

# TITLE:CREATE A CHATBOT USING PYTHON

## PHASE 5: FINAL SUBMISSION

### PROBLEM DEFINITION AND DESIGN THINKING OF CHATBOT:

#### CHATBOT:

**Problem Definition:** The first step in designing a chatbot using design thinking is to clearly define the problem it aims to solve. This involves understanding the pain points, needs, and goals of the users and the organization. For example, if you're designing a customer support chatbot, the problem might be long wait times for customer inquiries and the need to provide efficient support.

#### Design Thinking Process:

##### 1.Empathize:

- Conduct user research to understand the needs and preferences of your target audience.

- Create user personas to represent different types of users.
- Gather insights by conducting interviews, surveys, and observing user behavior.

## **2. Define:**

- Clearly define the problem based on the insights gathered during the empathize phase.
- Create a problem statement that summarizes the challenge the chatbot will address.

## **3. Ideate:**

- Brainstorm creative solutions to the defined problem.
- Encourage team collaboration to generate a variety of ideas.
- Consider different chatbot functionalities, interfaces, and features.

## **4. Prototype:**

- Develop a low-fidelity prototype of the chatbot's interface and functionality.
- Use wireframes or mockups to visualize the user experience.
- Keep the prototype flexible for iterations based on feedback.

## **5. Test:**

- Gather user feedback on the prototype.

- Conduct usability testing to identify usability issues and refine the design.
- Iterate on the chatbot's design and functionality based on the test results.

## **6.Implement:**

- Develop the chatbot using suitable technologies and programming languages.
- Integrate it with relevant data sources and systems.
- Ensure it aligns with the design and functionality defined during earlier phases.

## **7.Evaluate:**

- Continuously monitor the chatbot's performance and user satisfaction.
- Gather data on user interactions and use analytics tools to track key metrics.

# **INNOVATION:**

## **Natural Language Understanding (NLU):**

- Parse the user's input to extract meaning and intent. This can involve techniques from natural language processing (NLP) like tokenization, part-of-speech tagging, and dependency parsing.

- Use named entity recognition (NER) to identify important entities in the text (e.g., variables, functions, objects).

## **Intent Recognition:**

- Determine the user's intent based on the parsed input. Intent recognition may involve building a domain-specific ontology or using machine learning classifiers to categorize user requests.

## **Context Management:**

- Maintain context throughout the conversation to understand references and relationships between different parts of the user's input.
- Keep track of variables, objects, and their states.

## **Code Generation:**

- Translate the user's intent into executable code or commands. This step might involve using templates or generating code from predefined patterns.
- Handle ambiguity and disambiguate the user's request when necessary.

## **Code Execution:**

- Execute the generated code or commands within a controlled environment. This environment should provide safety and security measures to prevent malicious or unintended actions.

## **Error Handling:**

- Implement error detection and handling mechanisms to gracefully handle situations where the user's request is unclear, incorrect, or leads to an error in code execution.

## **Feedback and Interaction:**

- Provide feedback to the user, such as confirming actions, reporting errors, or asking for clarification when the input is ambiguous.

## **Learning and Adaptation:**

- Continuously learn from user interactions to improve the system's understanding and response capabilities. Machine learning techniques can be employed for this purpose.

## **Integration with External Systems:**

- Integrate with relevant databases, APIs, or external systems to execute tasks that require external data or actions.

## **Security and Privacy:**

- Implement security measures to protect user data and prevent unauthorized access or actions.
- Ensure compliance with privacy regulations and user consent for data processing.

## **Performance Optimization:**

- Optimize the algorithm's performance to ensure fast response times, especially for real-time interactions.

## **Testing and Validation:**

- Thoroughly test the NPL system to identify and correct errors, handle edge cases, and refine its performance.

## **Deployment:**

- Deploy the NPL system in a production environment, monitor its performance, and maintain it over time.

Creating a fully functional NLP algorithm is a substantial undertaking, often requiring a combination of techniques from NLP, machine learning, and software engineering. Additionally, the success of such a system greatly depends on the quality of training data, the accuracy of language understanding, and the specific domain it's designed for.

