Preamble: A Brand New Jay

After an eventful season on season 8 of *A Brand New Jay*, the 3 remaining contestants were invited to Jay Stacksby's private island for the last three episodes. When the day of filming the finale came Mr. Stacksby was found with one of his Professional Series 8-inch Chef Knives plunged through his heart! After the initial investigation highlighted that the film crew all lived in a separate house on the other side of the island, it was concluded that only the three contestants were near enough to Stacksby in order to commit a crime. At the scene of the crime, a letter was left. Here are the contents of that letter:

> You may call me heartless, a killer, a monster, a murderer, but I'm still NOTHING compared to the villian that Jay was. This whole contest was a sham, an elaborate plot to shame the contestants and feed Jay's massive, massive ego. SURE you think you know him! You've seen him smiling for the cameras, laughing, joking, telling stories, waving his money around like a prop but off camera he was a sinister beast, a cruel cruel taskmaster, he treated all of us like slaves, like cattle, like animals! Do you remember Lindsay, she was the first to go, he called her such horrible things that she cried all night, keeping up all up, crying, crying, and more crying, he broke her with his words. I miss my former cast members, all of them very much. And we had to live with him, live in his home, live in his power, deal with his crazy demands. AND FOR WHAT! DID YOU KNOW THAT THE PRIZE ISN'T REAL? He never intended to marry one of us! The carrot on the stick was gone, all that was left was stick, he told us last night that we were all a terrible terrible disappointment and none of us would ever amount to anything, and that regardless of who won the contest he would never speak to any of us again! It's definitely the things like this you can feel in your gut how wrong he is! Well I showed him, he got what he deserved all right, I showed him, I showed him the person I am! I wasn't going to be pushed around any longer, and I wasn't going to let him go on pretending that he was some saint when all he was was a sick sick twisted man who deserved every bit of what he got. The fans need to know, Jay Stacksby is a vile amalgamation of all things evil and bad and the world is a better place without him.

Pretty sinister stuff! Luckily, in addition to this bold-faced admission, we have the introduction letters of the three contestants. Maybe there is a way to use this information to determine who the author of this murder letter is?

Myrtle Beech's introduction letter:

> Salutations. My name? Myrtle. Myrtle Beech. I am a woman of simple tastes. I enjoy reading, thinking, and doing my taxes. I entered this competition because I want a serious relationship. I want a commitment. The last man I dated was too whimsical. He wanted to go on dates that had no plan. No end goal. Sometimes we would just end up wandering the streets after dinner. He called it a "walk". A "walk" with no destination. Can you imagine? I like every action I take to have a measurable effect. When I see a movie, I like to walk away with insights that I did not have before. When I take a bike ride, there better be a worthy destination at the end of the bike path. Jay seems frivolous at times. This worries me. However, it is my staunch belief that one does not make and keep money without having a modicum of discipline. As such, I am hopeful. I will now list three things I cannot live without. Water. Emery boards. Dogs. Thank you for the opportunity to introduce myself. I look forward to the competition.

Lily Trebuchet's introduction letter:

> Hi, I'm Lily Trebuchet from East Egg, Long Island. I love cats, hiking, and curling up under a warm blanket with a book. So they gave this little questionnaire to use for our bios so lets get started. What are some of my least favorite household chores? Dishes, oh yes it's definitely the dishes, I just hate doing them, don't you? Who is your favorite actor and why? Hmm, that's a hard one, but I think recently I'll have to go with Michael B. Jordan, every bit of that man is handsome, HANDSOME! Do you remember seeing him shirtless? I can't believe what he does for the cameras! Okay okay next question, what is your perfect date? Well it starts with a nice dinner at a delicious but small restaurant, you know like one of those places where the owner is in the back and comes out to talk to you and ask you how your meal was. My favorite form of art? Another hard one, but I think I'll have to go with music, music you can feel in your whole body and it is electrifying and best of all, you can dance to it! Okay final question, let's see, What are three things you cannot live without? Well first off, my beautiful, beautiful cat Jerry, he is my heart and spirit animal. Second is pasta, definitely pasta, and the third I think is my family, I love all of them very much and they support me in everything I do. I know Jay Stacksby is a handsome man and all of us want to be the first to walk down the aisle with him, but I think he might truly be the one for me. Okay that's it for the bio, I hope you have fun watching the show!

