Python Lab Assignment 5

# Questions

Generate two or more random paragraphs using the given online website

<https://randomwordgenerator.com/paragraph.php>

and perform the following tasks on it.

1. Find the total number of paragraphs and total number of lines and words in each paragraph.
2. Find all the words that begin with vowel in the first paragraph & those which begin with consonants in the second paragraph.
3. Find all the words having numerals.
4. Find all the words which begin with letter 'data/Data/DATA'. If no such words exist then add such related words and repeat the search.
5. Find all the words which end with letter 'e'.
6. Find all the words that begins with an vowel and ends with an vowel OR that begins with a consonant and ends with a consonant.
7. Find all the words that have the letters 'to' in them, find the position of 'to' in each word.
8. Find all the words that have capital letters in them.
9. Find all the words that have special symbols in them.
10. Find the first occurrence of full stop in your input.
11. Find the words that don’t have any vowel in them. If no such words exist add a few related words and repeat the task.
12. Find the first occurrent of a word that does not begin with consonant but ends with 'ing' and has 'ta' in it.

**NOTE**

Before starting the execution, the paragraph should be appended with the following sentence...

"This random entertaining paragraph was generated using https://randomwordgenerator.com/paragraph.php for [your register number]”.

# Code

import re

from os.path import isfile

*# INITIALISING THE PROGRAM...*

*#------------------------------------------------*

fileName = "randomText.txt"

file = 0

*# CHECKING IF FILE EXISTS*

if isfile(fileName):

file = open(fileName, "r") *# Opening the file stream.*

else:

print("\nFile '" + fileName + "' not found.\n")

exit() *# Terminates main program.*

*# CREATING THE STRING*

text = "These random entertaining paragraphs were generated using https://randomwordgenerator.com/paragraph.php for 1940223.\n\n"

text = text + file.read()

file.close() *# Closing the file stream as it's no longer needed*

*# SOME IMPORTANT PATTERNS*

wordSplitPattern = **r**"[\.\,\"]\*\s+[\.\,\"]\*"

lineSplitPattern = **r**"\.\s+"

*# ... INITIALISATION FINISHED*

*#------------------------------------------------*

*# UNIQUE WORD MATCHING FUNCTION*

**def** none(tmp): return ""

**def** start(tmp): return tmp.start() + 1

**def** end(tmp): return tmp.end()

**def** \_is(tmp): return tmp

**def** \_not(tmp): return not tmp

**def** uwm4(pattern, string, functions, inverse):

uwmList, condition = [], 0 *# unique word matches*

words = re.split(wordSplitPattern, string)

*# Match the pattern or match everything not the pattern*

if inverse: condition = \_not

else: condition = \_is

for x in words:

tmp = re.search(pattern, x)

*# This function returns a match object that contains useful information*

if condition(tmp) and x not in uwmList:

uwmList.append(x)

print("-", x, end = "")

for f in functions:

try: print(f[0](tmp), end = f[1])

except: print()

**def** uwm3(pattern, string, functions): uwm4(pattern, string, functions, False)

**def** uwm2(pattern, string): uwm4(pattern, string, [none], False)

**def** uwm(args):

n = len(args)

if n == 2: uwm2(args[0], args[1])

elif n == 3: uwm3(args[0], args[1], args[2])

elif n == 4: uwm4(args[0], args[1], args[2], args[3])

else: print("Invalid number of arguments!")

*# PRINTING THE SOURCE TEXT*

**def** seeText():

print("========================")

print("THE SOURCE TEXT")

print("\_\_\_\_\_\_\_\_\_\_\_\_")

print(text)

*# QUESTION 1*

**def** q1():

*# SPLITTING TEXT INTO PARAGRAPHS*

paragraphs = re.split(**r**"\n+", text)

nParagraphs = len(paragraphs)

print("------------------------")

print("No. of paragraphs:", nParagraphs)

for i in range(0, nParagraphs):

print("\nPARAGRAPH", i + 1)

nLines = len(re.split(lineSplitPattern, paragraphs[i]))

nWords = len(re.split(wordSplitPattern, paragraphs[i]))

print("No. of sentences:", nLines)

print("No. of words:", nWords)

*# QUESTION 2*

**def** q2():

print("------------------------")

print("WORDS BEGINNING WITH VOWELS IN PARAGRAPH 1")

paragraphs = re.split(**r**"\n+", text)

uwm(args = (**r**"^[aeiouAEIOU]", paragraphs[0]))

print("------------------------")

print("WORDS BEGINNING WITH CONSONANTS IN PARAGRAPH 2")

uwm(args = (**r**"^[^aeiouAEIOU\d\W]", paragraphs[1]))

"""

[^\d] excludes numeric characters

[^\W] excludes non-alphanumeric characters

"""

*# QUESTION 3*

**def** q3():

print("------------------------")

print("WORDS CONTAINING NUMERALS")

print("(Initial reference at the start also included)")

uwm(args = (**r**"\d+", text))

*# Searches for numerals within the word*

*# QUESTION 4*

**def** q4():

print("------------------------")

print("WORDS STARTING WITH 'DATA'")

uwm(args = (**r**"^data|Data|DATA", text))

*# QUESTION 5*

**def** q5():

print("------------------------")

print("WORDS ENDING WITH E")

uwm(args = (**r**"e$", text))

*# QUESTION 6*

**def** q6():

print("------------------------")

print("WORDS FLANKED BY ONLY VOWELS OR ONLY CONSONANTS")

pattern = **r**"(^[aeiouAEIOU].\*[aeiouAEIOU]$)|(^[^aeiouAEIOU\d\W].\*[^aeiouAEIOU\d\W]$)"

uwm(args = (pattern, text))