Creating a problem for a chatbot using Python involves defining a scenario or task that the chatbot must assist with. Let's design a simple

**PROBLEM:**    **UNIT CONVERSION CHATBOT**

**PROBLEM STATEMENT:**

problem where you'll create a chatbot to help users convert between different units of measurement (e.g., temperature, length,

You are tasked with creating a Python-based chatbot that can assist users with unit conversions. The chatbot should be able to convert between various units, including temperature, length, and weight.

**Requirements:**

1. Your chatbot should support conversions between at least three different types of units (e.g., Celsius to Fahrenheit, meters to feet, kilograms to pounds).
2. The chatbot should be interactive, asking the user for input and providing clear instructions on how to use it.
3. The chatbot should handle invalid input gracefully and provide helpful error messages when the user enters incorrect information.
4. The chatbot should display the converted value to the user, rounded to an appropriate number of decimal places.
5. Ensure that your code is well-structured, modular, and easy to understand.

## **Example Interaction:**

**Chatbot:** Hello! I can help you with unit conversions.  
Here are some supported conversions:

1. Celsius to Fahrenheit
2. Meters to Feet
3. Kilograms to Pounds



Please enter the number of the conversion you want (e.g., 1):

User: 1

Chatbot: Great choice! Please enter the temperature in Celsius:

User: 25

Chatbot: 25.0 degrees Celsius is approximately equal to 77.0 degrees Fahrenheit.

Do you want to do another conversion? (yes/no)

User: yes

Chatbot: Please enter the number of the conversion you want (e.g., 2):

User: 2

Chatbot: Sure thing! Please enter the length in meters:

User: 10

Chatbot: 10.0 meters is approximately equal to 32.81 feet.

Do you want to do another conversion? (yes/no)

User: no

Chatbot: Thank you for using the unit conversion chatbot. Goodbye!

## **Implementation Guidelines:**

1. Use functions to modularize your code. For example, you could have separate functions for each type of unit conversion.
2. Handle user input and validation carefully to ensure the chatbot doesn't crash or produce incorrect results when given invalid input.
3. Display clear and user-friendly messages throughout the interaction.
4. You can implement the conversion formulas using Python code.

Consider rounding the converted values to a reasonable number of decimal places.

This problem allows you to practice user interaction, input validation, and mathematical calculations in Python while building a practical chatbot.

## **DEVELOPMENT PHASE PART I**

### **Introduction**

#### **Bots**

Bots are specially built software that interacts with internet users automatically. Bots are made up of algorithms that assist them in completing jobs. By auto-designed, we mean that they run on their own, following instructions, and therefore begin the conservation process without the need for human intervention.

Bots are responsible for the majority of internet traffic. For e-commerce sites, traffic can be significantly higher, accounting for up to 90% of total traffic. They can communicate with people and on social media accounts, as well as on websites.

## **TYPES OF BOT**

**1. ChatBot — An Artificial Intelligence (AI) programme that communicates with users through app, message, or phone. It is most commonly utilised by Twitter.**

2. Social Media Bot- Created for social media sites to answer automatically all at once.

3. Google Bot is commonly used for indexing and crawling. Spider Bots—Developed for retrieving data from websites, the Google Bot is widely used for indexing and crawling.

4. Spam Bots are programmed that automatically send spam emails to a list of addresses

```

import warnings
warnings.filterwarnings('ignore')
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import keras
from tqdm import tqdm
from keras.layers import Dense
import json
import re
import string
from sklearn.feature_extraction.text import TfidfVectorizer
import unicodedata
from sklearn.model_selection import train_test_split

```

```

In [2]:question =[]
        answer = []
        with open("../input/simple-dialogs-for-chatbot/dialogs.txt",'r') as f :
        for line in f :
            line = line.split('\t')
            question.append(line[0])
            answer.append(line[1])
print(len(question) == len(answer))

```

True

In [3]:question[:5]

```

Out[3]:
['hi, how are you doing?', "i'm pretty good. thanks for asking.",
 'no problem. so how have you been?',
 "i've been great. what about you?"]

