Number 1

WordNet is a lexical database, aka english dictionary, that can be used along the nltk library originating from Princeton. WordNet works by forming synonym sets for words and developing hiearchies based on the sematic relations between sysnets.

```
# all my used imports
import nltk
from nltk.corpus import wordnet as wn
from nltk.wsd import lesk
from nltk.corpus import sentiwordnet as swn
from nltk.book import text4
from nltk.tokenize import word_tokenize
```

Number 2

Selected noun - cake

```
 # output all synsets of 'cake'

wn.synsets('cake')
```

```
[Synset('cake.n.01'),
 Synset('patty.n.01'),
 Synset('cake.n.03'),
 Synset('coat.v.03')]
```

Number 3

selected synset - cake.n.01

```
# extract definition
wn.synset('cake.n.01').definition()
```

```
'a block of solid substance (such as soap or wax)'
```

```
# extract examples
wn.synset('cake.n.01').examples()
```

```
['a bar of chocolate']
```

```
# extract lemmas
wn.synset('cake.n.01').lemmas()
```

```
[Lemma('cake.n.01.cake'), Lemma('cake.n.01.bar')]
```

```
# traverse the hiearacy
cake = wn.synset('cake.n.01')
hyper = lambda s: s.hypernyms()
list(cake.closure(hyper))
```

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

In term of the organization of nouns, all nouns can be traced back to the sysnet `entity.n.01`, allowing us to easily traverse the hiearachy of hypernyms and get to the "root". This logically makes sense because all nouns are technically "things" aka entities, thus it makes sense that all nouns can be encapsulated by a singular sysnet.

Number 4

```
cake = wn.synset('cake.n.01')
print('hypernyms: ', cake.hypernyms())
print('hyponyms: ', cake.hyponyms())
print('holonyms: ', cake.member_holonyms())
print('meronyms: ', cake.member_meronyms())
print('antonyms: ', cake.lemmas()[0].antonyms())
```

```
hypernyms:  [Synset('block.n.01')]
hyponyms:  [Synset('tablet.n.03')]
holonyms:  []
meronyms:  []
antonyms:  []
```

Number 5

Selected verb - juggle

```
 # get all synsets of 'juggle'

wn.synsets('juggle')
```

```
[Synset('juggle.n.01'),
 Synset('juggle.n.02'),
 Synset('juggle.v.01'),
 Synset('juggle.v.02'),
 Synset('juggle.v.03'),
 Synset('juggle.v.04'),
 Synset('juggle.v.05')]
```

Number 6

```
# extract definition
wn.synset('juggle.v.01').definition()
```

```
# extract examples
wn.synset('juggle.v.01').examples()
```

```
# extract lemmas
wn.synset('juggle.v.01').lemmas()
```

```
# traverse the hieracy
juggle = wn.synset('juggle.v.01')
hyper = lambda s: s.hypernyms()
list(juggle.closure(hyper))
```

```
[Synset('cheat.v.01'),
 Synset('victimize.v.01'),
 Synset('wrong.v.01'),
 Synset('treat.v.01'),
 Synset('interact.v.01'),
 Synset('act.v.01')]
```

In term of the organization of verbs, not all verbs can be traced back to a single sysnet, as was the case in nouns. Logically this makes sense because not all verbs can be classified under one terms as easily as noun can. We see that the highest we go in this verb hierachy is going up to "act" which makes sense as the word juggle is an act, not a transitive or helping verb.

Number 7

```
print(wn.morphy("juggle", wn.ADJ))
```

```
None
```

```
print(wn.morphy("juggle", wn.VERB))
```

```
juggle
```

```
print(wn.morphy("juggle", wn.NOUN))
```

```
juggle
```

```
print(wn.morphy("juggle", wn.ADV))
```

```
None
```

Number 8

Selected words - French, English

```
# get the synsets
print(wn.synsets('french'))
print(wn.synsets('english'))
french = wn.synset('french.n.01')
lemmas = [l.name() for l in french.lemmas()]
print(str(lemmas))
english = wn.synset('english.n.01')
lemmas = [l.name() for l in english.lemmas()]
print(str(lemmas))
```

```
[Synset('french.n.01'), Synset('french.n.02'), Synset('french.n.03'), Synset('french.v.01'), Synset('french.a.01')]
[Synset('english.n.01'), Synset('english.n.02'), Synset('english.n.03'), Synset('english.n.04'), Synset('english.a.01'), Synset('englis
['French']
['English', 'English_language']
```

```
# Wu-Palmer similarity
wn.wup_similarity(french, english)
```

```
0.631578947368421
```

```
# Running Lesk
sent = ['I', 'speak', 'french']
print(lesk(sent, 'french', 'n').definition())
```

```
United States sculptor who created the seated marble figure of Abraham Lincoln in the Lincoln Memorial in Washington D.C. (1850-1931)
```

