```
import random; random.seed(123)
import codecs
import string
from nltk.stem.porter import PorterStemmer
from nltk.probability import FreqDist
import gensim
from nltk.tokenize import word_tokenize
# Dokumentasjon på NLTK: https://kite.com/python/docs/nltk.FreqDist


#################
## OPPGAVE 1 ##
#################

# 1.1
f = codecs.open("pg3300.txt", "r", "utf-8")

# 1.2 Lager en lang streng med all texten, og deler deretter opp på linjeskift.
file = f.readlines()
all_text = ""
for line in file:
    all_text += line


# Deler opp teksten der det er linjeskift
paragraphs = all_text.split("\n\r")


filtered_gutenberg = [] # Filtrert liste uten gutenberg

# 1.3 Fjerner alle avsnitt med Gutenberg i seg. Lagrer disse i en ny liste.
for par in paragraphs:
    if "gutenberg" not in par.lower():
        filtered_gutenberg.append(par)

# Fjerner alle tomme paragrafer
def remove_empty_par(paragraphs):
    filtered_list = []
    for par in paragraphs:
        if len(par) > 0:
            filtered_list.append(par)
    return filtered_list

filtered_paragraphs = remove_empty_par(filtered_gutenberg)

# 1.4 Gjør om slik at hvert element i listen er en liste med ord
paragraphs_listed_words = []
for par in filtered_paragraphs:
    paragraphs_listed_words.append(par.split())


# 1.5 Fjerner alle tegn og gjør om til små bokstaver. Gjør flere deloppgaver i samme løkke for å slippe mange
doble løkker og lang kjøretid.
par_counter = 0 #Brukes i løkka under
```

```
word_counter = 0 #Brukes også i løkka under

stemmer = PorterStemmer()
freqDist =  FreqDist()

for par in paragraphs_listed_words:
  word_counter = 0
  for word in par:
    # 1.6 Stemmer alle ord i linjen under
    paragraphs_listed_words[par_counter][word_counter] = stemmer.stem(word.strip(string.punctuation + "\n\r\t").lower()) #Fjerner alle punctuation og evt resterende tegn for linjeskift innrykk etc og gjør om til små bokstaver
    # 1.7 Teller opp antall forekomster av hver ord
    freqDist[paragraphs_listed_words[par_counter][word_counter]] += 1
    word_counter += 1
  # Fjerner alle tomme strenger
  paragraphs_listed_words[par_counter] = list(filter(None, paragraphs_listed_words[par_counter]))
  par_counter += 1

# Fjerner til slutt alle tomme lister
paragraphs_listed_words = list(filter(None, paragraphs_listed_words))



#################
## OPPGAVE 2 ##
#################

# 2.1 Oppretter dictonary
dictionary = gensim.corpora.Dictionary(paragraphs_listed_words)

# 2.1 Filtrerer ut stopwords. Må da stemme de først og deretter fjerne ved å iterere gjennom.
stopwords = codecs.open("stopwords.txt", "r", "utf-8")
stopwords = stopwords.readlines()
stopwords = stopwords[0].split(",")

# Stemmer stopwords
def stem_stopwords(stopwords):
  for i in range (len(stopwords)):
    stopwords[i] = stemmer.stem(stopwords[i])

stem_stopwords(stopwords)
# Stopwords er nå en liste med alle ordene og er stemmet

stop_ids = []

# Fjerner alle stoppordene fra dictionary
def remove_stopword(stopwords):
  for word in stopwords:
    try:
      id = dictionary.token2id[word]
      stop_ids.append(id)
    except KeyError:
      continue
```

```
    dictionary.filter_tokens(stop_ids)

remove_stopword(stopwords)


# 2.2 Lager en bag of words
def map_to_bag(paragraphs):
    corpus = []
    for par in paragraphs:
        list = dictionary.doc2bow(par)
        corpus.append(list)
    return corpus


corpus = map_to_bag(paragraphs_listed_words)
##################
## OPPGAVE 3 ##
##################

# 3.1 Lager en TF-IDF modell
tfidf_model = gensim.models.TfidfModel(corpus)

# 3.2 Mapper bag-of-words til TF-IDF vekter
corpus_tfidf = tfidf_model[corpus]

# 3.3 Oppretter MatrixSimilarity objekt

matsim = gensim.similarities.MatrixSimilarity(corpus)

# 3.4 Gjør samme for LSI modell
lsi = gensim.models.LsiModel(corpus_tfidf, id2word=dictionary, num_topics=100)
lsi_corpus = lsi[corpus_tfidf]

# 3.5
#print(lsi.show_topic(1))


##################
## OPPGAVE 4 ##
##################

# 4.1 Skriver en funksjon for å preprossesere spørringen, og gjør deretter om til BOW
def preprocessing(q):
    list = q.split()
    for i in range(len(list)):
        list[i] = stemmer.stem(list[i].strip(string.punctuation).lower())
    return list



query = preprocessing("What is the function of money?")
