to be satisfied.

**evaluate Method**

Determines if a news story satisfies both trigger conditions. Invokes the evaluate method of both stored triggers and combines their results using the logical and operator.

**Purpose**

Facilitates creation of composite triggers requiring multiple conditions for activation. Enables specification of trigger conditions based on conjunction of multiple criteria. Enhances flexibility and expressiveness of the trigger system within the news alert application.

**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)
```

### Class Definition

Introduces the OrTrigger class inheriting from Trigger.

**Constructor**

Initializes instances with two triggers (t2 and t3) to be combined using logical disjunction.

**evaluate Method**

Determines if a news story satisfies at least one of the trigger conditions. Invokes the evaluate method of both stored triggers and combines their results using the logical or operator.

**Purpose**

Facilitates creation of composite triggers triggering if any of the specified conditions is met.

Enables specification of trigger conditions based on disjunction of multiple criteria.

Enhances flexibility and expressiveness of the trigger system within the news alert application.

## Problem 10

```
def filter_stories(stories, triggerlist):
    filtered_stories = []

    for story in stories:
            if isinstance(story, NewsStory):  # Check if story is a NewsStory object
                for trigger in triggerlist:
                    if trigger.evaluate(story):
                        filtered_stories.append(story)
                        break
            else:
                print("No Description")
```

### Function Definition

Introduces the filter_stories function to filter news stories based on triggers.

**Function Parameters**

Takes two parameters: stories (a list of NewsStory instances) and triggerlist (a list of trigger objects).

**Function Documentation**

Includes a docstring explaining the function's purpose and parameters.

**Filtering Logic**

Utilizes list comprehension to iterate over each story and trigger. Evaluates each trigger's evaluate method for the current story. Uses the any function to check if any trigger fires for a given story. Generates a filtered list containing stories for which at least one trigger fires.

**Return Statement**

Returns the filtered list of stories where at least one trigger fires.

**Placeholder Comment**

Indicates that the current implementation is temporary and subject to future enhancements or replacements. Contrary to the comment, the provided implementation does filter stories based on triggers, not returning all stories.

## 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)

    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:
            trigger_name, trigger_type = parts[0], 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':
                triggers[trigger_name] = NotTrigger(triggers[parts[2]])
            elif trigger_type == 'AND':
                triggers[trigger_name] = AndTrigger(
                    triggers[parts[2]], triggers[parts[3]])
                trigger_list.append(triggers[trigger_name])
            elif trigger_type == 'OR':
                triggers[trigger_name] = OrTrigger(
                    triggers[parts[2]], triggers[parts[3]])
                trigger_list.append(triggers[trigger_name])

    return trigger_list  # Return the trigger list, not print
```

### Function Definition

Introduces the read_trigger_config function to parse a trigger configuration file.

**Function Parameter**

Takes a single parameter filename representing the name of the configuration file.

**Documentation**

Includes a docstring explaining the function's purpose and parameters.

**Reading Configuration File**

Opens the file specified by filename in read mode. Stores non-blank and non-comment lines in the lines list after stripping whitespace.

**Parsing Configuration**

The parsing of the configuration file is marked as a TODO item for future implementation. Temporary measure: Prints the parsed lines to visualize the file's content.

**Return Statement**

Currently doesn't return any value; prints parsed lines for visualization.

**Overall**

Serves as an initial step in processing the configuration file, preparing it for further parsing and trigger object creation. The actual parsing and trigger instantiation logic are pending implementation.

## output