```
from flask import Flask, render_template, request, jsonify
import aiml as aiml
import os
#HP1
kernel = aiml.Kernel()


#Reference : https://github.com/parulnith/Building-a-Simple-Chatbot-in-Python-using-NLTK,

import nltk
import warnings
import pandas as pd
import numpy as np
import scipy.sparse
import re
from capstone1 import LemNormalize, Answer_id, df_answers

warnings.filterwarnings("ignore")

import pickle
#nltk.download() # for downloading packages

import random
import string # to process standard python strings
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity


import os
import pymysql
import pandas as pd

l1 = pickle.load( open( "tfidf.dat", "rb" ) )
l2 = pickle.load( open( "tfidf_2.dat", "rb" ) )

GREETING_INPUTS = ("hello", "hi", "greetings", "sup", "what's up","hey",)
GREETING_RESPONSES = ["hi", "hey", "*nods*", "hi there", "hello", "I am glad! You are ta
# Checking for greetings

def greeting(sentence):
    """If user's input is a greeting, return a greeting response"""
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)
stopwords = ['ourselves', 'hers', 'between', 'yourself', 'but', 'again', 'there', 'about




def weight(user_response):
    last_query1=(user_response[-1].split(" "))
    last_query=[]
    for word in last_query1:
        if word not in stopwords:
            last_query.append(word)
    query_words_df=pd.DataFrame(columns=last_query,data=np.full(shape=(1,len(last_query)
    i=len(user_response)-1
    j=2
```

```
    while i:
        temp=user_response[i-1].split(" ")
        for word in temp:
            if word in query_words_df.columns:
                if word not in stopwords:
                    query_words_df[word]=query_words_df[word]+(1/j)
            else:
                if word not in stopwords:
                    query_words_df[word]=1/j
        i=i-1
        j=j+1
    return query_words_df

#pickle.dump([tfidf,TfidfVec.get_feature_names(),sentence,Answer_id], open( "tfidf.dat",


def user_response_matrix(user_response):
    l1 = pickle.load( open( "tfidf.dat", "rb" ) )
    names = [x.encode('UTF8') for x in l1[1]]
    #names = names[0:-1]
    print(type(l1[1]))
    print(l1[1])
    print(names)
    TfidfVec_response = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    user_response_new=nltk.sent_tokenize(user_response)
    tfidf_response = TfidfVec_response.fit_transform(user_response_new)
    query_words_df=pd.DataFrame(columns=TfidfVec_response.get_feature_names(), data=tfid
    #creating dataframe from tfidf

    df_all=pd.DataFrame(np.zeros((1,len(names))),columns=names)
    for col in query_words_df.columns:
        for col2 in df_all.columns:
            if col==col2:
                df_all[col2]=query_words_df[col]
    tfidf_response=scipy.sparse.csr_matrix(df_all)
    return tfidf_response

def user_response_matrix_secondary(user_response):
    l2 = pickle.load( open( "tfidf_2.dat", "rb" ) )
    print(type(l2[1]))
    print(l2[1])
    TfidfVec_response_2 = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
    user_response_new_2=nltk.sent_tokenize(user_response)
    tfidf_response_2 = TfidfVec_response_2.fit_transform(user_response_new_2)
    query_words_df_2=pd.DataFrame(columns=TfidfVec_response_2.get_feature_names(), data=
    #creating dataframe from tfidf

    df_all_2=pd.DataFrame(np.zeros((1,len(l2[1]))),columns=l2[1])
    for col in query_words_df_2.columns:
        for col2 in df_all_2.columns:
            if col==col2:
                df_all_2[col2]=query_words_df_2[col]
    tfidf_response_2=scipy.sparse.csr_matrix(df_all_2)
    return tfidf_response_2

def response(user_response):
    #query_words_df=weight(user_response)
    robo_response=''
    tfidf_response_matrix=user_response_matrix(user_response)
    vals = cosine_similarity(tfidf_response_matrix, l1[-1])
```

```
    idx=vals.argsort()[0][-1]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-1]
    if(req_tfidf==0):
        tfidf_response_matrix_2=user_response_matrix_secondary(user_response)
        vals_2 = cosine_similarity(tfidf_response_matrix_2, l2[-1])
        idx_2=vals_2.argsort()[0][-1]
        flat_2 = vals_2.flatten()
        flat_2.sort()
        req_tfidf_2 = flat_2[-1]
        if(req_tfidf_2==0):
            robo_response=robo_response+"I am sorry! I don't understand you"
            return robo_response
        else:
            robo_response=robo_response+sentence_2[idx_2]+" \n\n Visit This URL for more
            return robo_response
    else:
        ans_id=Answer_id[idx]
        robo_response = robo_response+ df_answers[df_answers['answer_id']==ans_id]['answ
        return robo_response



flag=True
print("ROBO: My name is Robo. I will answer your queries about Chatbots. If you want to
```