NUST

NATIONAL UNIVERSITY
OF SCIENCES & TECHNOLOGY

# End semester Fop project

| Name | Reg no |
|---|---|
| Hasnain Ali | 478806 |
| Salar Azam | 479001 |
| Haseeb Tahir | 453901 |
| Sakhawat Ali | 470935 |
| Abdul Rafeh | 417648 |

```
import feedparser

import string

import time

import threading

from project_util import translate_html

from tkinter import *

from datetime import datetime


def process(url):
    """
    Fetches news items from the rss url and parses them.
    Returns a list of NewsStory instances.
```

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

    # Check if description field exists
    if 'description' in entry:
        description = translate_html(entry.description)
    else:
        description = ""

    # Handling different date formats
    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

#======================
# Data structure design
#======================

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

#======================
# Triggers
#======================

class Trigger(object):
    def evaluate(self, story):
        raise NotImplementedError

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


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


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


class TimeTrigger(Trigger):
    def __init__(self, time):
        self.time = datetime.strptime(time, "%Y-%m-%dT%H:%M:%SZ")


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)
```

```
#======================
# 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]
        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

#======================
# Main Thread
#======================

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 = []

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

            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-separated): ").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()
```