```

In [4]:

```

answer[:5]Out[4]:
["i'm fine. how about yourself?\n",
 "i'm pretty good. thanks for asking.\n",
 'no problem. so how have you been?\n',
 "i've been great. what about you?\n",
 "i've been good. i'm in school right now.\n"]

```

In [5]:

```

answer = [ i.replace("\n","") for i in answer]

```

In [6]:

```

answer[:5]
["i'm fine. how about yourself",
 "i'm pretty good. thanks for asking.",
 'no problem. so how have you been?',
 "i've been great. what about you?",
 "i've been good. i'm in school right now."]

```

```
data = pd.DataFrame({"question" : question , "answer":answer})
data.head()
```

Out[7]:

	Question	answer
0	hi, how are you doing?	i'm fine. how about yourself?
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking.
2	i'm pretty good. thanks for asking.	no problem. so how have you been?
3	no problem. so how have you been?	i've been great. what about you?
4	i've been great. what about you?	i've been good. i'm in school right now.

```
In [8]:
def unicode_to_ascii(s):
    return ''.join(c for c in unicodedata.normalize('NFD', s)
                    if unicodedata.category(c) != 'Mn')
```

```
In [9]:
linkcode
def clean_text(text):
    text = unicode_to_ascii(text.lower().strip())
    text = re.sub(r"i'm", "i am", text)
    text = re.sub(r"\r", "", text)
    text = re.sub(r"he's", "he is", text)
    text = re.sub(r"she's", "she is", text)
    text = re.sub(r"it's", "it is", text)
    text = re.sub(r"that's", "that is", text)
    text = re.sub(r"what's", "that is", text)
    text = re.sub(r"where's", "where is", text)
    text = re.sub(r"how's", "how is", text)
    text = re.sub(r"\ll", " will", text)
    text = re.sub(r"\ve", " have", text)
    text = re.sub(r"\re", " are", text)
    text = re.sub(r"\d", " would", text)
    text = re.sub(r"\re", " are", text)
    text = re.sub(r"won't", "will not", text)
    text = re.sub(r"can't", "cannot", text)
    text = re.sub(r"n't", " not", text)
    text = re.sub(r"n'", "ng", text)
    text = re.sub(r"'bout", "about", text)
    text = re.sub(r"'til", "until", text)
    text = re.sub(r"[-()\"#/@;:<>{}`+=~|.!? ,]", "", text)
    text = text.translate(str.maketrans('', '', string.punctuation))
```

```

text = re.sub("(\\W)", " ", text)
text = re.sub('\\S*d\\S*\\s*', '', text)
text = "<sos> " + text + " <eos>"
return text

```

```

In [10]:
data["question"][0]

```

```

Out[10]:
'hi, how are you doing?'

```

```

In [11]:
data["question"] = data.question.apply(clean_text)

```

```

In [12]:
data["question"][0]

```

```

Out[12]:
'<sos> hi how are you doing <eos>'

```

```

In [13]:
data["answer"] = data.answer.apply(clean_text)

```

```

In [14]:
question = data.question.values.tolist()
answer = data.answer.values.tolist()

```

```

In [15]:
def tokenize(lang):
    lang_tokenizer = tf.keras.preprocessing.text.Tokenizer(
        filters='')
    lang_tokenizer.fit_on_texts(lang)
    tensor = lang_tokenizer.texts_to_sequences(lang)

    tensor = tf.keras.preprocessing.sequence.pad_sequences(tensor,
                                                            padding='post')

    return tensor, lang_tokenizer

```

```

In [16]:
input_tensor , inp_lang = tokenize(question)

```

```

In [17]:
target_tensor , targ_lang = tokenize(answer)

```

```

In [18]:
#len(inp_question) == len(inp_answer)

```

```

In [19]:
def remove_tags(sentence):
    return sentence.split("<start>")[-1].split("<end>")[0]

```

```

In [20]:
max_length_targ, max_length_inp = target_tensor.shape[1], input_tensor.shape[1]

