import string

#======================

# Data structure design

#======================

# Problem 1

class NewsStory(object):

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):

"""

Returns True if an alert should be generated

for the given news item, or False otherwise.

"""

# DO NOT CHANGE THIS!

raise NotImplementedError

# PHRASE TRIGGERS

# Problem 2

# TODO: PhraseTrigger

class PhraseTrigger(Trigger):

""""

inherits from Trigger

"""

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

self.phrase = phrase.lower()

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

"""

Takes a phrase as input

returns True if phrase is in the story, ignoring all punctuation

"""

text = text.lower()

for item in (string.punctuation):

text = text.replace(item, " ")

wordlist = text.split(" ")

phraselist = self.phrase.split(" ")

while "" in wordlist:

wordlist.remove("")

print(phraselist)

for word in phraselist:

if word not in wordlist:

return False

if phraselist.index(word) + 1 != len(phraselist):

try:

if wordlist[wordlist.index(word) + 1] != phraselist[phraselist.index(word) + 1]:

return False

except IndexError:

return False

return True

story = NewsStory("", "Hello !! world", "", "", "")

trigger = PhraseTrigger("hello world")

trigger.is\_phrase\_in(story.get\_title())

# storylist = []

# for story in stories:

# for trigger in triggerlist:

# if trigger.evaluate(story):

# storylist.append(story)

# break

# return storylist

# CODE THAT RAN CORRECTLY:

# 6.0001/6.00 Problem Set 5 - RSS Feed Filter

# Name:

# Collaborators:

# Time:

import feedparser

import string

import time

import threading

from project\_util import translate\_html

from mtTkinter import \*

from datetime import datetime

import pytz

#-----------------------------------------------------------------------

#======================

# Code for retrieving and parsing

# Google and Yahoo News feeds

# Do not change this code

#======================

def process(url):

"""

Fetches news items from the rss url and parses them.

Returns a list of NewsStory-s.

"""

feed = feedparser.parse(url)

entries = feed.entries

ret = []

for entry in entries:

guid = entry.guid

title = translate\_html(entry.title)

link = entry.link

description = translate\_html(entry.description)

pubdate = translate\_html(entry.published)

try:

pubdate = datetime.strptime(pubdate, "%a, %d %b %Y %H:%M:%S %Z")

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

# pubdate = pubdate.astimezone(pytz.timezone('EST'))

# pubdate.replace(tzinfo=None)

except ValueError:

pubdate = datetime.strptime(pubdate, "%a, %d %b %Y %H:%M:%S %z")

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

ret.append(newsStory)

return ret

#======================

# Data structure design

#======================

# Problem 1

class NewsStory(object):

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):

"""

Returns True if an alert should be generated

for the given news item, or False otherwise.

"""

# DO NOT CHANGE THIS!

raise NotImplementedError

# PHRASE TRIGGERS

# Problem 2

class PhraseTrigger(Trigger):

""""

inherits from Trigger

"""

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

self.phrase = phrase.lower()

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

"""

Takes a phrase as input

returns True if phrase is in the story, ignoring all punctuation

"""

text = text.lower()

for item in (string.punctuation):

text = text.replace(item, " ")

wordlist = text.split(" ")

phraselist = self.phrase.split(" ")

while "" in wordlist:

wordlist.remove("")

for word in phraselist:

if word not in wordlist:

return False

if phraselist.index(word) + 1 != len(phraselist):

try:

if wordlist[wordlist.index(word) + 1] != phraselist[phraselist.index(word) + 1]:

return False

except IndexError:

return False

return True

# Problem 3

class TitleTrigger(PhraseTrigger):

def evaluate(self, story):

return self.is\_phrase\_in(story.get\_title())

# Problem 4

class DescriptionTrigger(PhraseTrigger):

def evaluate(self, story):

return self.is\_phrase\_in(story.get\_description())

# TIME TRIGGERS

# Problem 5

class TimeTrigger(Trigger):

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

format = "%d %b %Y %H:%M:%S"

pubdate = datetime.strptime(pubdate, format)

pubdate = pubdate.replace(tzinfo=pytz.timezone("EST"))

self.pubdate = pubdate

# Problem 6

# TODO: BeforeTrigger and AfterTrigger

class BeforeTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate().replace(tzinfo=pytz.timezone("EST")) < self.pubdate

class AfterTrigger(TimeTrigger):

def evaluate(self, story):

return story.get\_pubdate().replace(tzinfo=pytz.timezone("EST")) > self.pubdate

# COMPOSITE TRIGGERS

# Problem 7

class NotTrigger(Trigger):

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

self.trig = trigger

def evaluate(self, story):

return not self.trig.evaluate(story)

# Problem 8

class AndTrigger(Trigger):

def \_\_init\_\_(self, trigger1, trigger2):

self.trig1 = trigger1

self.trig2 = trigger2

def evaluate(self, story):

return self.trig1.evaluate(story) and self.trig2.evaluate(story)

# Problem 9

class OrTrigger(Trigger):

def \_\_init\_\_(self, trigger1, trigger2):

self.trig1 = trigger1

self.trig2 = trigger2

def evaluate(self, story):

return self.trig1.evaluate(story) or self.trig2.evaluate(story)

#======================

# Filtering

#======================

# 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

return stories

#======================

# User-Specified Triggers

#======================

# Problem 11

def read\_trigger\_config(filename):

"""

filename: the name of a trigger configuration file

Returns: a list of trigger objects specified by the trigger configuration

file.

"""

# We give you the code to read in the file and eliminate blank lines and

# comments. You don't need to know how it works for now!

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)

# TODO: Problem 11

# line is the list of lines that you need to parse and for which you need

# to build triggers

print(lines) # for now, print it so you see what it contains!

SLEEPTIME = 120 #seconds -- how often we poll

def main\_thread(master):

# A sample trigger list - you might need to change the phrases to correspond

# to what is currently in the news

try:

t1 = TitleTrigger("Cummings")

t2 = DescriptionTrigger("Dominic")

t3 = DescriptionTrigger("cummings")

t4 = AndTrigger(t2, t3)

triggerlist = [t1, t4]

# Problem 11

# TODO: After implementing read\_trigger\_config, uncomment this line

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

# HELPER CODE - you don't need to understand this!

# Draws the popup window that displays the filtered stories

# Retrieves and filters the stories from the RSS feeds

frame = Frame(master)

frame.pack(side=BOTTOM)

scrollbar = Scrollbar(master)

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

t = "Google 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=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=' ')

# Get stories from Google's Top Stories RSS news feed

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

# Get stories from Yahoo's Top Stories RSS news feed

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

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()