Assignment 2

Author: Tanay Mukherjee

In [1]:

```
# 1. Describe the class of strings matched by the following regular expressions.No code is needed and just

#a.[a-zA-Z]+
#b.[A-Z][a-z]*
#c.p[aeiou]{,2}t
#d.\d+(\.\d+)?
#e.([^aeiou][aeiou][^aeiou])*
#f.\w+|[^\w\s]+
```

In [2]:

```
import nltk, re, pprint
```

In [3]:

```
test = "The companies, Moderna and Pfizer, revealed details about how participants are being selected and
# Took the test data from NY Times blog

# a.
nltk.re_show(r'[a-zA-Z]+', test)

# Ans: It is matching anything that has alphabets, whether or not it has upper case or lower case.
```

```
{The} {companies}, {Moderna} {and} {Pfizer}, {revealed} {details} {about} {how} {participants} {are} {bein
g} {selected} {and} {monitored}, {the} {conditions} {under} {which} {the} {trials} {could} {be} {stopped}
{early} {if} {there} {were} {problems}, {and} {the} {evidence} {researchers} {will} {use} {to} {determine}
{whether} {people} {who} {got} {the} {vaccines} {were} {protected} {from} {Covid}-19.
```

In [4]:

```
# b.
nltk.re_show(r'[A-Z][a-z]*', test)

# Ans: All capitalized words are matched which has first letter as capital.
```

```
{The} companies, {Moderna} and {Pfizer}, revealed details about how participants are being selected and mon
itored, the conditions under which the trials could be stopped early if there were problems, and the eviden
ce researchers will use to determine whether people who got the vaccines were protected from {Covid}-19.
```

In [5]:

```
# c.
nltk.re_show(r'p[aeiou]{,2}t', test)

# Ans: Words starting with p and ending in t. And between them either 0, 1 or 2 vowels from english langua

# The example I chose had not like that so see a new example below:
nltk.re_show(r'p[aeiou]{,2}t', "pant")
nltk.re_show(r'p[aeiou]{,2}t', "pat")
nltk.re_show(r'p[aeiou]{,2}t', "pout")
nltk.re_show(r'p[aeiou]{,2}t', "pan")
```

```
The companies, Moderna and Pfizer, revealed details about how participants are being selected and monitore
d, the conditions under which the trials could be stopped early if there were problems, and the evidence re
searchers will use to determine whether people who got the vaccines were protected from Covid-19.
pant
{pat}
{pout}
pan
```

In [6]:

```
# d.
nltk.re_show(r'\d+(\.\d+)?', test)

# Ans: It will return any number - whole, fraction or integer. Anything that falls in the number line.
```

```
The companies, Moderna and Pfizer, revealed details about how participants are being selected and monitore
d, the conditions under which the trials could be stopped early if there were problems, and the evidence re
searchers will use to determine whether people who got the vaccines were protected from Covid-{19}.
```

In [7]:

```
# e.
nltk.re_show(r'([^aeiou][aeiou][^aeiou])*', test)

# Ans: Zero or more sequence of a combination of consonants-vowel-consonants.
```

```
{}T{he compan}{}i{}e{}s{},{} {Mod}{}e{}r{na }{}a{}n{}d{} {}P{fiz}{}e{}r{},{} {rev}{}e{}a{led}{} {det}{}a{}i
{}l{}s{ ab}{}o{}u{}t{} {how}{} {partic}{}i{pan}{}t{}s{ ar}{}e{} {}b{}e{}i{}n{}g{} {sel}{}e{}c{ted an}{}d{}
{mon}{}i{tor}{}e{}d{},{} {}t{he condit}{}i{}o{}n{}s{ under}{} {}w{hic}{}h{} {}t{he }{}t{}r{}i{}a{}l{}s{} {}
c{}o{}u{}l{}d{} {be }{}s{topped}{} {}e{}a{}r{}l{}y{ if}{} {}t{her}{}e{} {wer}{}e{} {}p{roblem}{}s{},{ an}{}
d{} {}t{he }{}e{vid}{}e{}n{ce res}{}e{}a{}r{}c{her}{}s{} {wil}{}l{ us}{}e{} {to det}{}e{}r{min}{}e{} {}w{he
ther}{} {}p{}e{}o{}p{le }{}w{ho got}{} {}t{he vaccin}{}e{}s{} {wer}{}e{} {}p{rot}{}e{}c{ted}{} {}f{rom}{}
{Cov}{}i{}d{}-{}1{}9{}.{}
```