```

```

In [21]:
# Creating training and validation sets using an 80-20 split
input_tensor_train, input_tensor_val, target_tensor_train, target_tensor_val = tra
in_test_split(input_tensor, target_tensor, test_size=0.2)

```

```

In [22]:
#print(len(train_inp) , len(val_inp) , len(train_target) , len(val_target))

```

```

In [23]:

```

```

BUFFER_SIZE = len(input_tensor_train)
BATCH_SIZE = 64
steps_per_epoch = len(input_tensor_train)//BATCH_SIZE
embedding_dim = 256
units = 1024
vocab_inp_size = len(inp_lang.word_index)+1
vocab_tar_size = len(targ_lang.word_index)+1

dataset = tf.data.Dataset.from_tensor_slices((input_tensor_train, target_tensor_train)).shuffle(BUFFER_SIZE)
dataset = dataset.batch(BATCH_SIZE, drop_remainder=True)

example_input_batch, example_target_batch = next(iter(dataset))
example_input_batch.shape, example_target_batch.shape

Out[23]:
(TensorShape([64, 22]), TensorShape([64, 22]))

In [24]:
class Encoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

    def call(self, x, hidden):
        def call(self, x, hidden):
            x = self.embedding(x)
            output, state = self.gru(x, initial_state = hidden)
            return output, state

        def initialize_hidden_state(self):
            return tf.zeros((self.batch_sz, self.enc_units))

In [25]:
encoder = Encoder(vocab_inp_size, embedding_dim, units, BATCH_SIZE)

# sample input
sample_hidden = encoder.initialize_hidden_state()
sample_output, sample_hidden = encoder(example_input_batch, sample_hidden)
print('Encoder output shape: (batch size, sequence length, units) {}'.format(sample_output.shape))
print('Encoder Hidden state shape: (batch size, units) {}'.format(sample_hidden.shape))

Encoder output shape: (batch size, sequence length, units) (64, 22, 1024)
Encoder Hidden state shape: (batch size, units) (64, 1024)

In [26]:
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

```

```

def call(self, query, values):
    # query hidden state shape == (batch_size, hidden size)
    # query_with_time_axis shape == (batch_size, 1, hidden size)
    # values shape == (batch_size, max_len, hidden size)
    # we are doing this to broadcast addition along the time axis to calculate
the score
    query_with_time_axis = tf.expand_dims(query, 1)

    # score shape == (batch_size, max_length, 1)
    # we get 1 at the last axis because we are applying score to self.V
    # the shape of the tensor before applying self.V is (batch_size, max_lengt
h, units)
    score = self.V(tf.nn.tanh(
        self.W1(query_with_time_axis) + self.W2(values)))

    # attention_weights shape == (batch_size, max_length, 1)
    attention_weights = tf.nn.softmax(score, axis=1)

    # context_vector shape after sum == (batch_size, hidden_size)
    context_vector = attention_weights * values
    context_vector = tf.reduce_sum(context_vector, axis=1)

    return context_vector, attention_weights

```

```

In [27]:
attention_layer = BahdanauAttention(10)
attention_result, attention_weights = attention_layer(sample_hidden, sample_output
)

```

```

print("Attention result shape: (batch size, units) {}".format(attention_result.sha
pe))
print("Attention weights shape: (batch_size, sequence_length, 1) {}".format(atten
tion_weights.shape))

```

```

Attention result shape: (batch size, units) (64, 1024)
Attention weights shape: (batch_size, sequence_length, 1) (64, 22, 1)

```

```

In [28]:
class Decoder(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()
        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                         return_sequences=True,
                                         return_state=True,
                                         recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(vocab_size)

        # used for attention
        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        # enc_output shape == (batch_size, max_length, hidden_size)
        context_vector, attention_weights = self.attention(hidden, enc_output)

        # x shape after passing through embedding == (batch_size, 1, embedding_dim
)
        x = self.embedding(x)

```



```

ze)    # x shape after concatenation == (batch_size, 1, embedding_dim + hidden_si
x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

    # passing the concatenated vector to the GRU
    output, state = self.gru(x)

    # output shape == (batch_size * 1, hidden_size)
    output = tf.reshape(output, (-1, output.shape[2]))

    # output shape == (batch_size, vocab)
    x = self.fc(output)

    return x, state, attention_weights

```

```

In [29]:
optimizer = tf.keras decoder = Decoder(vocab_tar_size, embedding_dim, units, BATCH
_SIZE)

```

```

sample_decoder_output, _, _ = decoder(tf.random.uniform((BATCH_SIZE, 1)),
                                     sample_hidden, sample_output)

```

```

print ('Decoder output shape: (batch_size, vocab size) {}'.format(sample_decoder_o
utput.shape))

```

```

Decoder output shape: (batch_size, vocab size) (64, 2347)

```

```

In [30]:
.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

```

```

def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)

```

```

In [31]:
@tf.function
def train_step(inp, targ, enc_hidden):
    loss = 0

    with tf.GradientTape() as tape:
        enc_output, enc_hidden = encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([targ_lang.word_index['<sos>']] * BATCH_SIZE, 1
)

        # Teacher forcing - feeding the target as the next input
        for t in range(1, targ.shape[1]):
            # passing enc_output to the decoder
            predictions, dec_hidden, _ = decoder(dec_input, dec_hidden, enc_output
)

            loss += loss_function(targ[:, t], predictions)

```

```
# using teacher forcing
dec_input = tf.expand_dims(targ[:, t], 1)

batch_loss = (loss / int(targ.shape[1]))

variables = encoder.trainable_variables + decoder.trainable_variables

gradients = tape.gradient(loss, variables)

optimizer.apply_gradients(zip(gradients, variables))

return batch_loss
```

## DEVELOPMENT PART II

### PROBLEM STATEMENT: CONTINUE BUILDING THE CHATBOT BY INTEGRATING IT INTO A WEB APP USING FLASK.

#### INTRODUCTION:

Integrating a chatbot into a web app using Flask is a great way to create interactive and dynamic web applications. Here's a step-by-step guide to help you get started:

#### Step 1: Set Up Your Development Environment

#### Step 2: Create a Flask Project Structure

- **app.py** will contain your Flask application.

- **templates** will store your HTML templates.
- **static** is where you can store CSS, JavaScript, and other static files.
- **chatbot.py** will be used to implement your chatbot logic.

### **Step 3: Create the Flask Application**

### **Step 4: Create the Chatbot Logic**

### **Step 5: Create the HTML Template**

### **Step 6: Add JavaScript for Real-time Interaction**

### **Step 7: Run Your Flask Application**

## **HOW TO MAKE CHATBT IN PYTHON?**

Now we are going to build the chatbot using Flask framework but first, let us see the file structure and the type of files we will be creating:

- **data.json** – The data file which has predefined patterns and responses.
- **trainning.py** – In this Python file, we wrote a script to build the model and train our chatbot.

- **Texts.pkl** – This is a pickle file in which we store the words Python object using Nltk that contains a list of our vocabulary.
- **Labels.pkl** – The classes pickle file contains the list of categories(Labels).
- **model.h5** – This is the trained model that contains information about the model and has weights of the neurons.
- **app.py** – This is the flask Python script in which we implemented web-based GUI for our chatbot. Users can easily interact with the bot.

## **Here are the 5 steps to create a chatbot in Flask from scratch:**

- 1.Import and load the data file
- 2.Preprocess data
- 3.split the data into training and test
- 4.Build the ANN model using keras
- 5.Predict the outcomes
- 6.Deploy the model in the Flask app