Gregg T Fishy's introduction letter:

A good day to you all, I am Gregg T Fishy, of the Fishy Enterprise fortune. I am 37 years young. An adventurous spirit and I've never lost my sense of childlike wonder. I do love to be in the backyard gardening and I have the most extraordinary time when I'm fishing. Fishing for what, you might ask? Why, fishing for compliments of course! I have a stunning pair of radiant blue eyes. They will pierce the soul of anyone who dare gaze upon my countenance. I quite enjoy going on long jaunts through garden paths and short walks through greenhouses. I hope that Jay will be as absolutely interesting as he appears on the television. I find that he has some of the most curious tastes in style and humor. When I'm out and about I quite enjoy hearing tales that instill in my heart of hearts the fascination that beguiles my every day life. Every fiber of my being scintillates and vascillates with extreme pleasure during one of these charming anecdotes and significantly pleases my beautiful personage. I cannot wait to enjoy being on A Brand New Jay. It certainly seems like a grand time to explore life and love.

Saving The Different Examples as Variables

First let's create variables to hold the text data in! Save the muder note as a string in a variable called `murder_note`. Save Lily Trebuchet's introduction into `lily_trebuchet_intro`. Save Myrtle Beech's introduction into `myrtle_beech_intro`. Save Gregg T Fishy's introduction into `gregg_t_fishy_intro`.

In [17]:
```
murder_note = """You may call me heartless, a killer, a monster, a mur
derer, but I'm still NOTHING compared to the villian that Jay was. Thi
s whole contest was a sham, an elaborate plot to shame the contestants
and feed Jay's massive, massive ego. SURE you think you know him! You'
ve seen him smiling for the cameras, laughing, joking, telling stories
, waving his money around like a prop but off camera he was a sinister
beast, a cruel cruel taskmaster, he treated all of us like slaves, lik
e cattle, like animals! Do you remember Lindsay, she was the first to
go, he called her such horrible things that she cried all night, keepi
ng up all up, crying, crying, and more crying, he broke her with his w
ords. I miss my former cast members, all of them very much. And we had
to live with him, live in his home, live in his power, deal with his c
razy demands. AND FOR WHAT! DID YOU KNOW THAT THE PRIZE ISN'T REAL? He
never intended to marry one of us! The carrot on the stick was gone, a
ll that was left was stick, he told us last night that we were all a t
errible terrible disappointment and none of us would ever amount to an
ything, and that regardless of who won the contest he would never spea
k to any of us again! It's definitely the things like this you can fee
l in your gut how wrong he is! Well I showed him, he got what he deser
ved all right, I showed him, I showed him the person I am! I wasn't go
ing to be pushed around any longer, and I wasn't going to let him go o
n pretending that he was some saint when all he was was a sick sick tw
isted man who deserved every bit of what he got. The fans need to know
, Jay Stacksby is a vile amalgamation of all things evil and bad and t
he world is a better place without him."""
```

In [18]:
```
lily_trebuchet_intro = """Hi, I'm Lily Trebuchet from East Egg, Long I
sland. I love cats, hiking, and curling up under a warm blanket with a
book. So they gave this little questionnaire to use for our bios so le
ts get started. What are some of my least favorite household chores? D
ishes, oh yes it's definitely the dishes, I just hate doing them, don'
t you? Who is your favorite actor and why? Hmm, that's a hard one, but
I think recently I'll have to go with Michael B. Jordan, every bit of
that man is handsome, HANDSOME! Do you remember seeing him shirtless?
I can't believe what he does for the cameras! Okay okay next question,
what is your perfect date? Well it starts with a nice dinner at a deli
cious but small restaurant, you know like one of those places where th
e owner is in the back and comes out to talk to you and ask you how yo
ur meal was. My favorite form of art? Another hard one, but I think I'
ll have to go with music, music you can feel in your whole body and it
is electrifying and best of all, you can dance to it! Okay final quest
ion, let's see, What are three things you cannot live without? Well fi
rst off, my beautiful, beautiful cat Jerry, he is my heart and spirit
animal. Second is pasta, definitely pasta, and the third I think is my
family, I love all of them very much and they support me in everything
I do. I know Jay Stacksby is a handsome man and all of us want to be t
he first to walk down the aisle with him, but I think he might truly b
e the one for me. Okay that's it for the bio, I hope you have fun watc
hing the show!"""
```

In [19]:
```
myrtle_beech_intro = """Salutations. My name? Myrtle. Myrtle Beech. I
am a woman of simple tastes. I enjoy reading, thinking, and doing my t
axes. I entered this competition because I want a serious relationship
. I want a commitment. The last man I dated was too whimsical. He want
ed to go on dates that had no plan. No end goal. Sometimes we would ju
st end up wandering the streets after dinner. He called it a \"walk".
A "walk" with no destination. Can you imagine? I like every action I t
ake to have a measurable effect. When I see a movie, I like to walk aw
ay with insights that I did not have before. When I take a bike ride,
there better be a worthy destination at the end of the bike path. Jay
seems frivolous at times. This worries me. However, it is my staunch b
elief that one does not make and keep money without having a modicum o
f discipline. As such, I am hopeful. I will now list three things I ca
nnot live without. Water. Emery boards. Dogs. Thank you for the opport
unity to introduce myself. I look forward to the competition."""
```

In [20]:
```
gregg_t_fishy_intro = """A good day to you all, I am Gregg T Fishy, of
the Fishy Enterprise fortune. I am 37 years young. An adventurous spir
it and I've never lost my sense of childlike wonder. I do love to be i
n the backyard gardening and I have the most extraordinary time when I
'm fishing. Fishing for what, you might ask? Why, fishing for complime
nts of course! I have a stunning pair of radiant blue eyes. They will
pierce the soul of anyone who dare gaze upon my countenance. I quite e
njoy going on long jaunts through garden paths and short walks through
greenhouses. I hope that Jay will be as absolutely interesting as he a
ppears on the television. I find that he has some of the most curious
tastes in style and humor. When I'm out and about I quite enjoy hearin
g tales that instill in my heart of hearts the fascination that beguil
es my every day life. Every fiber of my being scintillates and vascill
ates with extreme pleasure during one of these charming anecdotes and
significantly pleases my beautiful personage. I cannot wait to enjoy b
eing on A Brand New Jay. It certainly seems like a grand time to explo
re life and love."""
```

The First Indicator: Sentence Length

Perhaps some meaningful data can first be gleaned from these text examples if we measure how long the average sentence length is. Different authors have different patterns of written speech, so this could be very useful in tracking down the killer.

Write a function `get_average_sentence_length` that takes some `text` as an argument. This function should return the average number of words in a sentence in the text.

Hint (highlight this hint in order to reveal it): Use your knowledge of *string methods* to create a list of all of the sentences in a text, called **sentences_in_text**. Further break up each **sentences_in_text** into a list of words and save the *length* of that list of words to a new list that contains all the sentence lengths, called **sentence_lengths**. Take the average of all of the sentence lengths by adding them all together and dividing by the number of sentences (which should be the same as the length of the **sentence_lengths**). Remember sentences can end with more than one kind of punctuation, you might find it easiest to use **.replace()** so you only have to split on one punctuation mark. Remember **.replace()** doesn't modify the string itself, it returns a new string!

In [21]:
```
def get_average_sentence_length(text):
    #Variables
    number_of_words = 0
    number_of_sentences = 0
    avg_sentence_lengths = 0
    words = []

    #Replace (!,?,","," " and then split at ". "), Create list of all
sentences in text
    sentences_in_text = text.replace("?",".").replace("!",".").replace
("  "," ")
    sentences_in_text = sentences_in_text.split(". ")

    #Split at each word, " ", in the sentences
    for word in sentences_in_text:
        words.append(word.split(" "))

    #Sentence length in words
    for word in words:
        number_of_words = number_of_words + len(word)

    #Average number of words per sentence
    number_of_sentences = len(words)
    avg_sentence_lengths = int(number_of_words / number_of_sentences)
    return avg_sentence_lengths

#Testing each note
print(get_average_sentence_length(murder_note))
print(get_average_sentence_length(myrtle_beech_intro))
print(get_average_sentence_length(lily_trebuchet_intro))
print(get_average_sentence_length(gregg_t_fishy_intro))
```

```
22
6
15
13
```

Creating The Definition for Our Model

Now that we have a metric we want to save and data that is coupled with that metric, it might be time to create our data type. Let's define a class called `TextSample` with a constructor. The constructor should take two arguments: `text` and `author`. `text` should be saved as `self.raw_text`. Call `get_average_sentence_length` with the raw text and save it to `self.average_sentence_length`. You should save the author of the text as `self.author`.