In [8]:

```
# f.
nltk.re_show(r'\w+|[^\w\s]+', test)

# Ans: All alphanumeric characters including punctuation and non-whitespaces.
```

```
{The} {companies}{,} {Moderna} {and} {Pfizer}{,} {revealed} {details} {about} {how} {participants} {are} {b
eing} {selected} {and} {monitored}{,} {the} {conditions} {under} {which} {the} {trials} {could} {be} {stopp
ed} {early} {if} {there} {were} {problems}{,} {and} {the} {evidence} {researchers} {will} {use} {to} {deter
mine} {whether} {people} {who} {got} {the} {vaccines} {were} {protected} {from} {Covid}{-}{19}{.}
```

In [9]:

```
# 2. Rewrite the following loop as a list comprehension:
# sent = ['This', 'is', 'an', 'introduction', 'class']
# result = []
# for word in sent:
#     word_len = (word, len(word))
#     result.append(word_len)
# result
```

In [10]:

```
# Running the sample code:

sent = ['This', 'is', 'an', 'introduction', 'class']
result = []
for word in sent:
    word_len = (word, len(word))
    result.append(word_len)
result
```

Out[10]:

```
[('This', 4), ('is', 2), ('an', 2), ('introduction', 12), ('class', 5)]
```

In [11]:

```
# Re-writing it as a list comprehension:

sent = ['This', 'is', 'an', 'introduction', 'class']
print([(w, len(w)) for w in sent])
```

```
[('This', 4), ('is', 2), ('an', 2), ('introduction', 12), ('class', 5)]
```

In [12]:

```
# 3. Read in some text from your own document in your local disk, tokenize it, and print the list of all wı
```

In [13]:

```
# The text in the file is same as what I used above for regex check

'''
"The companies, Moderna and Pfizer, revealed details about how participants are being selected and monitor
'''
```

Out[13]:

```
'\n"The companies, Moderna and Pfizer, revealed details about how participants are being selected and monit
ored, the conditions under which the trials could be stopped early if there were problems, and the evidence
researchers will use to determine whether people who got the vaccines were protected from Covid-19."\n'
```

In [14]:
```
file = open('test.txt')
```

In [15]:
```
file
```

Out[15]:
```
<_io.TextIOWrapper name='test.txt' mode='r' encoding='cp1252'>
```

In [16]:
```
from nltk import word_tokenize
from nltk.corpus import gutenberg
```

In [17]:
```
raw = file.read()
tokens = word_tokenize(raw)
```

In [18]:
```
# Method 1: Using the method startswith()

print([wh for wh in tokens if wh.lower().startswith('wh')])
```

```
['which', 'whether', 'who']
```

In [19]:
```
# Method 2: Using regex expression

print([wh for wh in tokens if re.findall('^[Ww]h', wh)])
```

```
['which', 'whether', 'who']
```

In [20]:
```
# 4. Create your own file consisting of words and (made up) frequencies, where each line consists of a wor
```

In [21]:

```
# Content of the file que4.txt is as follows:

'''
hello 5
tanay 5
nlp 3
goodbye 7
friday 6
'''
```

Out[21]:

```
'\nhello 5\ntanay 5\nnlp 3\ngoodbye 7\nfriday 6\n'
```

In [22]:

```
que4 = open('que4.txt', encoding = "utf-8").readlines()
print([[w, int(i)] for w, i in (rows.split() for rows in que4)])
```

```
[['hello', 5], ['tanay', 5], ['nlp', 3], ['goodbye', 7], ['friday', 6]]
```

In [23]:

```
#5. Readability measures are used to score the reading difficulty of a text, for the purposes of selecting
```

In [24]:

```
from nltk.corpus import brown
```

In [25]:

```
def ARI(category):
    words = brown.words(categories = category)
    sents = brown.sents(categories = category)
    μw = sum(len(w) for w in words)/len(words)
    μs = sum(len(s) for s in sents)/len(sents)
    return (4.71 * μw + 0.5 * μs - 21.43)
```

In [26]:

```
for category in brown.categories():
    print("The category is {0:<15} and its corresponding ARI is {1}". format(category, round(ARI(category)
```

```
The category is adventure       and its corresponding ARI is 4.08
The category is belles_lettres  and its corresponding ARI is 10.99
The category is editorial       and its corresponding ARI is 9.47
The category is fiction         and its corresponding ARI is 4.91
The category is government      and its corresponding ARI is 12.08
The category is hobbies         and its corresponding ARI is 8.92
The category is humor           and its corresponding ARI is 7.89
The category is learned         and its corresponding ARI is 11.93
The category is lore            and its corresponding ARI is 10.25
The category is mystery         and its corresponding ARI is 3.83
The category is news            and its corresponding ARI is 10.18
The category is religion        and its corresponding ARI is 10.2
The category is reviews         and its corresponding ARI is 10.77
The category is romance         and its corresponding ARI is 4.35
The category is science_fiction and its corresponding ARI is 4.98
```

