# RSS Feed Filter

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"))

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

# TODO: NewsStory

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):

"""

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):

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 phrase\_words == words[i:i + len(phrase\_words)]:

return True

return False

# Problem 3

# TODO: TitleTrigger

class TitleTrigger(PhraseTrigger):

def evaluate(self, story):

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

# Problem 4

# TODO: DescriptionTrigger

class DescriptionTrigger(PhraseTrigger):

def evaluate(self, story):

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

# TIME TRIGGERS

# Problem 5

# TODO: TimeTrigger

# Constructor:

# Input: Time has to be in EST and in the format of "%d %b %Y %H:%M:%S".

# Convert time from string to a datetime before saving it as an attribute.

class TimeTrigger(Trigger):

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

est = pytz.timezone("EST")

self.time = est.localize(datetime.strptime(time\_string, "%d %b %Y %H:%M:%S"))

# Problem 6

# TODO: BeforeTrigger and AfterTrigger

class BeforeTrigger(TimeTrigger):

def evaluate(self, story):

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

class AfterTrigger(TimeTrigger):

def evaluate(self, story):

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

# COMPOSITE TRIGGERS

# Problem 7

# TODO: NotTrigger

class NotTrigger(Trigger):

def \_init\_(self, trigger):

self.trigger = trigger

def evaluate(self, story):

return not self.trigger.evaluate(story)

# Problem 8

# TODO: AndTrigger

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)

# Problem 9

# TODO: OrTrigger

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

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

# 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.

"""

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

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

# 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)

trigger\_file.close()

# TODO: Problem 11

triggers = {}

triggerlist = []

for line in lines:

parts = line.split(',')

if parts[0] == "ADD":

for name in parts[1:]:

triggerlist.append(triggers[name])

else:

name = parts[0]

trigger\_type = parts[1]

if trigger\_type == "TITLE":

triggers[name] = TitleTrigger(parts[2])

elif trigger\_type == "DESCRIPTION":

triggers[name] = DescriptionTrigger(parts[2])

elif trigger\_type == "BEFORE":

triggers[name] = BeforeTrigger(parts[2])

elif trigger\_type == "AFTER":

triggers[name] = AfterTrigger(parts[2])

elif trigger\_type == "NOT":

triggers[name] = NotTrigger(triggers[parts[2]])

elif trigger\_type == "AND":

triggers[name] = AndTrigger(triggers[parts[2]], triggers[parts[3]])

elif trigger\_type == "OR":

triggers[name] = OrTrigger(triggers[parts[2]], triggers[parts[3]])

return triggerlist

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("election")

t2 = DescriptionTrigger("Trump")

t3 = DescriptionTrigger("Clinton")

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 & 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=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()