```python
# -*- coding: utf-8 -*-
"""
Created on Mon Mar 16 14:25:42 2020

VHB 2020 Preconference workshop "Introduction to Textual Analysis using Python"
Date: March 17, 2020

@author: Alexander Hillert, Goethe-University Frankfurt

"""

##########################################
# 1. Opening the earnings call transcript
##########################################

# To rerun this code, enter the directory of the MSFT transcript below.
# Make sure to use "/" not "\" to set your directory in Python.
directory="C:/Lehre/VHB_2020/"

# Open the txt file using open()
# Enter the filename, specify whether to read ("r") or write ("w") the file,
# and set the encoding.
conference_call_file=open(directory+"MSFT_2020Q2_Call_Transcript.txt","r",encoding="utf-8")
# The previous command only creates a reference to the txt file.
# To read the actual content/text of the file, you have to use the .read() command
conference_call_text=conference_call_file.read()

# Now you should find the new variable "conference_call_text" in your "Variable Explorer"
# To check whether opening the file worked properly, you can print the content.
print(conference_call_text)

#######################################################
# 2. How to count (negative) words? Introductory example
#######################################################
# Obtain the Loughran and McDonald (2011) dictionary at
# https://sraf.nd.edu/textual-analysis/resources/#LM%20Sentiment%20Word%20Lists
# Download the file "LoughranMcDonald_SentimentWordLists_2018.xlsx" and copy
# the negative words to an empty txt file.
# Save the file as "LMD_negative_dictionary.txt"
# Open this txt file. -> open() command -> see line 23.
word_list_file=open(directory+'LMD_negative_dictionary.txt','r',encoding='utf-8')
word_list_text=word_list_file.read()
print(word_list_text)

# The LMD words are all in upper case. Does this create a problem?
# Example:
test_text="MSFT reported good earnings."
print("How often is the word 'good' in the test text?")
frequency_of_good=test_text.count("good")
print(frequency_of_good)
print("How often is the word 'GOOD' in the test text?")
frequency_of_GOOD=test_text.count("GOOD")
print(frequency_of_GOOD)

# -> we have to use a consistent format. For example, change both the original text
# and the word list to lower case.
conference_call_text_lower_case=conference_call_text.lower()
word_list_text=word_list_text.lower()

# So far, all negative words are in a single text/variable. However, we need
# them to be in separate variables so that we can count how often each negative
# word appears in the text.
# Idea: count negative word 1, count negative words 2, ..., count negative word N.
# Lists are a type of Python objects that are very useful for this purpose.
# To get the list of negative words we can split the text by line, as
# every line contains exactly one negative word.
# To split by line, use the split() command. "\n" is the code for line breaks.
negative_words=word_list_text.split('\n')

print("\nBelow you see the list of negative words.")
print(negative_words)
print("\n\n") # creates empty lines to better structure the output window.

# Now, the negative words are separate list elements (see also Variable Explorer).
# How to work with the list?
print("The first negative word is "+negative_words[0])
print("The second negative word is "+negative_words[1])
print("The third negative word is "+negative_words[2])
print("The fourth negative word is "+negative_words[3])
print("The last negative word is "+negative_words[2354])
# The list has 2355 elements (as shown in the Variable Explorer).
# However, as counting starts at 0, the last element is number 2354.
# the following command would raise an error because there is no element 2355.
# print("The last negative word is "+negative_words[2355])

# How is the list helpful?
# -> we can write a loop from 0 to 2354 to automatically count the frequencies
# of the negative words.
print("\nFirst example\n")
test_text="Due to the Corona virus the firm had to abandon its operations."

# The first negative word is abadon -> count how often we find abadon in the text.
frequency_of_abandon=test_text.count(negative_words[0])
print("\nThe negative word "+negative_words[0]+" is found "+str(frequency_of_abandon))

# Now count many negative words.
# In this example we use a while loop
print("\nFirst example - using a loop\n")
i=0
while i<=3:
    print("The negative word number "+str(i)+" is "+negative_words[i])
    # count the frequency of negative word i.
    frequency_of_negative_word=test_text.count(negative_words[i])
    # print the result.
    print("In the test text "+negative_words[i]+" is found "\
    +str(frequency_of_negative_word)+" times.")
    # continue with the next negative word
    i=i+1
# -> So far, the result looks okay.

# Consider a second example
print("\nSecond example\n")
test_text="Due to the Corona virus the firm had to abandon its operations. "\
+"This abandonment hurts earnings."

print("Now, our test_text is:\n"+test_text)
i=0
while i<=3:
    print("The negative word number "+str(i)+" is "+negative_words[i])
    frequency_of_negative_word=test_text.count(negative_words[i])
    print("In the test text "+negative_words[i]+" is found "\
    +str(frequency_of_negative_word)+" times.")
```

```python
    # continue with the next negative word
    i=i+1

# PROBLEM: abandon is counted twice!!!
# Why? Because "abandon" is part of ABADONment.
# How to solve this problem?
# -> Split test_text into a list of individual words.
# How does splitting into words work?
# -> use so called "word boundaries"
# In Python syntax a word boundary is "\W".
# Symbols like ".", "?", " ", and "\n" are all word boundaries.
# There can be multiple word boundaries in a row. In our example "operations. This"
# -> there is the "." and the " "
# -> require not exactly 1 but at least 1 word boundary -> "\W{1,}"
# i.e. we split the text by one or more word boundaries.
# Word boundaries are part of so called Regular Expressions.
# -> import the Python module for Regular Expressions.
import re
# Split the test_text into words
test_words=re.split("\W{1,}",test_text)
print("\nThe words of our test_text are:")
print(test_words)

# Does using the list help in getting the word count right?
print("\nSecond example - second try\n")
i=0
while i<=3:
    print("The negative word number "+str(i)+" is "+negative_words[i])
    # now use test_words not test_text.
    frequency_of_negative_word=test_words.count(negative_words[i])
    print("In the test text "+negative_words[i]+" is found "\
    +str(frequency_of_negative_word)+" times.")
    # continue with the next negative word
    i=i+1
# Yes, this approach works!

##############################################################
# 3. How to count (negative) words? Applied to MSFT's call
##############################################################
# the list of negative words is ready.
# We need to split MSFT's text into words
# Like before, use at least one word boundary "\W{1,}" to split.
conference_call_words=re.split("\W{1,}",conference_call_text_lower_case)
print(conference_call_words)

# We want to count the number of negative words in total.
negative_word_count=0

# Use the while loop but now from 0 to 2354 (=number of negative words).
i=0
while i<=2354:
    # Determine the frequency of negative word i.
    frequency_of_negative_word=conference_call_words.count(negative_words[i])
    # add it to the total number of negative words in the text.
    negative_word_count=negative_word_count+frequency_of_negative_word
    # continue with the next negative word.
    i=i+1

print("\nThe earnings conference call contains "+str(negative_word_count)\
+" negative words in total.\n")
```

```
################################################################################
# 4. How to determine the PERCENTAGE of negative words? Applied to MSFT's call
################################################################################
# What is the total number of words in the call?
# Every list element in conferece_call_words is a single word.
# -> the number of list elements is the number of words.
# You can get the number of list elements by the length len() of the list.

word_count=len(conference_call_words)

print("The earnings conference call contains "+str(word_count)+" words in total.\n")

percentage_of_negative_words=negative_word_count/word_count*100

print("The earnings conference call contains "+str(percentage_of_negative_words)\
+"% negative words.\n")


####################################################################################
# 5. What are the most frequent words in the conference call?
####################################################################################
# An easy way to get the most frequent words is to use so called counters.
# Counters take a list as input and show the frequency of every value in the list.

# To use Counters you have to import the "collections" package.
import collections

# Example
word_list_example=["good","bad","earnings","bad","good","bad"]
print("\nThe list in our example is:\n")
print(word_list_example)

# Add the words to our counter
word_list_example_counter=collections.Counter(word_list_example)
print("\nThe resulting counter in our example is:")
print(word_list_example_counter)

# Apply the counter to MSFT's earnings call.
conference_call_counter=collections.Counter(conference_call_words)

# Get the 30 most frequent words
top_30_words=conference_call_counter.most_common(30)

# Print the top 30 words
print("\nThe 30 most frequent words in MSFT's conference call are")
print(top_30_words)


####################################################################################
