Common measures of textual complexity are derived from simple counts of words, sentences and syllables. In this homework, you'll implement two of them: type-token ratio (a measure of vocabulary richness) and the Flesch-Kincaid Grade Level.

In [1]:
```
import nltk, re, warnings
```

In [2]:
```
# If you haven't downloaded the sentence segmentation model before, do so here
nltk.download("punkt")
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\12062\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```
Out[2]:
```
True
```

In [3]:
```
warnings.filterwarnings('ignore')
```

Q1: Find two different texts you'd like to compare (from any source). For potential sources, see the The American Presidency Project for all state of the union addresses and Project Gutenberg for books in the public domain. Paste them in the `text1` and `text2` strings below. Ensure that both texts are a minimum of 500 words.

In [4]:
```
text1=""" “Let your mind become a lens, thanks to the converging rays of attention; let
absorbing idea.” This advice comes from Antonin-Dalmace Sertillanges, a Dominican friar
penned a slim but influential volume titled The Intellectual Life. Sertillanges wrote t
make a living in the world of ideas. Throughout The Intellectual Life, Sertillanges rec
reader for this challenge. For this reason, his book proves useful in our quest to bett
To understand Sertillanges’s advice, let’s return to the quote from earlier. In these w
that to advance your understanding of your field you must tackle the relevant topics sy
in each. In other words, he teaches: To learn requires intense concentration . This ide
Sertillanges uncovered a fact about mastering cognitively demanding tasks that would ta
This task of formalization began in earnest in the 1970s, when a branch of psychology,
explore what separates experts (in many different fields) from everyone else. In the ea
together these strands into a single coherent answer, consistent with the growing resea
Ericsson opens his seminal paper on the topic with a powerful claim: “We deny that thes
Instead, we argue that the differences between expert performers and normal adults refl
domain.” American culture, in particular, loves the storyline of the prodigy (“Do you k
Will Hunting as he makes quick work of proofs that stymie the world’s top mathematician
accepted (with caveats *), de-stabilizes these myths. To master a cognitively demanding
made for natural talent. (On this point too, Sertillanges seems to have been ahead of h
by bringing all their power to bear on the point on which they had decided to show thei
This brings us to the question of what deliberate practice actually requires. Its core
specific skill you’re trying to improve or an idea you’re trying to master; (2) you rec
where it’s most productive. The first component is of particular importance to our disc
and that it instead requires uninterrupted concentration. As Ericsson emphasizes, “Diff
deliberate practice” (emphasis mine). """
```

In [5]:
```
text2="""The deified Augustus, to whom the gods granted more than to anyone else, never
and to seek a respite from public affairs. Everything he said always reverted to this t
He used to beguile his labours with this consolation, sweet though false, that one day
himself. In a letter he wrote to the senate, after he promised that his rest would not
nor inconsistent with his former glory, I find these words: 'But it is more impressive
than to promise them. Nevertheless, since the delightful reality is still a long way of
```