In [27]:

```
# 6.Use the Porter Stemmer to normalize some tokenized text (see below), calling the stemmer on each word.
# text='Technologies based on NLP are becoming increasingly widespread. For example, phones and handheld c
```

In [28]:

```
porter = nltk.PorterStemmer()
lancaster = nltk.LancasterStemmer()
```

In [29]:

```
 text = 'Technologies based on NLP are becoming increasingly widespread. For example, phones and handheld
```

In [30]:

```
tokens = word_tokenize(text)
```

In [31]:

```
print([porter.stem(t) for t in tokens])
```

```
['technolog', 'base', 'on', 'nlp', 'are', 'becom', 'increasingli', 'widespread', '.', 'for', 'exampl', ',',
'phone', 'and', 'handheld', 'comput', 'support', 'predict', 'text', 'and', 'handwrit', 'recognit', ';', 'we
b', 'search', 'engin', 'give', 'access', 'to', 'inform', 'lock', 'up', 'in', 'unstructur', 'text', ';', 'ma
chin', 'translat', 'allow', 'us', 'to', 'retriev', 'text', 'written', 'in', 'chines', 'and', 'read', 'the
m', 'in', 'spanish', ';', 'text', 'analysi', 'enabl', 'us', 'to', 'detect', 'sentiment', 'in', 'tweet', 'an
d', 'blog', '.', 'By', 'provid', 'more', 'natur', 'human-machin', 'interfac', ',', 'and', 'more', 'sophis
t', 'access', 'to', 'store', 'inform', ',', 'languag', 'process', 'ha', 'come', 'to', 'play', 'a', 'centra
l', 'role', 'in', 'the', 'multilingu', 'inform', 'societi']
```

In [32]:

```
print([lancaster.stem(t) for t in tokens])
```

```
['technolog', 'bas', 'on', 'nlp', 'ar', 'becom', 'increas', 'widespread', '.', 'for', 'exampl', ',', 'pho
n', 'and', 'handheld', 'comput', 'support', 'predict', 'text', 'and', 'handwrit', 'recognit', ';', 'web',
'search', 'engin', 'giv', 'access', 'to', 'inform', 'lock', 'up', 'in', 'unstruct', 'text', ';', 'machin',
'transl', 'allow', 'us', 'to', 'retriev', 'text', 'writ', 'in', 'chines', 'and', 'read', 'them', 'in', 'spa
n', ';', 'text', 'analys', 'en', 'us', 'to', 'detect', 'senty', 'in', 'tweet', 'and', 'blog', '.', 'by', 'p
rovid', 'mor', 'nat', 'human-machine', 'interfac', ',', 'and', 'mor', 'soph', 'access', 'to', 'stor', 'info
rm', ',', 'langu', 'process', 'has', 'com', 'to', 'play', 'a', 'cent', 'rol', 'in', 'the', 'multil', 'infor
m', 'socy']
```

Explanation: Porter algorithm is the less aggressive as an algorithm. The stems of the words are somewhat intuitive and are understandable. On the other hand, Lancaster algorithm is very aggressive because of its strictly chopping words and making it much confusing. With this algorithm in use, the stems become non-relatable to some extent

In [33]:

```
# 7. Obtain raw texts from two or more genres and compute their respective reading difficulty scores as in
```

In [34]:

```
from nltk.corpus import abc
from nltk import word_tokenize, sent_tokenize
```

In [35]:

```
def ARI(category):
    words = word_tokenize(category)
    sents = [nltk.word_tokenize(sent) for sent in nltk.sent_tokenize(category)]
    µw = sum(len(w) for w in words)/len(words)
    µs = sum(len(s) for s in sents)/len(sents)
    return 4.71 * µw + 0.5 * µs - 21.43

print("ABC Rural News has an ARI of {0:>7}". format(round(ARI(abc.raw("rural.txt")),2)))
print("ABC Science News has an ARI of {0:>5}". format(round(ARI(abc.raw("science.txt")),2)))
```

```
ABC Rural News has an ARI of   12.62
ABC Science News has an ARI of 12.77
```

