

Project

Name: Muhammad Ibrahim

Class: ME-15-A

Section: A

CMS ID: 463865

**Code and Explanation:**

import feedparser

import string

import time

import threading

from mtTkinter import \*

from datetime import datetime

import pytz

class NewsStory:

def \_\_init\_\_(self, guid, title, description, link, pubdate):

self.guid, self.title, self.description, self.link, self.pubdate = guid, title, description, link, 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 Trigger:

def evaluate(self, story):

raise NotImplementedError

class PhraseTrigger(Trigger):

def \_\_init\_\_(self, phrase):

self.phrase = phrase.lower()

def is\_phrase\_in(self, text):

text = ''.join([c if c not in string.punctuation else ' ' for c in text.lower()])

return f' {self.phrase} ' in f' {text} '

def evaluate(self, story):

raise NotImplementedError

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\_string):

self.time = datetime.strptime(time\_string, "%d %b %Y %H:%M:%S").replace(tzinfo=pytz.utc).astimezone(pytz.timezone("US/Eastern"))

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, self.trigger2 = trigger1, 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, self.trigger2 = trigger1, trigger2

def evaluate(self, story):

return self.trigger1.evaluate(story) or self.trigger2.evaluate(story)

def process(url):

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

guid, title, link = entry.get('guid', ''), entry.get('title', ''), entry.get('link', '')

description = entry.get('description', '')

pubdate\_str = entry.get('published', '')

pubdate = datetime.strptime(pubdate\_str, "%a, %d %b %Y %H:%M:%S %Z").replace(tzinfo=pytz.utc).astimezone(pytz.timezone("US/Eastern"))

ret.append(NewsStory(guid, title, description, link, pubdate))

return ret

def filter\_stories(stories, triggerlist):

return [story for story in stories if any(trigger.evaluate(story) for trigger in triggerlist)]

def read\_trigger\_config(triggers):

trigger\_file = open(triggers, 'r')

triggers = {}

trigger\_list = []

for line in trigger\_file:

line = line.strip()

if line and not line.startswith('//'):

parts = line.split(',')

if parts[0] == 'ADD':

trigger\_list.extend(triggers[name] for name in parts[1:] if name in triggers)

else:

trigger\_name, trigger\_type, trigger\_args = parts[0].strip(), parts[1].strip(), [arg.strip() for arg in parts[2:]]

trigger = eval(f"{trigger\_type}Trigger(\*trigger\_args)")

triggers[trigger\_name] = trigger

return trigger\_list

SLEEPTIME = 120

def main\_thread(master):

try:

frame = Frame(master)

frame.pack(side=BOTTOM)

scrollbar = Scrollbar(master)

scrollbar.pack(side=RIGHT, fill=Y)

title = StringVar()

title.set("Google & Yahoo Top News")

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=root.destroy)

button.pack(side=BOTTOM)

guidShown = set()

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.add(newstory.get\_guid())

while True:

print("Polling . . .", end=' ')

stories = process("http://news.google.com/news?output=rss")

stories.extend(process("http://news.yahoo.com/rss/topstories"))

triggerlist = read\_trigger\_config('triggers.txt')

stories = filter\_stories(stories, triggerlist)

list(map(get\_cont, stories))

scrollbar.config(command=cont.yview)

print("Sleeping...")

time.sleep(SLEEPTIME)

except Exception as e:

print(e)

if \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

root.title("Some RSS parser")

t = threading.Thread(target=main\_thread, args=(root,))

t.start()

root.mainloop()

Explanation:

1. The first part imports the source files into our program.

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 part defines a NewsStory class to store information about news articles. The \_\_init\_\_ method initializes a news story with GUID, title, description, link, and publication date. Getter methods provide access to each attribute.



def process(url):