```
desired time has led me to anticipate some of its delight by the pleasure arising from
leisure seem to him that because he could not enjoy it in actuality, he did so mentally
that everything depended on himself alone, who decided the fortune of individuals and n
when thinking of that day on which he would lay aside his own greatness. He knew from e
sweat those blessings gleaming through every land cost him, how many secret anxieties t
forced to fight first with his fellow-countrymen, then with his colleagues, and finally
shedding blood on land and sea. Driven to fight in Macedonia,
Sicily, Egypt, Syria, Asia - almost every country - he turned his armies against foreig
were tired of shedding Roman blood. While he was establishing peace in the Alps and sub
established in the middle of his peaceful empire; while he was extending his boundaries
the Euphrates and the Danube, at Rome itself Murena, Caepio, Lepidus, Egnatius and othe
their swords against him. Nor had he yet escaped their plots when his daughter and all
to her by adultery as though by an oath kept alarming his feeble old age, as did Iullus
woman linked to an Antony. He cut away these ulcers, limbs and all, but others took the
with a surfeit of blood which is always subject to a haemorrhage somewhere. So he longe
as his hopes and thoughts dwelt on that he found relief for his labours: this was the p
grant the prayers of mankind. When Marcus Cicero was cast among men like Cati line and
them undisguised enemies and some doubtful friends - when he was tossed about in the st
state, he tried to hold it steady as it went to its doom; but at last he was swept away
prosperity nor patience in adversity, and how often does he curse that very consulship,
without ceasing though not without good reason! What woeful words he uses in a letter t
elder Pompey had been conquered, and his son was still trying to revive his defeated fo
want to know,' he said, 'what I am doing here? I am staying a semi-prisoner in my Tuscu
goes on to bewail his former life, to complain of the present, and to despair of the fu
a semi-prisoner, but really and truly the wise man will never go so far as to use such
never be a semi-prisoner, but will always enjoy freedom which is solid and complete, at
master and higher than all others. For what can be above the man who is above fortune?
```

Q2: Use the `nltk.word_tokenize` method to implement the type-token ratio:

$$TTR = \frac{\text{number of distinct word types}}{\text{number of word tokens}}$$

TTR is dependent on text length (intuitively, the longer a text is, the greater chance you have of a word type repeating), so this number is only comparable between documents of identical lengths. Calculate this measure for the first 500 words of your two documents and report the results here. Exclude tokens that are exclusively punctuation from all counts.

In [6]:
```
test_line = " Driven to fight in Macedonia, Sicily, Egypt, Syria, Asia - almost every c
```

In [7]:
```
#remove punctuaiton from string
def remove_punc(text):

    # initializing punctuations string
    punc = '''!()-[]{};:'"\,<>./?@#$%^&*_~'''

    #remove punc if see one
    for i in text:
        if i in punc:
            text = text.replace(i, '')

    return text
```

In [8]:
```
#get the number of words
def count_num_words(text):
```

```
    line_list = []
    words = text.split()

    for i in words:
        line_list.append(i)

    return len(line_list), line_list
```

In [9]:

```
#get the number of distinct words (type)
def count_type(text):
    dist_word_list = []
    words = text.split()

    for i in words:
        if i not in dist_word_list:
            dist_word_list.append(i)

    return len(dist_word_list), dist_word_list
```

In [10]:

```
#count number of tokens

def count_token(text):

    token = nltk.word_tokenize(text, language= 'english')

    return len(token)
```

In [11]:

```
def type_token_ratio(text, num_words=500):
    # your answer here

    #Lower case the text and remove punctuations, and strip it to length of 500
    text_no_punc = remove_punc(text.lower())
    text_here = text_no_punc[:num_words]

    #get the number of type and tokens
    num_type, text_list = count_type(text_here)
    num_token = count_token(text_here)

    return (num_type/num_token)
```

In [12]:

```
type_token_ratio(test_line)
```

Out[12]:

```
0.7590361445783133
```

In [13]:

```
type_token_ratio(text1)
```

Out[13]:

```
0.7303370786516854
```

In [14]:

```
type_token_ratio(text2)
```

Out[14]:

```
0.7311827956989247
```

Now we'll implement the Flesch-Kincaid Grade Level, which has the following formula:

$$0.39\left(\frac{\text{total words}}{\text{total sentences}}\right)+11.8\left(\frac{\text{total syllables}}{\text{total words}}\right)-15.59$$

Use `nltk.sent_tokenize` or spacy's `sents` function for counting the number of sentences, any word tokenization method we've covered for counting the number of words, and the `get_syllable_count` function below for counting the number of syllables in a word. Exclude tokens that are exclusively punctuation from word and syllable counts.

For calculating the syllables, we're going to use a number of resources: the CMU pronunciation dictionary, which lists the ARPABET pronunciation for a list of words, along with g2p, a neural model trained to predict the pronunciation for words (which we can use for words not in the CMU dictionary).

In [15]:
```
arpabet = nltk.corpus.cmudict.dict()
```

In [16]:
```
!pip install g2p_en
```