## **PROJECT CODE:**

## app.py

```
import nltk

nltk.download('popular')

from nltk.stem import

WordNetLemmatizer

lemmatizer =

WordNetLemmatizer()

import ipickle import

numpy as np

    from keras.models
    import load_model
model =
load_model('model.
h5') import json
import random
```

```
intents =

json.loads(open('data.json').read())

words =

pickle.load(open('texts.pkl','rb'))

classes =

pickle.load(open('labels.pkl','rb'))

def

clean_up_sentence(sentence):

    # tokenize the pattern - split
words into array      sentence_words =

nltk.word_tokenize(sentence)      # stem
each word - create short form for word
```

```

        sentence_words =
[lemmatizer.lemmatize(word.lower()) for word
in sentence_words]        return sentence_words

# return bag of words array: 0 or 1 for each
word in the bag that exists in the sentence

def bow(sentence, words,
show_details=True):

    # tokenize the pattern

sentence_words =

clean_up_sentence(sentence)

    # bag of words - matrix of N words,
vocabulary matrix        bag = [0]*len(words)

    for s in sentence_words:

```

```
        for i,w in
enumerate(words):
    if w == s:
```

```
return return_list
```

```
def getResponse(ints,
intents_json):
```

```
    tag = ints[0]['intent']

list_of_intents =

intents_json['intents']        for i

in list_of_intents:

    if(i['tag']== tag):

        result =

        random.choice(i['responses'])

    break        return result
```



```
def

chatbot_response(msg):

    ints = predict_class(msg,

model)    res =

getResponse(ints, intents)

return res


from flask import Flask,

render_template, request


app = Flask(__name__)
app.static_folder =
'static'
```

```
@app.route("/")

def home():

    return render_template("index.html")


@app.route("/get")

def

get_bot_response():

    userText =

request.args.get('msg')

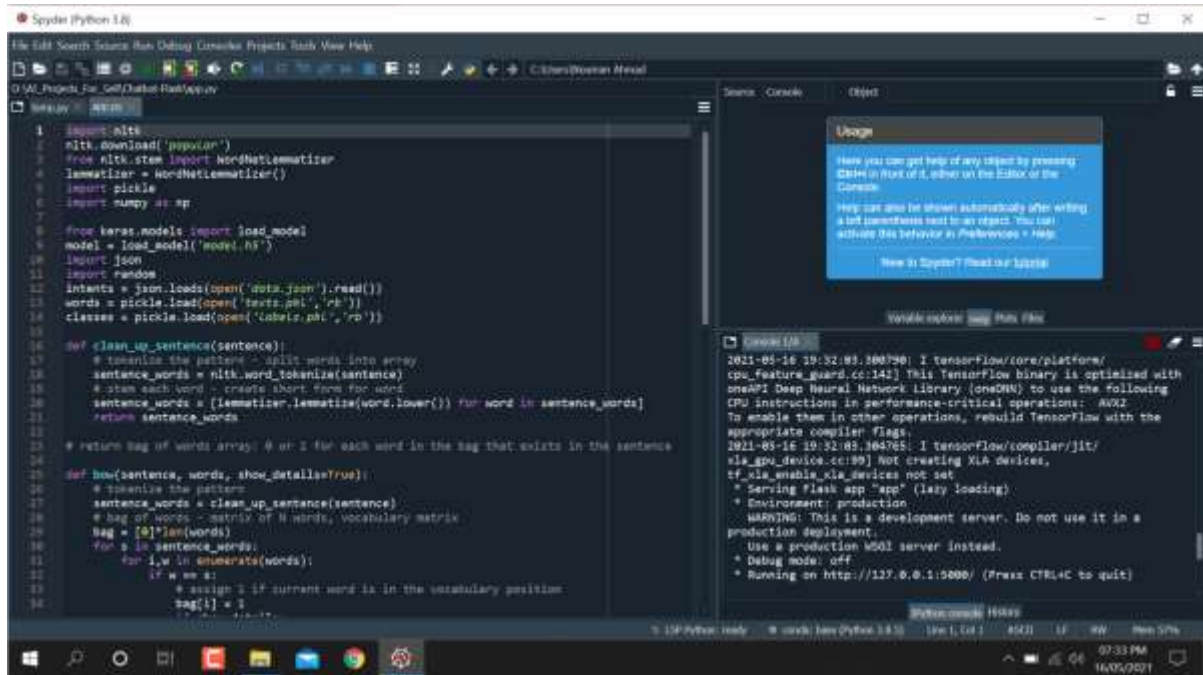
return

chatbot_response(userText)
```

```
if __name__ ==  
  
"__main__":  
  
    app.run()
```

