'''Part (b) get\_sentence\_lists\_from\_files(filenames)

This function takes in a list of strings filenames, each one the name of a text file, and returns the list of

every sentence contained in all the text files in filenames, in order.'''

'''

def get\_sentence\_lists\_from\_files(filenames):

for i in range(len(filenames)):

get\_sentence\_lists(filenames)

'''

'''Part (c) build\_semantic\_descriptors(sentences)

This function takes in a list sentences which contains lists of strings representing sentences, and returns

a dictionary d such that for every word w that appears in at least one of the sentences, d[w] is itself a

dictionary which represents the semantic descriptor of w (note: the variable names here are arbitrary).

For example, if sentences represents the opening of Notes from the Underground as above, part of the

dictionary returned would be:

{ ’man’: {’i’: 3, ’am’: 3, ’a’: 2, ’sick’: 1, ’spiteful’: 1, ’an’: 1,

’unattractive’: 1},

’liver’: {’i’: 1, ’believe’: 1, ’my’: 1, ’is’: 1, ’diseased’: 1},

... }

with as many keys as there are distinct words in the passage.'''

'''build a dictionary d

such that with every new word, we make a new d[w] a new subdictionary

each sub-dictionary goes through each of the sentences from the first function and takes down how many times each other word appears in a sentence itself does too'''

'''

def build\_semantic\_descriptors(sentences):

dictionary\_count = {}

for sentence in sentences: #looking at one sentence/list

for word in sentence: #looking at each word in the sentence/list

if word not in dictionary\_count:

dictionary\_count[word] = {}

else:

pass

for word in dictionary\_count: #each new/inserted word in the dict

for sentence in sentences:

for new\_word in sentence:

if (new\_word != word) and (new\_word not in word): #not sure about first past since new\_word is a word and word is a dictionary, but make sure we're comparing words, not using the same ones

word[new\_word] = 1

elif new\_word in word:

word[new\_word] += 1

return dictionary\_count

'''

'''Part (d) most\_similar\_word(word, choices, semantic\_descriptors)

This function takes in a string word, a list of strings choices, and a dictionary semantic\_descriptors

which is built according to the requirements for build\_semantic\_descriptors, and returns the element

of choices which has the largest semantic similarity to word, with the semantic similarity computed using

the data in semanticdescriptors. If the semantic similarity between two words cannot be computed, it

is considered to be neagtive 1. In case of a tie between several elements in choices, the one with the smallest index

in choices should be returned (e.g., if there is a tie between choices[5] and choices[7], choices[5] is

returned).'''

'''

comparisons

save each semantic output and the choice word into a list []

return max key of the list

finding semantic calculation

top = 0

bottom = 0

for i in dictionary\_count(word):

for m in dictionary\_count(choices[0]): #this function will have 2 word/strings as inputs, latter to replace choices[0]

if i == m:

top += i\*m #this is words \* words, right? so this i should change the code up there to be for rangelen

part\_one = 0

part\_two = 0

for i in dictionary\_count(word):

part\_one += (i[1])\*\*2

for m in dictionary\_count(choices[0]):

part\_two += (m[1])\*\*2

bottom = (i\*m)\*\*0.5

'''