```
Requirement already satisfied: g2p_en in c:\users\12062\anaconda3\envs\anlp\lib\site-pac
kages (2.1.0)
Requirement already satisfied: distance>=0.1.3 in c:\users\12062\anaconda3\envs\anlp\lib
\site-packages (from g2p_en) (0.1.3)
Requirement already satisfied: nltk>=3.2.4 in c:\users\12062\anaconda3\envs\anlp\lib\sit
e-packages (from g2p_en) (3.6.2)
Requirement already satisfied: numpy>=1.13.1 in c:\users\12062\anaconda3\envs\anlp\lib\s
ite-packages (from g2p_en) (1.20.3)
Requirement already satisfied: inflect>=0.3.1 in c:\users\12062\anaconda3\envs\anlp\lib
\site-packages (from g2p_en) (5.3.0)
Requirement already satisfied: tqdm in c:\users\12062\anaconda3\envs\anlp\lib\site-packa
ges (from nltk>=3.2.4->g2p_en) (4.62.1)
Requirement already satisfied: click in c:\users\12062\anaconda3\envs\anlp\lib\site-pack
ages (from nltk>=3.2.4->g2p_en) (8.0.1)
Requirement already satisfied: joblib in c:\users\12062\anaconda3\envs\anlp\lib\site-pac
kages (from nltk>=3.2.4->g2p_en) (1.0.1)
Requirement already satisfied: regex in c:\users\12062\anaconda3\envs\anlp\lib\site-pack
ages (from nltk>=3.2.4->g2p_en) (2021.8.3)
Requirement already satisfied: colorama in c:\users\12062\anaconda3\envs\anlp\lib\site-p
ackages (from click->nltk>=3.2.4->g2p_en) (0.4.4)
```

In [17]:
```
from g2p_en import G2p
g2p = G2p()
```

In [18]:
```
def get_pronunciation(word):
    if word in arpabet:
        # pick the first pronunciation
        return arpabet[word][0]

    else:
        return g2p(word)

def get_syllable_count(word):
    pronunciation=get_pronunciation(word)
```

```
    sylls=0
    for phon in pronunciation:
        # vowels in arpabet end in digits (indicating stress)
        if re.search("\d$", phon) is not None:
            sylls+=1
    return sylls
```

In [19]:
```
get_syllable_count("Bamman")
```

Out[19]: 2

In [20]:
```
get_syllable_count("The Australian platypus is seemingly a hybrid of a mammal and repti
```

Out[20]: 24

In [21]:
```
#counting number of sentece
def count_sent(line):
    return len(nltk.sent_tokenize(line))
```

Q3. Implement Flesch-Kincaid Grade Level and report its results for your two texts. Flesch-Kincaid relies on an implicit definition of a "word" and a "sentence", and different definitions will yield different grade level estimates. (In the problem definition above, we've already ruled out punctuation as constituing stand-alone words, and other assumptions lurk with every tokenization method). State your assumptions for the definition of "word" you have implemented and why they are reasonable.

In [30]:
```
def flesch_kincaid_grade_level(text):
    # your answer here
    total_sent = count_sent(text)

    text_new = remove_punc(text.lower())
    total_words, text_list = count_num_words(text_new)

    total_syll = get_syllable_count(text_new)

    return 0.39 * (total_words / total_sent) + 11.8 * (total_syll/total_words) - 15.59
```

In [31]:
```
# Should be 11.265
flesch_kincaid_grade_level("The Australian platypus is seemingly a hybrid of a mammal a
```

Out[31]: 11.264615384615386

In [32]:
```
flesch_kincaid_grade_level(text1)
```

Out[32]: 17.022581081081082

In [33]:
```
flesch_kincaid_grade_level(text2)
```

12.809655172413795

Out[33]:

Q3 "word" assumptions:

Words like "I'm", "let's" counts as 1 word, since eliminated the punctuation we are concatenating the words together as 1 word.

In [ ]: