```
!pip install english-words
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/
Collecting english-words
  Downloading english-words-1.1.0.tar.gz (1.1 MB)
     |████████████████████████████████| 1.1 MB 5.0 MB/s
Building wheels for collected packages: english-words
  Building wheel for english-words (setup.py) ... done
  Created wheel for english-words: filename=english_words-1.1.0-py3-none-any.whl siz
  Stored in directory: /root/.cache/pip/wheels/25/3d/4c/12a119ce90b46b4f90f9ddf41d71
Successfully built english-words
Installing collected packages: english-words
Successfully installed english-words-1.1.0
```

```
nltk.download('stopwords')
nltk.download('gutenberg')
nltk.download('wordnet')
nltk.download('omw-1.4')
nltk.download('sentiwordnet')
nltk.download('genesis')
nltk.download('inaugural')
nltk.download('nps_chat')
nltk.download('webtext')
nltk.download('treebank')
from nltk.corpus import wordnet as wn
import nltk
from english_words import english_words_lower_alpha_set
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.corpus import stopwords
from nltk.book import text4
import math
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package gutenberg to /root/nltk_data...
[nltk_data]   Package gutenberg is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package omw-1.4 to /root/nltk_data...
[nltk_data]   Package omw-1.4 is already up-to-date!
[nltk_data] Downloading package sentiwordnet to /root/nltk_data...
[nltk_data]   Package sentiwordnet is already up-to-date!
[nltk_data] Downloading package genesis to /root/nltk_data...
[nltk_data]   Package genesis is already up-to-date!
[nltk_data] Downloading package inaugural to /root/nltk_data...
[nltk_data]   Package inaugural is already up-to-date!
[nltk_data] Downloading package nps_chat to /root/nltk_data...
[nltk_data]   Package nps_chat is already up-to-date!
[nltk_data] Downloading package webtext to /root/nltk_data...
```

```
[nltk_data]   Package webtext is already up-to-date!
[nltk_data] Downloading package treebank to /root/nltk_data...
[nltk_data]   Package treebank is already up-to-date!
```

Q1 : Wordnet

Wordnet is a lexical database of the English language, which groups words by part of speech and concept they represent. On the surface level, its similar to a Thesaurus, but Wordnet establishes the semantical relationships between words, while a Thesaurus simply groups words by similar meanings.

Q2 : Synsets

```
noun = "chair"
wn.synsets(noun)
```

```
[Synset('chair.n.01'),
 Synset('professorship.n.01'),
 Synset('president.n.04'),
 Synset('electric_chair.n.01'),
 Synset('chair.n.05'),
 Synset('chair.v.01'),
 Synset('moderate.v.01')]
```

Q3 : Extract Synset

```
print("Definition : " + wn.synset('chair.n.01').definition())
print("Examples : " + str(wn.synset('chair.n.01').examples()))
print("Lemmas : " + str(wn.synset('chair.n.01').lemmas()))
print("Traversing Hierarchy")
curr_set = wn.synset('chair.n.01').hypernyms()
while(len(curr_set) > 0):
  print(curr_set)
  curr_set = curr_set[0].hypernyms()
```

```
Definition : a seat for one person, with a support for the back
Examples : ['he put his coat over the back of the chair and sat down']
Lemmas : [Lemma('chair.n.01.chair')]
Traversing Hierarchy
[Synset('seat.n.03')]
[Synset('furniture.n.01')]
[Synset('furnishing.n.02')]
[Synset('instrumentality.n.03')]
[Synset('artifact.n.01')]
```

```
[Synset('whole.n.02')]
[Synset('object.n.01')]
[Synset('physical_entity.n.01')]
[Synset('entity.n.01')]
```

The hierarchy of wordnet is organized into specific objetc, then more and more general terms, until it reaches the point to entity, which is the definition of a noun.

Q4 : The 'Nyms

```
print("Hypernyms : " + str(wn.synset('chair.n.01').hypernyms()))
print("Hyponyms : " + str(wn.synset('chair.n.01').hyponyms()))
print("Meronyms : " + str(wn.synset('chair.n.01').part_meronyms()))
print("Holonyms : " + str(wn.synset('chair.n.01').part_holonyms()))
print("Antonyms : " + str(wn.synset('chair.n.01').lemmas()[0].antonyms()))
```

```
Hypernyms : [Synset('seat.n.03')]
Hyponyms : [Synset('armchair.n.01'), Synset('barber_chair.n.01'), Synset('chair_of_s
Meronyms : [Synset('back.n.08'), Synset('leg.n.03')]
Holonyms : []
Antonyms : []
```

Q5 : Verb Synsets

```
verb = "run"
verb_synsets = wn.synsets(verb)
print(verb_synsets)
```

```
:.v.01'), Synset('run.v.03'), Synset('operate.v.01'), Synset('run.v.05'), Synset('run.
```

Q6 : Select Verb Synset

```
verb_syn = 'run.v.03'
print("Definition : " + wn.synset(verb_syn).definition())
print("Examples : " + str(wn.synset(verb_syn).examples()))
print("Lemmas : " + str(wn.synset(verb_syn).lemmas()))
print("Traversing Hierarchy")
curr_set = wn.synset(verb_syn).hypernyms()
while(len(curr_set) > 0):
```

```
8   print(str(curr_set[0]) + " = " + str(curr_set[0].definition()))
9   curr_set = curr_set[0].hypernyms()
```

```
Definition : stretch out over a distance, space, time, or scope; run or extend betwe
Examples : ['Service runs all the way to Cranbury', "His knowledge doesn't go very f
Lemmas : [Lemma('run.v.03.run'), Lemma('run.v.03.go'), Lemma('run.v.03.pass'), Lemma
Traversing Hierarchy
Synset('be.v.03') = occupy a certain position or area; be somewhere
```

Unlike Nouns, the verb hierarchy generalizes to "to be", in other words defining verbs in general as a state of being. In this case, "run", or grammatically, "running", is a state of being.

Q7 : Morphy

```
1 for word in english_words_lower_alpha_set:
2   if wn.morphy(word) == "run":
3     print(word)
```

```
ran
run
```

Q8 : Word Similarity

```
1 word_1 = wn.synset('stroll.v.01')
2 word_2 = wn.synset('walk.v.01')
3 word_1_leskTest = word_2.definition()
4 word_2_leskTest = word_1.definition()
5 print("Wu-Palmer Similarity = " + str(wn.wup_similarity(word_1, word_2)))
6 lesk_syn_1 = lesk(word_2_leskTest, "walk")
7 lesk_syn_2 = lesk(word_1_leskTest, "stroll")
8 print("Lesk Algorithm word_1 = " + str(lesk_syn_1) + " : " + lesk_syn_1.definition())
9 print("Lesk Algorithm word_2 = " + str(lesk_syn_2) + " : " + lesk_syn_2.definition())
```

```
Wu-Palmer Similarity = 0.8
Lesk Algorithm word_1 = Synset('walk.v.10') : take a walk; go for a walk; walk for p
Lesk Algorithm word_2 = Synset('amble.n.01') : a leisurely walk (usually in some pub
```

The main difference is that Wu-Palmer compares the similarity of two words, while Lesk attempts to figure out the correct synset of the word based on teh context of the sentence its in. Because of this, in order to test the word similarity, I used the definition of each word as the sentence of the other

word, in order to see if they were similar enough that the word in the other sentence would return the synset, but they did not. To be honest, Lesk Algorithm really isnt suited to this.

Q9 : SentiWordNet

SentiWordNet is basically a lexical analysis method of deducing the emotional charge of a given word, based on the context and the general definitions of said word. One of the best ways to use it is to deduce the tone of a piece of text.

```
word = "tantrum"
tantrum = swn.senti_synset('tantrum.n.01')
print(tantrum)
print("Positive score = ", tantrum.pos_score())
print("Negative score = ", tantrum.neg_score())
print("Objective score = ", tantrum.obj_score())
ex_sent = "The boy threw a tantrum because he wanted more candy".split(' ')
for word in ex_sent:
  word1 = swn.senti_synsets(word)
  first = True
  for item in word1:
    print(item)
```

```
<fit.n.01: PosScore=0.0 NegScore=0.5>
Positive score =  0.0
Negative score =  0.5
Objective score =  0.5
<male_child.n.01: PosScore=0.25 NegScore=0.0>
<boy.n.02: PosScore=0.0 NegScore=0.0>
<son.n.01: PosScore=0.0 NegScore=0.0>
<boy.n.04: PosScore=0.0 NegScore=0.125>
<throw.v.01: PosScore=0.0 NegScore=0.0>
<throw.v.02: PosScore=0.0 NegScore=0.0>
<shed.v.01: PosScore=0.0 NegScore=0.0>
<throw.v.04: PosScore=0.0 NegScore=0.0>
<give.v.07: PosScore=0.0 NegScore=0.375>
<throw.v.06: PosScore=0.0 NegScore=0.0>
<project.v.10: PosScore=0.0 NegScore=0.0>
<throw.v.08: PosScore=0.0 NegScore=0.0>
<bewilder.v.02: PosScore=0.0 NegScore=0.25>
<hurl.v.03: PosScore=0.125 NegScore=0.0>
<hold.v.03: PosScore=0.125 NegScore=0.0>
<throw.v.12: PosScore=0.0 NegScore=0.0>
<throw.v.13: PosScore=0.0 NegScore=0.0>
<throw.v.14: PosScore=0.0 NegScore=0.0>
<confuse.v.02: PosScore=0.125 NegScore=0.625>
<angstrom.n.01: PosScore=0.0 NegScore=0.0>
<vitamin_a.n.01: PosScore=0.125 NegScore=0.25>
<deoxyadenosine_monophosphate.n.01: PosScore=0.0 NegScore=0.0>
<adenine.n.01: PosScore=0.0 NegScore=0.0>
<ampere.n.02: PosScore=0.0 NegScore=0.0>
```

```
<a.n.06: PosScore=0.0 NegScore=0.0>
<a.n.07: PosScore=0.0 NegScore=0.0>
<fit.n.01: PosScore=0.0 NegScore=0.5>
<helium.n.01: PosScore=0.0 NegScore=0.0>
<he.n.02: PosScore=0.0 NegScore=0.0>
<desire.v.01: PosScore=0.25 NegScore=0.0>
<want.v.02: PosScore=0.125 NegScore=0.125>
<want.v.03: PosScore=0.0 NegScore=0.0>
<want.v.04: PosScore=0.0 NegScore=0.0>
<want.v.05: PosScore=0.0 NegScore=0.625>
<wanted.a.01: PosScore=0.625 NegScore=0.0>
<cherished.s.01: PosScore=0.625 NegScore=0.25>
<more.n.01: PosScore=0.0 NegScore=0.0>
<more.a.01: PosScore=0.0 NegScore=0.0>
<more.a.02: PosScore=0.0 NegScore=0.0>
<more.r.01: PosScore=0.0 NegScore=0.0>
<more.r.02: PosScore=0.0 NegScore=0.0>
<candy.n.01: PosScore=0.0 NegScore=0.0>
<sugarcoat.v.01: PosScore=0.0 NegScore=0.0>
```

What the scores tell the program is basically whether each word has a positive or negative connotation, and the intensity of said word. This can allow the program to gain a second layer of comprehension, by allowing it to not just understand literal meaning, but also the meaning hidden in the tone of the sentence.

Q10 : Collocation

Collocations are phrases, in which the words that make them up have more meaning together than they do apart.

```
1 text4.collocations()
```

```
United States; fellow citizens; years ago; four years; Federal
Government; General Government; American people; Vice President; God
bless; Chief Justice; one another; fellow Americans; Old World;
Almighty God; Fellow citizens; Chief Magistrate; every citizen; Indian
tribes; public debt; foreign nations
```

```
1 text = ' '.join(text4.tokens)
2 collo = "Chief Justice"
3 vocab = len(set(text))
4 hg = text.count('Chief Justice')/vocab
5 print("p(Chief Justice) = ",hg )
6 h = text.count('Chief')/vocab
7 print("p(Chief) = ", h)
8 g = text.count('Justice')/vocab
9 print('p(Justice) = ', g)
10 pmi = math.log2(hg / (h * g))
```

```
pmi = math.log2(ng / (nf * ng))
print('Mutual Information = ', pmi)
```

```
p(Chief Justice) =  0.16666666666666666
p(Chief) =  0.3333333333333333
p(Justice) =  0.2857142857142857
Mutual Information =  0.8073549220576041
```

Based on a Mutual Information that comes to about 0.81, "Chief Justice" is very likely to be a collocation. In an NLTK application, this would be helpful because if the program were to fail to catch collocations, it would lose the comprehension that comes from the added meaning from the combination fo the words.