Obeservations - As we can see above running the Wu-Palmer similarity metric gave us an output of 0.6315 or around 63%. Wu Palmer is based on how similar word senses are to one another and where the synsets occur relative to each other in the hiearchy of each word. This similarity implies that some hypernyms must be shared and may potentially be realitvely close to one another. Logically this makes sense, as english and french are both languages. Upon running the Lesk algorithm, we can see that I am aiming for the sysnet `french.n.01` (french as a language) by having my sentence as `I speak french` however we see that I did not get that, I ended up getting a historic artist returned, which was not my intention in the sentence.

Number 9

SentiWordNet is resource built on top of wordNet and essentially returns three scores for each parameter: how positive the word is, how negative the word and how objective the word is. The max of each score is a 1.0, meaning that the text is 100% that attribute This allows users to perform sentinet analysis on some text to see potentially how objective/subjective a test is or if it has more of a negative or positive connotation. For example in order to find out if an article or site is a good source for objective facts, you could perform sentinet analysis on that site/article and see if SentiWordNet considers it objective.

My emotionally charged word - "disgusting"

```
sysnets = list(swn.senti_synsets('disgusting'))

for breakdown in sysnets:
  print(f"sysnet {breakdown}")
  print("Positive score = ", breakdown.pos_score())
  print("Negative score = ", breakdown.neg_score())
  print("Objective score = ", breakdown.obj_score())
  print()
```

```
sysnet <disgust.v.01: PosScore=0.0 NegScore=0.125>
Positive score =  0.0
Negative score =  0.125
Objective score =  0.875

sysnet <disgust.v.02: PosScore=0.125 NegScore=0.625>
Positive score =  0.125
Negative score =  0.625
Objective score =  0.25

sysnet <disgusting.s.01: PosScore=0.25 NegScore=0.75>
Positive score =  0.25
Negative score =  0.75
Objective score =  0.0
```

```
sent = 'That brownie was really disgusting'
tokens = sent.split()
for token in tokens:
  syn_list = list(swn.senti_synsets(token))
  if syn_list:
    syn = syn_list[0]
    print(f"Negative score of {token} - {syn.neg_score()}")
    print(f"Positive score of {token} - {syn.pos_score()}")
    print(f"Objectivity of  {token} - {syn.obj_score()}")
    print()
```

```
Negative score of brownie - 0.0
Positive score of brownie - 0.0
Objectivity of  brownie - 1.0

Negative score of was - 0.0
Positive score of was - 0.0
Objectivity of  was - 1.0

Negative score of really - 0.0
Positive score of really - 0.625
Objectivity of  really - 0.375

Negative score of disgusting - 0.125
Positive score of disgusting - 0.0
Objectivity of  disgusting - 0.875
```

Observations - Upon performing a sentient analysis on the sentence `that brownie was really disgusting`, an obviously negative sentence, we can see that the analysis somewhat picked up upon that negative connotation with the word "disgusting". It is intereseting to see that the word "really" which was used to emphasize a negative effect had a no negative score, but infact a relatively high positive score. So while performing a sentient analysis is useful, we can see that this algorithm may not be the most accurate when attempting to figure out the "intentions" behind a sentence.

Number 10

A collocation is two or more words are found together more often than probabilities would suggest. Its important to notes that changing any word in the collocation would give it a different meaning. Examples of collocations include `Sick and tired` and `crystal clear`. Trying to change sick and tired to unwell and exhausted removes the original intention of those words, as explained above.

```
# Display Collocations
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
```

```
# Selected Collocation - God bless
import math
vocab = len(set(text4))
text = ' '.join(text4.tokens)
gb = text.count('God bless')/vocab
print("p(God Bless) = ",gb )
g = text.count('God')/vocab
print("p(God) = ", g)
b = text.count('bless')/vocab
print('p(bless) = ', b)
pmi = math.log2(gb / (g * b))
print('pmi = ', pmi)
```

```
p(God Bless) =  0.0016957605985037406
p(God) =  0.011172069825436408
p(bless) =  0.0085785536159601
pmi =  4.145157780720282
```

Upon looking at the results of the calculation, we see that the PMI of 'God bless' is a positive number `4.145`. Ths means that God bless is likely to be a collocation, if the number was negative, than it would indicate the opposite

✓ 0s completed at 9:24 PM