# 6. What are the most frequent negative words?
####################################################################################
# Determining the most frequent negative words is not as easy as determining
# the most frequent words in the text because we first need to create a list
# of all negative words in the text.
# Approach: go over each word in the document. If the word is in the LMD negative
# dictionary, add the word to the list of negative words
# create an empty list.
negative_conference_call_words=[]

# As before, we use a while loop.
```

```python
# We know that there are 8348 words in the transcript.
# Numbering in Python starts at 0 -> loops ends at i<=8347
i=0
while i<=8347:
    # Check whether word i of the text is a negative word.
    if conference_call_words[i] in negative_words:
        negative_conference_call_words.append(conference_call_words[i])

    # continue with the next word of the text.
    i=i+1

# Consistency check
# From our counting of negative words, we know that there are 54 negative
# words in the text.
# Consequently, the list negative_conference_call_words should have 54 elements.
length_negative_call_words=len(negative_conference_call_words)
print("\nThe list negative_conference_call_words has "+\
str(length_negative_call_words)+" elements.\n")
# -> consistent.

# What are the most frequent negative words?
# -> apply the Counter commands from Step 5.
conference_call_negative_counter=collections.Counter(negative_conference_call_words)

# Get the 30 most frequent negative words.
top_30_negative_words=conference_call_negative_counter.most_common(30)

# Print the top 30 negative words.
print("\nThe 30 most negative frequent words in MSFT's conference call are")
print(top_30_negative_words)

# Question is by far the most frequent negative words (29 instances).
# While question may be negative in an annual report, it is unlikely to be
# a negative word in an earnings call ("Next question, please.").
# CONCLUSION:
# check the most frequent tone words? Do they make economically sense?
# If not, adjust the word list.


################################################################################
# 7. What should be done before running a textual analysis?
# Cleaning/editing of the texts.
# We do these operations on the text containing lower and upper case letters
# to be better able to compare the edited and unedited version.
################################################################################

# 1. delete the document header
# Microsoft FY20 Second Quarter Earnings Conference Call
# Michael Spencer, Satya Nadella, Amy Hood
# Wednesday, January 29, 2020
# Search for the last line of the document header
match_document_header=re.search("Wednesday, January 29, 2020",conference_call_text)
# get the position of this text within the text
end_coordinate_document_header=match_document_header.end()
# keep the text after the header
conference_call_text_edited=conference_call_text[end_coordinate_document_header:]

# 2. delete the document footer, i.e. the "END" at the end of the document
# Search for the line of "END"
match_document_end=re.search("\nEND\n",conference_call_text_edited)
# get the position of "END" within the text
```

```python
start_coordinate_document_end=match_document_end.start()
# isolate the text before the "END"
conference_call_text_edited=conference_call_text_edited[:start_coordinate_document_end]

# 3. Delete operator statements
# operator statement lines start with "OPERATOR: "
# These are only single-line statements. We can use a Regex searching for
# "OPERATOR: " at the beginning and a line break "\n" at the end of the line.
# Between the beginning of the line and the end of the line, there can be any character.
# Any character means letters, numbers, smybols, whitespaces but not a line break.
# "." is the Regex symbol for any character.
# ".{1,}" means that one or more letter, number, smybols can be between the
# beginning of the line and the line break at the end.
# We replace the operator text line by an empty line ("\n")
conference_call_text_edited=re.sub("\nOPERATOR: .{1,}\n","\n",conference_call_text_edited)


# To compare the original text to the edited version, we create a second txt file.
conference_call_file_output=open(directory+"MSFT_2020Q2_Call_Transcript_edited.txt",\
"w",encoding="utf-8")
# write the text to the output file.
conference_call_file_output.write(conference_call_text_edited)
# close the output file to write it to your hard drive.
conference_call_file_output.close()

# Now, you can open the original and the edited file and compare the two
# in Notepad++ using the "compare" plugin (see Slides).
```