Additionally, define a string representation for the model. If you print a `TextSample` it should render:

- The author's name
- The average sentence length

This will be your main class for the problem at hand. All later instruction to update `TextSample` should be done in the code block below. After updating `TextSample`, click on the `Cell` option in the Jupyter Notebook main menu above, then click `Run All` to rerun the cells from top to bottom. If you need to restart your Jupyter Notebook either run the cells below first or move the `TextSample` class definition & instantiation cells to the bottom.

In [22]:
```
#Create class to start saving data coupled with specific metrics
class TextSample():
#Set constructors
    def __init__(self, text, author):
        self.raw_text = text
        self.average_sentence_length = get_average_sentence_length(sel
f.raw_text)
        self.author = author
        self.prepared_text = prepare_text(text)
        self.word_count_frequency = build_frequency_table(prepare_text
(text))
        self.n_gram_frequency = build_frequency_table(bigram_creator(p
repare_text(text)))
#String representation
    def __str__(self):
        return "The Author's name is: {} \nAnd the average sentence le
ngth is: {}.\n".format(self.author, self.average_sentence_length)
```

Creating our TextSample Instances

Now create a `TextSample` object for each of the samples of text that we have.

- `murderer_sample` for the murderer's note.
- `lily_sample` for Lily Trebuchet's note.
- `myrtle_sample` for Myrtle Beech's note.
- `gregg_sample` for Gregg T Fishy's note.

Print out each one after instantiating them.

In [23]:
```
#Creating objects from TextSample class
murderer_sample = TextSample(murder_note, "murderer")
lily_sample = TextSample(lily_trebuchet_intro, "Lily Trebuchet")
myrtle_sample = TextSample(myrtle_beech_intro, "Myrtle Beech")
gregg_sample = TextSample(gregg_t_fishy_intro, "Gregg T Fishy")

#Testing each of the objects
print(murderer_sample)
print(lily_sample)
print(myrtle_sample)
print(gregg_sample)
```

```
The Author's name is: murderer
And the average sentence length is: 22.

The Author's name is: Lily Trebuchet
And the average sentence length is: 15.

The Author's name is: Myrtle Beech
And the average sentence length is: 6.

The Author's name is: Gregg T Fishy
And the average sentence length is: 13.
```

Cleaning Our Data

We want to compare the word choice and usage between the samples, but sentences make our text data fairly messy. In order to analyze the different messages fairly, we'll need to remove all the punctuation and uppercase letters from the samples.

Create a function called `prepare_text` that takes a single parameter `text`, makes the text entirely lowercase, removes all the punctuation and returns a list of the words in the text in order.

For example: `"Where did you go, friend? We nearly saw each other."` would become `['where', 'did', 'you', 'go', 'friend', 'we', 'nearly', 'saw', 'each', 'other']`.

In [24]:
```
#Cleaning up text to ensure it is lowercase, and organized appropriate
ly for the word comparisons portion of exercise
def prepare_text(text):
    stripped_words = []
    stripped_words = text.replace(",",".").replace("!",".").replace(".
","").replace("?","").lower().split(" ")
    return stripped_words

#Testing the object output
print(prepare_text(myrtle_beech_intro))
```

```
['salutations', 'my', 'name', 'myrtle', 'myrtle', 'beech', 'i', 'am'
, 'a', 'woman', 'of', 'simple', 'tastes', 'i', 'enjoy', 'reading', '
thinking', 'and', 'doing', 'my', 'taxes', 'i', 'entered', 'this', 'c
ompetition', 'because', 'i', 'want', 'a', 'serious', 'relationship',
'i', 'want', 'a', 'commitment', 'the', 'last', 'man', 'i', 'dated',
'was', 'too', 'whimsical', 'he', 'wanted', 'to', 'go', 'on', 'dates'
, 'that', 'had', 'no', 'plan', 'no', 'end', 'goal', 'sometimes', 'we
', 'would', 'just', 'end', 'up', 'wandering', 'the', 'streets', 'aft
er', 'dinner', 'he', 'called', 'it', 'a', '"walk"', 'a', '"walk"', '
with', 'no', 'destination', 'can', 'you', 'imagine', 'i', 'like', 'e
very', 'action', 'i', 'take', 'to', 'have', 'a', 'measurable', 'effe
ct', 'when', 'i', 'see', 'a', 'movie', 'i', 'like', 'to', 'walk', 'a
way', 'with', 'insights', 'that', 'i', 'did', 'not', 'have', 'before
', 'when', 'i', 'take', 'a', 'bike', 'ride', 'there', 'better', 'be'
, 'a', 'worthy', 'destination', 'at', 'the', 'end', 'of', 'the', 'bi
ke', 'path', 'jay', 'seems', 'frivolous', 'at', 'times', 'this', 'wo
rries', 'me', 'however', 'it', 'is', 'my', 'staunch', 'belief', 'tha
t', 'one', 'does', 'not', 'make', 'and', 'keep', 'money', 'without',
'having', 'a', 'modicum', 'of', 'discipline', 'as', 'such', 'i', 'am
', 'hopeful', 'i', 'will', 'now', 'list', 'three', 'things', 'i', 'c
annot', 'live', 'without', 'water', 'emery', 'boards', 'dogs', 'than
k', 'you', 'for', 'the', 'opportunity', 'to', 'introduce', 'myself',
'i', 'look', 'forward', 'to', 'the', 'competition']
```

Update the constructor for `TextSample` to save the prepared text as `self.prepared_text`.

Building A Frequency Table

Now we want to see which words were most frequently used in each of the samples. Create a function called `build_frequency_table`. It takes in a list called `corpus` and creates a dictionary called `frequency_table`. For every element in `corpus` the value `frequency_table[element]` should be equal to the number of times that element appears in `corpus`. For example the input `['do', 'you', 'see', 'what', 'i', 'see']` would create the frequency table `{'what': 1, 'you': 1, 'see' 2, 'i': 1}`.

In [25]:
```
#Creates dictionary to determine freqency of words in persons intro/mu
rder note
def build_frequency_table(corpus):
    frequency_table = {}
    counter = 0
    fresh_words = []
    total_count = []

#Loops text to add new words to fresh_words if not already there
    for element in corpus:
        if element not in fresh_words:
            fresh_words.append(element)

#Loops fresh_words, and within loops corpus, to add the occurences tog
ether in the counter. Counter reset after each word.
    for element in fresh_words:
        for elements in corpus:
            if element == elements:
                counter += 1
        total_count.append(counter)
        counter = 0

#Create dictionary to organize data for smoother processing
    frequency_table = dict(zip(fresh_words, total_count))
    return frequency_table

#Testing each note
print(build_frequency_table(prepare_text(gregg_t_fishy_intro)))
```

```
{'a': 4, 'good': 1, 'day': 2, 'to': 4, 'you': 2, 'all': 1, 'i': 10,
'am': 2, 'gregg': 1, 't': 1, 'fishy': 2, 'of': 9, 'the': 7, 'enterpr
ise': 1, 'fortune': 1, '37': 1, 'years': 1, 'young': 1, 'an': 1, 'ad
venturous': 1, 'spirit': 1, 'and': 8, "i've": 1, 'never': 1, 'lost':
1, 'my': 6, 'sense': 1, 'childlike': 1, 'wonder': 1, 'do': 1, 'love'
: 2, 'be': 2, 'in': 3, 'backyard': 1, 'gardening': 1, 'have': 2, 'mo
st': 2, 'extraordinary': 1, 'time': 2, 'when': 2, "i'm": 2, 'fishing
': 3, 'for': 2, 'what': 1, 'might': 1, 'ask': 1, 'why': 1, 'complime
nts': 1, 'course': 1, 'stunning': 1, 'pair': 1, 'radiant': 1, 'blue'
: 1, 'eyes': 1, 'they': 1, 'will': 2, 'pierce': 1, 'soul': 1, 'anyon
e': 1, 'who': 1, 'dare': 1, 'gaze': 1, 'upon': 1, 'countenance': 1,
'quite': 2, 'enjoy': 3, 'going': 1, 'on': 3, 'long': 1, 'jaunts': 1,
'through': 2, 'garden': 1, 'paths': 1, 'short': 1, 'walks': 1, 'gree
nhouses': 1, 'hope': 1, 'that': 4, 'jay': 2, 'as': 2, 'absolutely':
1, 'interesting': 1, 'he': 2, 'appears': 1, 'television': 1, 'find':
1, 'has': 1, 'some': 1, 'curious': 1, 'tastes': 1, 'style': 1, 'humo
r': 1, 'out': 1, 'about': 1, 'hearing': 1, 'tales': 1, 'instill': 1,
'heart': 1, 'hearts': 1, 'fascination': 1, 'beguiles': 1, 'every': 2
, 'life': 2, 'fiber': 1, 'being': 2, 'scintillates': 1, 'vascillates
': 1, 'with': 1, 'extreme': 1, 'pleasure': 1, 'during': 1, 'one': 1,
'these': 1, 'charming': 1, 'anecdotes': 1, 'significantly': 1, 'plea
ses': 1, 'beautiful': 1, 'personage': 1, 'cannot': 1, 'wait': 1, 'br
and': 1, 'new': 1, 'it': 1, 'certainly': 1, 'seems': 1, 'like': 1, '
grand': 1, 'explore': 1}
```

The Second Indicator: Favorite Words

Use `build_frequency_table` with the prepared text to create a frequency table that counts how frequently all the words in each text sample appears. Call these functions in the constructor for `TextSample` and assign the word frequency table to a value called `self.word_count_frequency` .

The Third Indicator: N-Grams

An n-gram (https://en.wikipedia.org/wiki/N-gram) is a text analysis technique used for pattern recognition and applicable throughout lingusitics. We're going to use n-grams to find who uses similar word-pairs to the murderer, and we think it's going to make our evidence strong enough to conclusively find the killer.

Create a function called `ngram_creator` that takes a parameter `text_list` , a treated in-order list of the words in a text sample. `ngram_creator` should return a list of all adjacent pairs of words, styled as strings with a space in the center.

For instance, calling `ngram_creator` with the input `['what', 'in', 'the', 'world', 'is', 'going', 'on']` Should produce the output `['what in', 'in the', 'the world', 'world is', 'is going', 'going on']` .

These are two-word n-grams.

In [26]:
```
#Creates new "sentences" with 2 words each by iterating through the pr
epared version of notes
def bigram_creator(text_list):
    bigram_list = []
    for i in range(len(text_list)-1):
        bigram_list.append((text_list[i] + " " + text_list[i+1]))
    return bigram_list

#Testing one note for good measure
bigram_creator(prepare_text(gregg_t_fishy_intro))
```