In [36]:

```
# 8.Rewrite the following nested loop as a nested list comprehension:
#  words = ['attribution', 'confabulation', 'elocution',
#          'sequoia', 'tenacious', 'unidirectional']
#  vsequences = set()
#  for word in words:
#      vowels = []
#      for char in word:
#          if char in 'aeiou':
#              vowels.append(char)
#      vsequences.add(''.join(vowels))
#  sorted(vsequences)
```

In [37]:

```
words = ['attribution', 'confabulation', 'elocution', 'sequoia', 'tenacious', 'unidirectional']
vsequences = set()
for word in words:
    vowels = []
    for char in word:
        if char in 'aeiou':
            vowels.append(char)
    vsequences.add(''.join(vowels))
sorted(vsequences)
```

Out[37]:

```
['aiuio', 'eaiou', 'eouio', 'euoia', 'oauaio', 'uiieioa']
```

In [38]:

```
# Re-writing it as a list comprehension:
# Method 1: Using a generic for loop for vowels and then on words array

words = ['attribution', 'confabulation', 'elocution', 'sequoia', 'tenacious', 'unidirectional']
sorted(set([''.join([w for w in word if w in 'aeiou']) for word in words]))
```

Out[38]:

```
['aiuio', 'eaiou', 'eouio', 'euoia', 'oauaio', 'uiieioa']
```

In [39]:

```
# Re-writing it as a list comprehension:
# Method 2: Using regex to identify vowels and then a for loop on words array

sorted(re.sub('[^aeiou]', '', w) for w in words)
```

Out[39]:

```
['aiuio', 'eaiou', 'eouio', 'euoia', 'oauaio', 'uiieioa']
```

In [41]:

```
# 9.Try to refer the following sample code to print the following sentences in a formatted way.(Hint: you
# output should look like:
'''
The Tragedie of Hamlet was written by William Shakespeare in 1599
Leaves of Grass        was written by Walt Whiteman       in 1855
Emma                   was written by Jane Austen         in 1816
# sample code:
template = 'Lee wants a {} right now'
menu = ['sandwich', 'spam fritter', 'pancake']
for snack in menu:
    print(template.format(snack))
'''
```

Out[41]:

```
"\nThe Tragedie of Hamlet was written by William Shakespeare in 1599\nLeaves of Grass        was written by
Walt Whiteman       in 1855\nEmma                   was written by Jane Austen         in 1816\n# sample co
de:\ntemplate = 'Lee wants a {} right now'\nmenu = ['sandwich', 'spam fritter', 'pancake']\nfor snack in me
nu:\n    print(template.format(snack))\n"
```

In [42]:

```
template = 'Lee wants a {} right now'
menu = ['sandwich', 'spam fritter', 'pancake']
for snack in menu:
    print(template.format(snack))
```

```
Lee wants a sandwich right now
Lee wants a spam fritter right now
Lee wants a pancake right now
```

In [43]:

```
# How will it work for the given sample:

menu = ['sandwich', 'spam fritter', 'pancake']
for i in range (len (menu)):
    print("{0:<11} {1:<12} {2}".format('Lee wants a', menu[i], 'right now'))
```

```
Lee wants a sandwich     right now
Lee wants a spam fritter right now
Lee wants a pancake      right now
```

In [44]:

```
# How will it work for the given question:

books = ['The Tragedie of Hamlet', 'Leaves of Grass', 'Emma']
author = ['William Shakespeare', 'Walt Whiteman', 'Jane Austen']
year = [1599, 1855, 1816]
for i in range (len (books)):
    print("{0:<22} was written by {1:<19} in {2}".format(books[i], author[i], year[i]))
```

```
The Tragedie of Hamlet was written by William Shakespeare in 1599
Leaves of Grass        was written by Walt Whiteman       in 1855
Emma                   was written by Jane Austen         in 1816
```

In [45]:

```
# 10. Define the variable quote to contain the list ['Action', 'speaks', 'louder', 'than', 'words']. Proce
```

In [46]:

```
quote = ['Action', 'speaks', 'louder', 'than', 'words']
lengths = [[len(i) for i in x.split()] for x in quote]
```

In [47]:

```
lengths
```

Out[47]:

```
[[6], [6], [6], [4], [5]]
```

In [48]:

```
# Also printing the arrays: quote and lengths using string formatting

for i in range (len (quote)):
    print("Length of the word {0:<6} is {1}".format(quote[i], lengths[i]))
```

```
Length of the word Action is [6]
Length of the word speaks is [6]
Length of the word louder is [6]
Length of the word than   is [4]
Length of the word words  is [5]
```