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

guid = entry.get('guid', '')

title = translate\_html(entry.get('title', ''))

link = entry.get('link', '')

description = translate\_html(entry.get('description', '')) if 'description' in entry else ''

pubdate = translate\_html(entry.get('published', ''))

pubdate\_parsed = None

date\_formats = ["%a, %d %b %Y %H:%M:%S %Z", "%Y-%m-%dT%H:%M:%SZ"]

for fmt in date\_formats:

try:

pubdate\_parsed = datetime.strptime(pubdate, fmt)

pubdate\_parsed = pubdate\_parsed.replace(tzinfo=pytz.timezone("GMT"))

break

except ValueError:

continue

if pubdate\_parsed is None:

raise ValueError(f"Date format not recognized for: {pubdate}")

newsStory = NewsStory(guid, title, description, link, pubdate\_parsed)

ret.append(newsStory)

return ret

The process function fetches and parses RSS feed from a given URL. Parses the feed using feedparser. Extracts and processes each entry (news story), converting the publication date to a standardized format. Creates NewsStory objects and appends them to the ret list. Returns the list of NewsStory objects.



class Trigger:

def evaluate(self, story):

raise NotImplementedError

It acts as an abstract base class for all triggers. It defines an evaluate method that should be implemented by subclasses.



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):

raise NotImplementedError

It is inherited from Trigger. It stores a phrase in lowercase. is\_phrase\_in checks if the phrase is present in the given text after normalizing it. It evaluates method to be implemented by subclasses.



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())

It is a subclasses of PhraseTrigger. TitleTrigger checks if the phrase is in the title. DescriptionTrigger checks if the phrase is in the description.



class TimeTrigger(Trigger):

def \_\_init\_\_(self, time\_string):

self.time = datetime.strptime(time\_string, "%d %b %Y %H:%M:%S").replace(tzinfo=pytz.utc).astimezone(pytz.timezone("US/Eastern"))

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

TimeTrigger is a base class for time based triggers. It converts a time string to a datetime object in EST. BeforeTrigger triggers if the story's publication date is before the specified time. AfterTrigger triggers if the story's publication date is after the specified 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)

NotTrigger: Inverts the result of another trigger. AndTrigger: Fires if both of its component triggers fire.

OrTrigger: Fires if at least one of its component triggers fire.



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

This part filters a list of NewsStory objects based on a list of triggers. It adds a story to the filtered list if any trigger fires for it.



def read\_trigger\_config(filename):

trigger\_file = open(filename, 'r')

lines = [line.rstrip() for line in trigger\_file if line.rstrip() and not line.startswith('//')]

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

This function reads the trigger configuration file. It parses each line to create the appropriate trigger objects and returns a list of triggers to be used.



def main\_thread(master):

try:

frame = Frame(master)

frame.pack(side=BOTTOM)

scrollbar = Scrollbar(master)

scrollbar.pack(side=RIGHT, fill=Y)

title = StringVar()

title.set("Google & Yahoo Top News")

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=root.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 . . .", end=' ')

stories = process("http://news.google.com/news?output=rss")

stories.extend(process("http://news.yahoo.com/rss/topstories"))

triggerlist = read\_trigger\_config('triggers.txt')

stories = filter\_stories(stories, triggerlist)

list(map(get\_cont, stories))

scrollbar.config(command=cont.yview)

print("Sleeping...")

time.sleep(SLEEPTIME)

except Exception as e:

print(e)

The main\_thread function handles the GUI and periodic fetching of news. Sets up the GUI elements (frames, scrollbar, title, text area, exit button). get\_cont function updates the GUI with new stories that haven't been shown yet. Polls news feeds, applies triggers, and updates the GUI every SLEEPTIME seconds.



if \_\_name\_\_ == '\_\_main\_\_':

root = Tk()

root.title("RSS Feed Filter")

t = threading.Thread(target=main\_thread, args=(root,))

t.start()

root.mainloop()

This checks if the script is run as the main module. It creates the main Tkinter window and starts the main thread for fetching and displaying news.