```
import math
import nltk
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('omw-1.4')
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
```

```
    [nltk_data] Downloading package punkt to /root/nltk_data...
    [nltk_data]   Unzipping tokenizers/punkt.zip.
    [nltk_data] Downloading package wordnet to /root/nltk_data...
    [nltk_data] Downloading package omw-1.4 to /root/nltk_data...
```

```
# define document
doc_1 = "Every year Maha Shivratri is celebrated with a lot of pomp and grandeur. It is considered to be a very special time of the year since
doc_2 = "Lord Shiva devotees celebrate this occasion with a lot of grandness. It is accompanied by folk dances,songs, prayers, chants, mantras
doc_3 = "People keep a fast on this Maha shivratri, stay awake at night and pray to the lord for blessings,happiness, hope and prosperity. This
doc_4 = "The festival of Maha Shivratri will be celebrated on February 18 and is a very auspicious festival. This Hindu festival celebrates the
documents = [doc_1,doc_2,doc_3,doc_4]
```

```
# Tokenizeing  the documents
#Tokenization is the process of replacing sensitive data with unique identification symbols
#that retain all the essential information about the data without compromising its security.
tokenized_docs = []
for doc in documents:
    tokens = nltk.word_tokenize(doc)
    tokenized_docs.append(tokens)
```

```
# Defineing  the vocabulary
vocab = set()
for tokens in tokenized_docs:
    for word in tokens:
        vocab.add(word)
```

```
# Lemmatizeing  the tokens
#Lemmatisation (or lemmatization) in linguistics is the process of grouping together
#the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.
```

```
#the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.
lemmatizer = WordNetLemmatizer()
lemmatized_docs = []
for tokens in tokenized_docs:
    lemmatized_tokens = []
    for token in tokens:
        lemmatized_token = lemmatizer.lemmatize(token)
        lemmatized_tokens.append(lemmatized_token)
    lemmatized_docs.append(lemmatized_tokens)


# Calculateing the term frequency for each document
tf = []
for doc in lemmatized_docs:
    doc_tf = {}
    for word in doc:
        if word in doc_tf:
            doc_tf[word] += 1
        else:
            doc_tf[word] = 1
    tf.append(doc_tf)


# Calculateing the inverse document frequency for each term in the vocabulary
idf = {}
N = len(documents)
epsilon = 1e-6
for word in vocab:
    n = sum([1 for doc in lemmatized_docs if word in doc])
    idf[word] = math.log(N/(n + epsilon))
```

```
# Calculateing  the tf-idf for each document
tf_idf = []
for i in range(N):
    doc_tf_idf = {}
    doc_tf = tf[i]
    for word in doc_tf:
        if word in idf:
            doc_tf_idf[word] = doc_tf[word] * idf[word]
    tf_idf.append(doc_tf_idf)


# the  query
query = input("Maha Shivratri will be celebrated on February")
```

```
Maha Shivratri will be celebrated on February18
```

```
# Calculateing  the tf-idf for the query
query_tf_idf = {}
query_words = query.split()
for word in query_words:
    if word in query_tf_idf:
        query_tf_idf[word] += 1
    else:
        query_tf_idf[word] = 1
for word in query_tf_idf:
    if word in idf:
        query_tf_idf[word] *= idf[word]



# Defineing  a function to calculate the cosine similarity between document and query
def cosine_similarity(set1, set2):
    dot = sum([set1[word] * set2[word] for word in set1 if word in set2])
    norm_doc = math.sqrt(sum([set1[word]**2 for word in set1]))
    norm_query = math.sqrt(sum([set2[word]**2 for word in set2]))
    return dot/ (norm_doc * norm_query)


# Definein a function to calculate the Jaccard similarity between two documents
def jaccard_similarity(set1, set2):
    intersection = set1.intersection(set2)
    union = set1.union(set2)
    return len(intersection) / len(union)


# the cosine similarity between the query and each document
cosine_similarities = []
for i in range(N):
    cosine_similarities.append(cosine_similarity(query_tf_idf, tf_idf[i]))

# the Jaccard similarity between the query and each document
jaccard_similarities = []
for i in range(N):
    jaccard_similarities.append(jaccard_similarity(set(query_words), set(documents[i].split())))


# Sorting the cosine similarities in descending order and printing  the top two documents

cosine_similarities_sorted = sorted(range(len(cosine_similarities)), key=lambda i: cosine_similarities[i], reverse=True)
print("Top two documents by cosine similarity:")
for i in range(2):
    print("Document {}: {}".format(cosine_similarities_sorted[i]+1, cosine_similarities[cosine_similarities_sorted[i]]))

# Sort the Jaccard similarities in descending order and print the top two documents
jaccard_similarities_sorted = sorted(range(len(jaccard_similarities)), key=lambda i: jaccard_similarities[i], reverse=True)
print("Top two documents by Jaccard similarity:")
for i in range(2):
    print("Document {}: {}".format(jaccard_similarities_sorted[i]+1, jaccard_similarities[jaccard_similarities_sorted[i]]))
```

```
Top two documents by cosine similarity:
Document 2: 0.12595463835235884
Document 4: 0.1046909068624883
Top two documents by Jaccard similarity:
Document 4: 0.027777777777777776
Document 1: 0.0
```

✓ 0s completed at 10:25 PM