"""

BREAKING DOWN THE PATTERN...

vowelFlankPattern = r"^[aeiouAEIOU].\*[aeiouAEIOU]$"

consonantFlankPattern = r"^[^aeiouAEIOU\d\D].\*[^aeiouAEIOU\d\D]$"

"""

*# QUESTION 7*

**def** q7():

print("------------------------")

print("FINDING WORDS WITH 'TO'")

print("(And finding the position of 'to' in the word)")

print("\nformat: word, start position of 'to' in word\n")

uwm(args = (**r**"(t|T)(o|O)", text, [(none, ", "), (start, "\n")]))

*# Allows for all varieties of 'to' ex. "To", "TO"...*

*# QUESTION 8*

**def** q8():

print("------------------------")

print("FINDING WORDS WITH CAPITAL LETTERS")

uwm(args = (**r**"[A-Z]", text))

*# QUESTION 9*

**def** q9():

print("------------------------")

print("FINDING WORDS WITH SPECIAL SYMBOLS")

uwm(args = (**r**"\W", text))

*# QUESTION 10*

**def** q10():

print("------------------------")

print("INDEX OF FIRST OCCURRENCE OF FULL STOP")

tmp = re.search(**r**"\.", text)

print(tmp.start())

*# QUESTION 11*

**def** q11():

print("------------------------")

print("FINDING WORDS WITH NO VOWELS")

uwm(args = (**r**"[aeiouAEIOU]", text, [none], True))

*# QUESTION 12*

**def** q12():

print("------------------------")

print("FINDING THE FIRST WORD THAT...")

print("\* does not start with a consonant")

print("\* has 'ta' in it")

print("\* ends with 'ing'")

print("\nformat: word, start index of word in text, end index of word in text\n")

*# SEARCHING THE PATTERN*

pattern = **r**"[\s\,\"\.]+[aeiouAEIOU\d\W][\w]\*ta[\w]\*ing[\s\,\"\.]+"

tmp = re.search(pattern, " " + text + " ")

*# " " is inserted at first and last, to allow the intended pattern to be checked for the first and last words of the file.*

*# FORMATTING THE MATCHED PATTERN*

match = re.split(**r**"[\s\,\"\.]+", tmp.group())[1]

*# This isolates the matched word i.e. tmp.group() from extraneous symbols and spaces.*

*# PRINTING THE RESULTS*

print("-", match, end = ", ")

print(tmp.start(), end = ", ")

print(tmp.end())

*# PRINTING THE QUESTIONS OF THIS ASSIGNMENT*

**def** seeQuestion():

fileName = "question.txt"

file = 0

*# CHECKING IF FILE EXISTS*

if isfile(fileName):

file = open(fileName, "r") *# Opening the file stream.*

else:

print("\nFile '" + fileName + "' not found.\n")

return

*# PRINTING THE FILE*

print("------------------------")

print(file.read())

file.close()

**def** seeInstructions():

print("========================")

print("ANSWERS TO QUESTIONS IN ASSIGNMENT 5")

print("\* Enter desired question number to get its answer.")

print("\* Enter \* to see the random text.")

print("\* Enter ? to view the questions.")

print("\* Enter x exit the program.")

print("\* Enter i to view these instructions again.")

**def** switch(option):

try:

{

"\*":seeText,

"1":q1,

"2":q2,

"3":q3,

"4":q4,

"5":q5,

"6":q6,

"7":q7,

"8":q8,

"9":q9,

"10":q10,

"11":q11,

"12":q12,

"x":exit,

"?":seeQuestion,

"i":seeInstructions

}[option]()

except KeyError: print("Invalid option!")

seeInstructions()

while True:

print("------------------------")

option = input(">> ")

switch(option)

# Inferences

## NOTES ON READ FUNCTION

The read function has two possible functions:

1. Read and return n number of bytes (1 character is 1 byte).

2. Read and return everything from the file as one string.

## NOTES ON THE SPLIT PATTERNS

**wordSplitPattern = r"[\.\,\"]\*\s+[\.\,\"]\*"**

The word split pattern allows for any combination of period, comma and double quote before or after the whitespace. This ensures that these special symbols are removed from the word to which they are attached. This is not the same as

**r"\.+|\,+|\"+|\s+"**

as this one checks for a sequence of consecutive characters of the same type.

**lineSplitPattern = r"\.\s+"**

The line split pattern presupposes an identification of sentences wherein

1. It has at least one period.

2. It has at least one whitespace after the period.

(Whitespace includes newline and end of file)

## NOTES ON BACKSLASH-PRECEDED CHARACTERS...

Characters such as quotes and double quotes (i.e. which have special meaning in Python) are represented in the string by preceding them with a backslash. This is to signify the character and not treat it as a special symbol. Other such characters are...

* . (dot)
* , (comma)
* \ (backslash)

When handled with RegEx functions, invisible or non-printable characters such as newline are represented as they are in Python ex. "\n". In RegEx, they are not translated into their on-screen equivalents (such as an actual newline on-screen).

## NOTES ON COMPILE FUNCTION

Example...

**wordSplitPattern = re.compile(r"[\.\,\"]\*\s+[\.\,\"]\*")**

The re.compile function returns a Pattern object based on the regular expression given. It is not necessary to use this in order to search for regular expressions using the re module. However, through a Pattern object, you can access tailor-made exceptions, functions and attributes. In my program, however, I have not used this function, simply to drive home the point that it is not strictly necessary.