**OUTPUT :**

**Run Flask App**

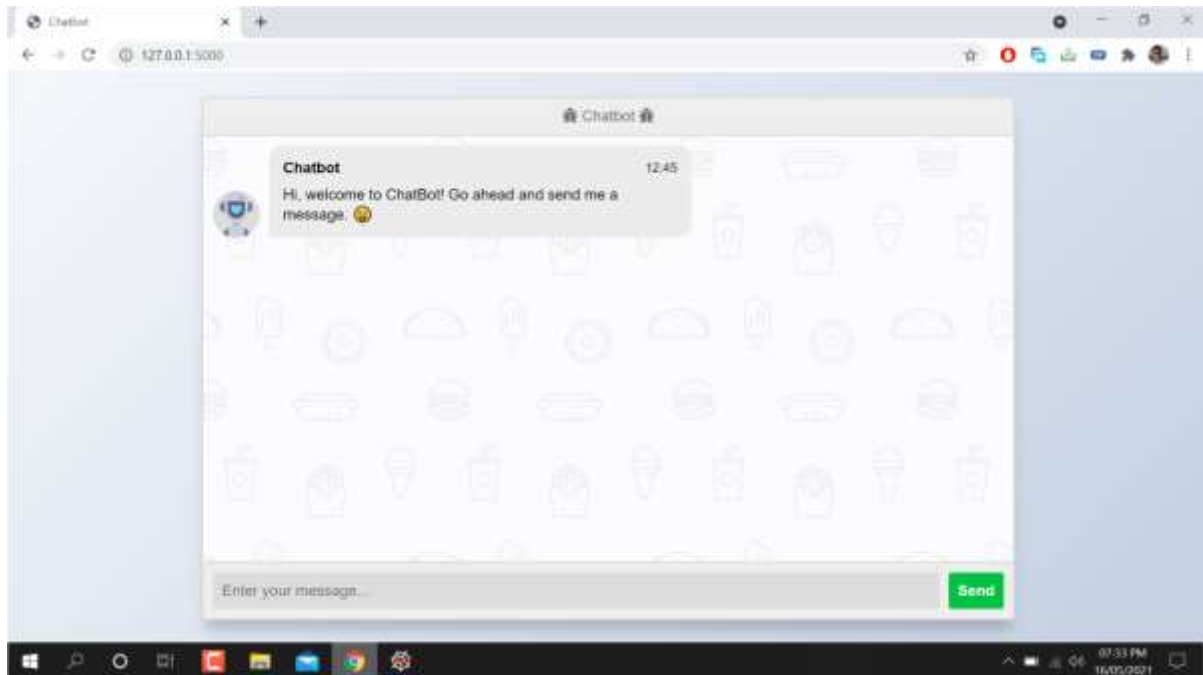


The image shows the Spyder Python IDE interface. The main editor window contains Python code for a chatbot. The code imports necessary libraries, loads a model, and defines functions for cleaning up sentences and processing input. The console window on the right shows the output of the code, indicating that the chatbot is running successfully on a local server.