query = dictionary.doc2bow(query)

# 4.2 Reporting weights
```

```
query_tfidf = tfidf_model[query]
index = gensim.similarities.MatrixSimilarity(corpus_tfidf)


def report_weights(query_tfidf):
    for pair in query_tfidf:
        weight = pair[1]
        word = dictionary.get(pair[0])
        print(word, ": ", "%0.2f" % weight)

print(report_weights(query_tfidf))
# Prints this to console:
    # money :  0.32
    # function :  0.95


# 4.3 Report top 3 most relevant paragraphs


def report_most_relevant_paragraphs(q):
    docs2similarity = enumerate(index[q])
    sorted_docs = sorted(docs2similarity, key=lambda kv: -kv[1])[:3]
    relevant_paragraphs = []
    for pair in sorted_docs:
        relevant_paragraphs.append(pair[0])
    for par in relevant_paragraphs:
        print("[Paragraph: ", par+1, "]", "\n")
        lines = filtered_paragraphs[par].splitlines(6)
        filter_lines = ""
        n = 0
        try:
            for i in range(6):
                filter_lines += " " +lines[i].strip("\n\r")
                n= i
        except IndexError:
            filter_lines = ""
            for i in range(n+1):
                filter_lines +=" " +lines[i]

        print(filter_lines, "\n")


# report_most_relevant_paragraphs(query_tfidf)

# Funksjonen over skriver ut følgende resultater til konsoll:
#[Paragraph:  677 ]
#
#
# The general stock of any country or society is the same with that of
# all its inhabitants or members; and, therefore, naturally divides itself
# into the same three portions, each of which has a distinct function or
# office.
#
#
```

```
# [Paragraph:  988 ]
#
# That wealth consists in money, or in gold and silver, is a popular
# notion which naturally arises from the double function of money, as the
# instrument of commerce, and as the measure of value. In consequence of
# its being the instrument of commerce, when we have money we can more
# readily obtain whatever else we have occasion for, than by means of any
#
# [Paragraph:  812 ]
#
# Whatever part of his stock a man employs as a capital, he always expects
# it to be replaced to him with a profit. He employs it, therefore,
# in maintaining productive hands only; and after having served in the
# function of a capital to him, it constitutes a revenue to them. Whenever
# he employs any part of it in maintaining unproductive hands of any kind.


# 4.4


# TESTKODE FOR Å SJEKKE AT DET FUNGERER LIKT SOM I OPPGAVEN
# test_query = "How taxes influence Economics?"
# test_query = preprocessing(test_query)
# test_query = dictionary.doc2bow(test_query)
# test_tfidf = tfidf_model[test_query]
# test_lsi = lsi[test_tfidf]
# sorted_test = (sorted(test_lsi, key=lambda kv: -abs(kv[1]))[:3] ) #[(3, 0.1236889420871775), (5,
0.08609030455385876), (9, 0.08523104132289301)]
# all_test_topics = (lsi.show_topics())
# for i in sorted_test:
#     print("[Topic ", i[0], "]")
#     print(all_test_topics[i[0]][1])
#printer følgende til konsoll, bekrefter at testquery gir samme output som i oppgavebeskrivelsen:
# [Topic  3 ]
# -0.467*"tax" + -0.201*"rent" + 0.193*"trade" + 0.166*"capit" + 0.154*"foreign" + 0.154*"employ" + -0.151*"upon"
+ 0.137*"quantiti" + 0.137*"labour" + 0.134*"manufactur"
# [Topic  5 ]
# 0.383*"tax" + 0.217*"capit" + 0.162*"foreign" + 0.137*"duti" + 0.133*"trade" + 0.130*"consumpt" + 0.127*"upon"
+ 0.126*"export" + 0.120*"profit" + 0.118*"home"
# [Topic  9 ]
# 0.314*"tax" + -0.258*"bank" + 0.206*"coloni" + -0.182*"land" + 0.181*"labour" + -0.156*"corn" + 0.154*"wage" +
-0.149*"manufactur" + -0.145*"rent" + -0.140*"export"
# TESTKODE SLUTT



lsi_query = lsi[query_tfidf]
sorted_lsi = (sorted(lsi_query, key=lambda kv: -abs(kv[1]))[:3] )
all_topics = lsi.show_topics()
for i in sorted_lsi:
    print("[Topic ", i[0], "]")
    print((all_topics[i[0]][1]))
```

```
# Får følgende resultat skrevet til konsoll:
# [Topic  4 ]
# -0.262*"bank" + 0.251*"price" + -0.233*"capit" + -0.232*"circul" + -0.188*"gold" + -0.184*"money" +
0.181*"corn" + 0.141*"import" + -0.140*"coin" + -0.140*"revenu"
# [Topic  20 ]
# -0.305*"expens" + -0.215*"work" + 0.192*"interest" + -0.182*"bounti" + -0.167*"bank" + 0.156*"money" +
-0.152*"coin" + 0.125*"stock" + -0.124*"mine" + 0.124*"revenu"
# [Topic  16 ]
# 0.309*"circul" + -0.219*"increas" + -0.199*"cent" + -0.194*"per" + -0.192*"coin" + 0.157*"mine" + 0.146*"money"
+ 0.145*"coloni" + -0.142*"industri" + 0.141*"materi"
```