Out[26]:
```
['a good',
 'good day',
 'day to',
 'to you',
 'you all',
 'all i',
 'i am',
 'am gregg',
 'gregg t',
 't fishy',
 'fishy of',
 'of the',
 'the fishy',
 'fishy enterprise',
 'enterprise fortune',
 'fortune i',
 'i am',
 'am 37',
 '37 years',
```

```
'years young',
'young an',
'an adventurous',
'adventurous spirit',
'spirit and',
"and i've",
"i've never",
'never lost',
'lost my',
'my sense',
'sense of',
'of childlike',
'childlike wonder',
'wonder i',
'i do',
'do love',
'love to',
'to be',
'be in',
'in the',
'the backyard',
'backyard gardening',
'gardening and',
'and i',
'i have',
'have the',
'the most',
'most extraordinary',
'extraordinary time',
'time when',
"when i'm",
"i'm fishing",
'fishing fishing',
'fishing for',
'for what',
'what you',
'you might',
'might ask',
'ask why',
'why fishing',
'fishing for',
'for compliments',
'compliments of',
'of course',
'course i',
'i have',
'have a',
'a stunning',
'stunning pair',
'pair of',
```

```
'of radiant',
'radiant blue',
'blue eyes',
'eyes they',
'they will',
'will pierce',
'pierce the',
'the soul',
'soul of',
'of anyone',
'anyone who',
'who dare',
'dare gaze',
'gaze upon',
'upon my',
'my countenance',
'countenance i',
'i quite',
'quite enjoy',
'enjoy going',
'going on',
'on long',
'long jaunts',
'jaunts through',
'through garden',
'garden paths',
'paths and',
'and short',
'short walks',
'walks through',
'through greenhouses',
'greenhouses i',
'i hope',
'hope that',
'that jay',
'jay will',
'will be',
'be as',
'as absolutely',
'absolutely interesting',
'interesting as',
'as he',
'he appears',
'appears on',
'on the',
'the television',
'television i',
'i find',
'find that',
'that he',
```

```
'he has',
'has some',
'some of',
'of the',
'the most',
'most curious',
'curious tastes',
'tastes in',
'in style',
'style and',
'and humor',
'humor when',
"when i'm",
"i'm out",
'out and',
'and about',
'about i',
'i quite',
'quite enjoy',
'enjoy hearing',
'hearing tales',
'tales that',
'that instill',
'instill in',
'in my',
'my heart',
'heart of',
'of hearts',
'hearts the',
'the fascination',
'fascination that',
'that beguiles',
'beguiles my',
'my every',
'every day',
'day life',
'life every',
'every fiber',
'fiber of',
'of my',
'my being',
'being scintillates',
'scintillates and',
'and vascillates',
'vascillates with',
'with extreme',
'extreme pleasure',
'pleasure during',
'during one',
'one of',
```

```
'of these',
'these charming',
'charming anecdotes',
'anecdotes and',
'and significantly',
'significantly pleases',
'pleases my',
'my beautiful',
'beautiful personage',
'personage i',
'i cannot',
'cannot wait',
'wait to',
'to enjoy',
'enjoy being',
'being on',
'on a',
'a brand',
'brand new',
'new jay',
'jay it',
'it certainly',
'certainly seems',
'seems like',
'like a',
'a grand',
'grand time',
'time to',
'to explore',
'explore life',
'life and',
'and love']
```

Use `ngram_creator` along with the prepared text to create a list of all the two-word ngrams in each `TextSample`. Use `build_frequency_table` to tabulate the frequency of each ngram. In the constructor for `TextSample` save this frequency table as `self.ngram_frequency`.

Comparing Two Frequency Tables

We want to know how similar two frequency tables are, let's write a function that computes the comparison between two frequency tables and scores them based on similarity.

Write a function called `frequency_comparison` that takes two parameters, `table1` and `table2`. It should define two local variables, `appearances` and `mutual_appearances`.

Iterate through `table1`'s keys and check if `table2` has the same key defined. If it is, compare the two values for the key -- the smaller value should get added to `mutual_appearances` and the larger should get added to `appearances`. If the key doesn't exist in `table2` the value for the key in `table1` should be added to `appearances`.

Remember afterwards to iterate through all of `table2`'s keys that aren't in `table1` and add those to `appearances` as well.

Return a frequency comparison score equal to the mutual appearances divided by the total appearances.

In [27]:
```
#Create comparison of two notes in frequency of words. Compare "def bu
ild_frequency_table(corpus):" dictionaries.
def frequency_comparison(table1, table2):
    appearances = 0
    mutual_appearances = 0
    for items in table1.keys():
        if items in table2.keys():
            if table1[items] <= table2[items]:
                mutual_appearances += table1[items]
                appearances += table2[items]
            else:
                mutual_appearances += table2[items]
                appearances += table1[items]
        else:
            appearances += table1[items]
    for word in table2.keys():
        if word not in table1.keys():
            appearances += table2[word]

    return mutual_appearances / appearances

#Testing tables against one another
print(frequency_comparison(build_frequency_table(prepare_text(murder_n
ote)),(build_frequency_table(prepare_text(myrtle_beech_intro)))))
print(frequency_comparison(build_frequency_table(prepare_text(murder_n
ote)),(build_frequency_table(prepare_text(lily_trebuchet_intro)))))
print(frequency_comparison(build_frequency_table(prepare_text(murder_n
ote)),(build_frequency_table(prepare_text(gregg_t_fishy_intro)))))
```

```
0.16591928251121077
0.23196881091617932
0.1561822125813449
```

Comparing Average Sentence Length

In order to calculate the change between the average sentence lengths of two `TextSamples` we're going to use the formula for the percent difference.