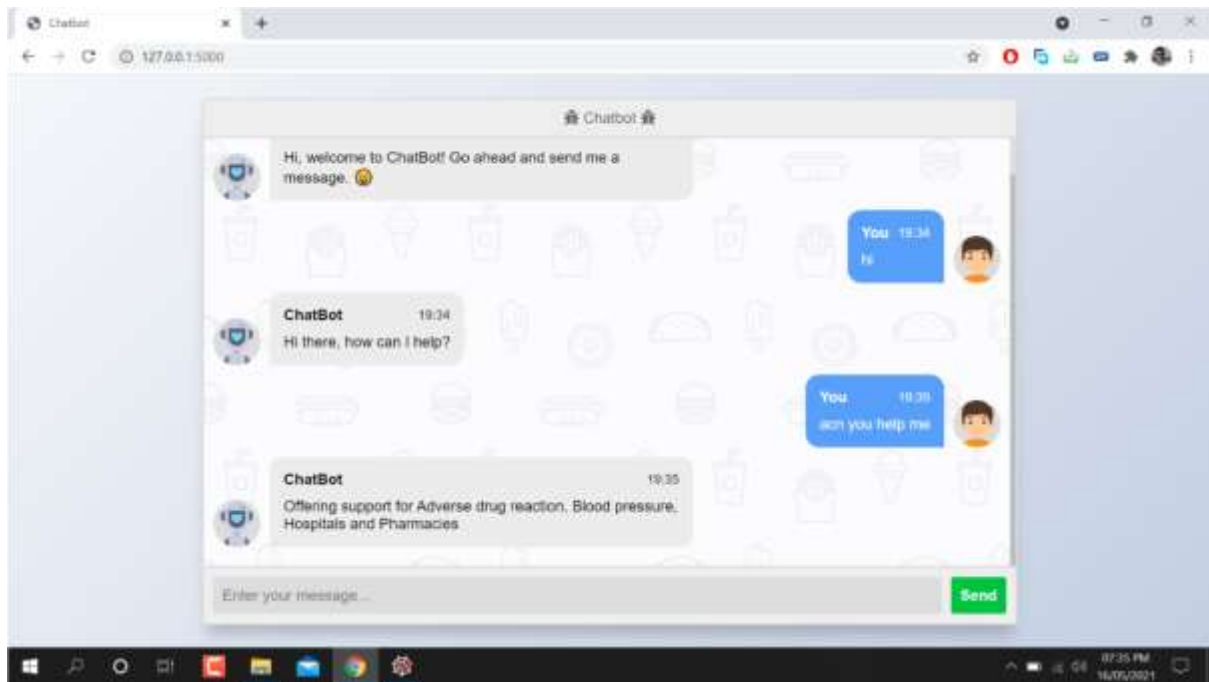
```
1 import nltk
2 nltk.download('popular')
3 from nltk.stem import WordNetLemmatizer
4 lemmatizer = WordNetLemmatizer()
5 import pickle
6 import numpy as np
7
8 from keras.models import load_model
9 model = load_model('model.h5')
10 import json
11 import random
12 intents = json.loads(open('data.json').read())
13 words = pickle.load(open('words.pkl', 'rb'))
14 classes = pickle.load(open('labels.pkl', 'rb'))
15
16 def clean_up_sentence(sentence):
17     # tokenize the pattern - split words into array
18     sentence_words = nltk.word_tokenize(sentence)
19     # stem each word - create short form for word
20     sentence_words = [lemmatizer.lemmatize(word.lower()) for word in sentence_words]
21     return sentence_words
22
23 # return bag of words array: 0 or 1 for each word in the bag that exists in the sentence
24
25 def bow(sentence, words, show_details=True):
26     # tokenize the pattern
27     sentence_words = clean_up_sentence(sentence)
28     # bag of words - matrix of N words, vocabulary matrix
29     bag = [0]*len(words)
30     for s in sentence_words:
31         for i,w in enumerate(words):
32             if w == s:
33                 # assign 1 if current word is in the vocabulary position
34                 bag[i] = 1
35     return bag
```

Console Output:

```
2021-05-16 19:32:05.366790: I tensorflow/core/platform/cpu_feature_guard.cc:143] This TensorFlow binary is optimized with
oneAPI Deep Neural Network library (oneDNN) to use the following
CPU instructions in performance-critical operations: AVX2
To enable them in other operations, rebuild TensorFlow with the
appropriate compiler flags.
2021-05-16 19:32:05.364765: I tensorflow/compiler/jit/
xla_gpu_device.cc:99] Not creating XLA devices,
tf_xla_enable_xla_devices not set
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a
production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```



**BOT RESPONSE**



## CONCLUSION:

This article is the base of knowledge of the definition of ChatBot, its importance in the Business, and how we can build a simple Chatbot by using Python and Library Chatterbot.