```
# To scrape Wikipedia
from bs4 import BeautifulSoup
# To access contents from URLs
import requests
# to preprocess text
import nltk
# to handle punctuations
from string import punctuation
# TF-IDF vectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
# cosine similarity score
from sklearn.metrics.pairwise import cosine_similarity
# to do array operations
import numpy as np
# to have sleep option
from time import sleep
import nltk
nltk.download('punkt')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
True
```

```
class ChatBot():

    # initialize bot
    def __init__(self):
        # flag whether to end chat
        self.end_chat = False
        # flag whether topic is found in wikipedia
        self.got_topic = False
        # flag whether to call respond()
        # in some cases, response be made already
        self.do_not_respond = True

        # wikipedia title
        self.title = None
        # wikipedia scraped para and description data
        self.text_data = []
        # data as sentences
        self.sentences = []
        # to keep track of paragraph indices
        # corresponding to all sentences
        self.para_indices = []
        # currently retrieved sentence id
        self.current_sent_idx = None

        # a punctuation dictionary
        self.punctuation_dict = str.maketrans({p:None for p in punctuation})
        # wordnet lemmatizer for preprocessing text
        self.lemmatizer = nltk.stem.WordNetLemmatizer()
        # collection of stopwords
        self.stopwords = nltk.corpus.stopwords.words('english')
        # initialize chatting
        self.greeting()

    # greeting method - to be called internally
    # chatbot initializing chat on screen with greetings
    def greeting(self):
        print("Initializing ChatBot ...")
        # some time to get user ready
        sleep(2)
        # chat ending tags
        print('Type "bye" or "quit" or "exit" to end chat')
        sleep(2)
        # chatbot descriptions
        print('\nEnter your topic of interest when prompted. \
        \nChaBot will access Wikipedia, prepare itself to \
        \nrespond to your queries on that topic. \n')
        sleep(3)
        print('ChatBot will respond with short info. \
        \nIf you input "more", it will give you detailed info \
        \nYou can also jump to next query')
        # give time to read what has been printed
        sleep(3)
        print('-'*50)
        # Greet and introduce
        greet = "Hello, Great day! Please give me a topic of your interest. "
        print("ChatBot >>  " + greet)

    # chat method - should be called by user
    # chat method controls inputs, responses, data scraping, preprocessing, modeling.
    # once an instance of ChatBot class is initialized, chat method should be called
    # to do the entire chatting on one go!
    def chat(self):
        # continue chat
        while not self.end_chat:
            # receive input
            self.receive_input()
            # finish chat if opted by user
            if self.end_chat:
                print('ChatBot >>  See you soon! Bye!')
                sleep(2)
                print('\nQuitting ChatBot ...')
            # if data scraping successful
            elif self.got_topic:
                # in case not already responded
                if not self.do_not_respond:
                    self.respond()
                # clear flag so that bot can respond next time
                self.do_not_respond = False

    # receive_input method - to be called internally
    # recieves input from user and makes preliminary decisions
    def receive_input(self):
        # receive input from user
        text = input("User    >> ")
        # end conversation if user wishes so
        if text.lower().strip() in ['bye', 'quit', 'exit']:
            # turn flag on
            self.end_chat=True
        # if user needs more information
        elif text.lower().strip() == 'more':
            # respond here itself
            self.do_not_respond = True
            # if at least one query has been received
            if self.current_sent_idx != None:
                response = self.text_data[self.para_indices[self.current_sent_idx]]
            # prompt user to start querying
            else:
                response = "Please input your query first!"
            print("ChatBot >>  " + response)
        # if topic is not chosen
        elif not self.got_topic:
            self.scrape_wiki(text)
        else:
            # add user input to sentences, so that we can vectorize in whole
            self.sentences.append(text)

    # respond method - to be called internally
    def respond(self):
```

```
        # tf-idf-modeling
        vectorizer = TfidfVectorizer(tokenizer=self.preprocess)
        # fit data and obtain tf-idf vector
        tfidf = vectorizer.fit_transform(self.sentences)
        # calculate cosine similarity scores
        scores = cosine_similarity(tfidf[-1],tfidf)
        # identify the most closest sentence
        self.current_sent_idx = scores.argsort()[0][-2]
        # find the corresponding score value
        scores = scores.flatten()
        scores.sort()
        value = scores[-2]
        # if there is matching sentence
        if value != 0:
            print("ChatBot >>  " + self.sentences[self.current_sent_idx])
        # if no sentence is matching the query
```

```
# instantiate an object
wiki = ChatBot()
# call chat method
wiki.chat()
```

```
Initializing ChatBot ...
Type "bye" or "quit" or "exit" to end chat

Enter your topic of interest when prompted.
ChaBot will access Wikipedia, prepare itself to
respond to your queries on that topic.

ChatBot will respond with short info.
If you input "more", it will give you detailed info
You can also jump to next query
--------------------------------------------------
ChatBot >>  Hello, Great day! Please give me a topic of your interest.
User    >> Natural language processing
ChatBot >>  Topic is "Wikipedia: Natural language processing". Let's chat!
User    >> what is turing test
ChatBot >>  Already in 1940, Alan Turing published an article titled " Computing Machinery and Intelligence " which proposed what is now called the Turing test as a criterion o
User    >> explain lemmatization
ChatBot >>  Lemmatization is another technique for reducing words to their normalized form.
User    >> what is the difference between stemming and lematization??
ChatBot >>  Stemming yields similar results as lemmatization, but does so on grounds of rules, not a dictionary.
User    >> what is neural machine translation
ChatBot >>  Neural machine translation , based on then-newly-invented sequence-to-sequence transformations, made obsolete the intermediate steps, such as word alignment, previo
User    >> Tell me about NLP Task -Document AI
ChatBot >>  A Document AI platform sits on top of the NLP technology enabling users with no prior experience of artificial intelligence, machine learning or NLP to quickly tra
User    >> more
ChatBot >>  A Document AI platform sits on top of the NLP technology enabling users with no prior experience of artificial intelligence, machine learning or NLP to quickly tra
User    >> Explain cognitive linguistics
ChatBot >>  Cognitive linguistics is an interdisciplinary branch of linguistics, combining knowledge and research from both psychology and linguistics.
User    >> ok bye
ChatBot >>  I am not sure. Sorry!
User    >> bye
ChatBot >>  See you soon! Bye!

Quitting ChatBot ...
```

```
                # exclude references, superscripts, formattings
                if i.name != 'sup' and i.string != None:
                    stripped = ' '.join(i.string.strip().split())
                    # collect data pieces
                    a.append(stripped)
            # with collected string pieces formulate a single string
            # each string is a paragraph
            self.text_data.append(' '.join(a))

        # obtain sentences from paragraphs
        for i,para in enumerate(self.text_data):
            sentences = nltk.sent_tokenize(para)
            self.sentences.extend(sentences)
            # for each sentence, its para index must be known
            # it will be useful in case user prompts "more" info
            index = [i]*len(sentences)
            self.para_indices.extend(index)

        # extract h1 heading tag from soup object
        self.title = soup.find('h1').string
        # turn respective flag on
        self.got_topic = True
```