Write a function called `percent_difference` that returns the percent difference as calculated from the following formula:

$$\frac{|\ value1 - value2\ |}{\frac{value1 + value2}{2}}$$

In the numerator is the absolute value (use `abs()`) of the two values subtracted from each other. In the denominator is the average of the two values (value1 + value2 divided by two).

In [28]:
```
def percent_difference(value1, value2):
    value = abs(value1 - value2) /((value1 + value2)/2)
    return value

print(percent_difference(get_average_sentence_length(murder_note), get
_average_sentence_length(myrtle_beech_intro)))
print(percent_difference(get_average_sentence_length(murder_note), get
_average_sentence_length(lily_trebuchet_intro)))
print(percent_difference(get_average_sentence_length(murder_note), get
_average_sentence_length(gregg_t_fishy_intro)))
```

```
1.1428571428571428
0.3783783783783784
0.5142857142857142
```

Scoring Similarity with All Three Indicators

We want to figure out who did it, so let's use all three of the indicators we built to score text similarity. Define a function `find_text_similarity` that takes two `TextSample` arguments and returns a float between 0 and 1 where 0 means completely different and 1 means the same exact sample. You can evaluate the similarity by the following criteria:

- Calculate the percent difference of their average sentence length using `percent_difference`. Save that into a variable called `sentence_length_difference`. Since we want to find how *similar* the two passages are calculate the inverse of `sentence_length_difference` by using the formula `abs(1 - sentence_length_difference)`. Save that into a variable called `sentence_length_similarity`.
- Calculate the difference between their word usage using `frequency_comparison` on both `TextSample`'s `word_count_frequency` attributes. Save that into a variable called `word_count_similarity`.
- Calculate the difference between their two-word ngram using `frequency_comparison` on both `TextSample`'s `ngram_frequency` attributes. Save that into a variable called `ngram_similarity`.
- Add all three similarities together and divide by 3.

In [29]:
```
#Create function to determine text similarity
def find_text_similarity(text1, text2):
    sentence_length_difference = percent_difference(get_average_senten
ce_length(text1), get_average_sentence_length(text2))
    sentence_length_similarity = abs(1-sentence_length_difference)
    word_count_similarity = frequency_comparison(build_frequency_table
(prepare_text(text1)),build_frequency_table(prepare_text(text2)))
    bigram_similarity = frequency_comparison(build_frequency_table(big
ram_creator(prepare_text(text1))),build_frequency_table(bigram_creator
(prepare_text(text2))))

    return (sentence_length_similarity + word_count_similarity + bigra
m_similarity)/3

#Testing notes against one another
print(find_text_similarity(murder_note, myrtle_beech_intro))
print(find_text_similarity(murder_note, lily_trebuchet_intro))
print(find_text_similarity(murder_note, gregg_t_fishy_intro))
```

```
0.10683172512278453
0.30354580413452426
0.2197126258686585
```

Rendering the Results

We want to print out the results in a way that we can read! For each contestant on *A Brand New Jay* print out the following:

- Their name
- Their similarity score to the murder letter

In [30]:
```
print("Author: Myrtle Beech \nTheir similarity score to murder letter:
{}".format(find_text_similarity(myrtle_beech_intro, murder_note)))
print("\nAuthor: Lily Trebuchet \nTheir similarity score to murder let
ter: {}".format(find_text_similarity(lily_trebuchet_intro, murder_note
)))
print("\nAuthor: Gregg T. Fishy \nTheir similarity score to murder let
ter: {}".format(find_text_similarity(gregg_t_fishy_intro, murder_note)
))
```

```
Author: Myrtle Beech
Their similarity score to murder letter: 0.10683172512278453

Author: Lily Trebuchet
Their similarity score to murder letter: 0.30354580413452426

Author: Gregg T. Fishy
Their similarity score to murder letter: 0.2197126258686585
```

Who Dunnit?

In the cell below, print the name of the person who killed Jay Stacksby.

In [43]:
```
print("Based on this code, the person who allegedly killed Jay Stacksb
y is: \nLily Trebuchet")
```

```
Based on this code, the person who allegedly killed Jay Stacksby is:
Lily Trebuchet
```