## File Naming Convention: Chatbot in python\_phase\_5

## How To Make A Chabot In Python?

You may have this question in your mind, how to create a chatbot? We’ll take a step by step approach and break down the process of building a Python Chabot. To build a chatbot in Python, you have to import all the necessary packages and initialize the variables you want to use in your chatbot project. Also, remember that when working with text data, you need to perform data preprocessing on your dataset before designing an ML model.This is where tokenizing helps with text data – it helps fragment the large text dataset into smaller, readable chunks (like words). Once that is done, you can also go for lemmatization that transforms a word into its lemma form. Then it creates a pickle file to store the python objects that are used for predicting the responses of the bot. Another vital part of the chatbot development process is creating the training and testing datasets. Now that we’ve covered the basics of chatbot development in Python, let’s dive deeper into the actual process! It will help you understand how to create a chatbot.

## Summary

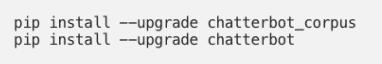
In this Article, you will learn about How to Make a Chatbot in Python Step By Step.

1. Prepare the Dependencies
2. Import Classes
3. Create and Train the Chatbot
4. Communicate with the Python Chatbot
5. Train your Python Chatbot with a Corpus of Data

### **1. Prepare the Dependencies**

The first step in creating a chatbot in Python with the ChatterBot library is to install the library in your system. It is best if you create and use a new Python virtual environment for the installation. To do so, you have to write and execute this command in your Python terminal:

You can also install ChatterBot’s latest development version directly from GitHub. For this, you will have to write and execute the following command:pip install git+git://github.com/gunthercox/ChatterBot.git@master If you wish to upgrade the command, you can do so as well:

Now that your setup is ready, we can move on to the next step to create chatbot using python.

### **2**. **Import Classes**

Importing classes is the second step in the Python chatbot creation process. All you need to do is import two classes – ChatBot from chatterbot and List Trainer from chatterbot.trainers. To do this, you can execute the following command:



### **3.** **Create and Train the Chatbot**

This is the third step on creating chatbot in python. The chatbot you are creating will be an instance of the class “ChatBot.” After creating a new Chatterbot instance, you can train the bot to improve its performance. Training ensures that the bot has enough knowledge to get started with specific responses to specific inputs. You have to execute the following command now:

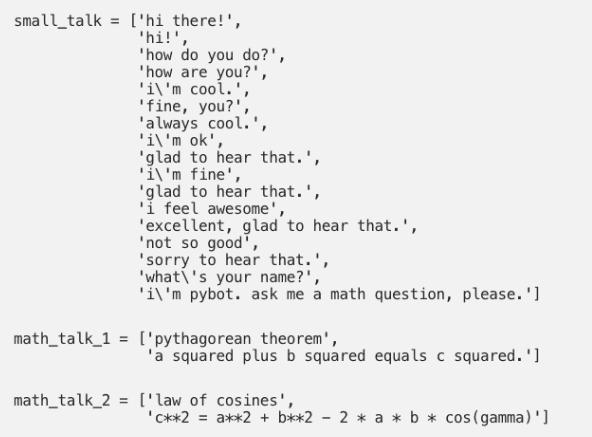


Here, the argument (that corresponds to the parameter name) represents the name of your Python chatbot. If you wish to disable the bot’s ability to learn after the training, you can include the “read\_only=True” command. The command “logic\_adapters” denotes the list of adapters used to train the chatbot.

While the “chatterbot.logi

c.MathematicalEvaluation” helps the bot to solve math problems, the “chatterbot.logic.BestMatch” helps it to choose the best match from the list of responses already provided.

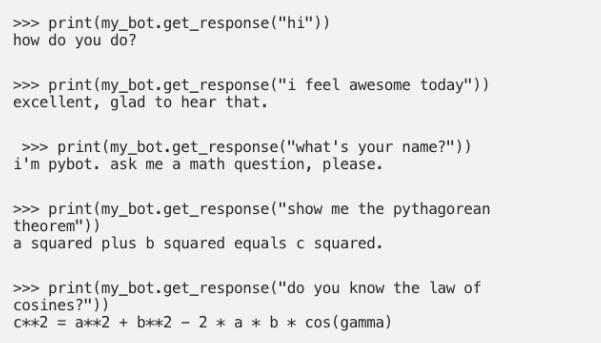
Since you have to provide a list of responses, you can do it by specifying the lists of strings that can be later used to train your Python chatbot, and find the best match for each query. Here’s an example of responses you can train your chatbot using python to learn:

You can also create and train the bot by writing an instance of “List Trainer” and supplying it with a list of strings like so:

Now, your Python chatbot is ready to communicate.

### **4. Communicate with the Python Chatbot**

To interact with your Python chatbot, you can use the .get\_response() function. This is how it should look while communicating:



However, it is essential to understand that the chatbot using python might not know how to answer all your questions. Since its knowledge and training is still very limited, you have to give it time and provide more training data to train it further.

### **5. Train your Python Chatbot with a Corpus of Data**

In this last step of how to make a chatbot in Python, for training your python chatbot even further, you can use an existing corpus of data. Here’s an example of how to train your Python chatbot with a corpus of data provided by the bot itself:

The good thing is that Chatterbot offers this functionality in many different languages. So, you can also specify a subset of a corpus in a language you would prefer. This is how we create Chabot in Python.

### **Benefits of a Chatbot**

We have covered how to make a chatboat in this article. Now we must understand its benefits to grasp its full utilization. Chabot’s Programming is very useful, especially when it comes to building good relationships with customers. Strong connections can be built with the help of chatboats because it helps you to interact with the visitors of your website directly. With the help of chatbot programming, you not only achieve all the marketing goals but also increase sales and better customer service.

Chatbot programming comes with several benefits –

1. **Increase in customer interaction** – Businesses may increase consumer interaction by using chatbots. Customer interaction may be pushed based on customer data and enhanced engagement by deploying conversational AI chatbots. This process is very important after you understand how to make a chatbot. Additionally, since bots may provide regular responses, you can avoid giving clients useless information. Additionally, clients are more likely to stay on your website more and continue the dialogue if they receive pertinent and prompt replies.
2. **Generating more leads** – The ability to engage consumers with individualized messaging during their journey is a strength of today’s highly evolved bots. In reality, your company may employ chatbots to help clients make rapid decisions by generating leads. In order to convince the consumers of lead generation and greater conversion rates.
3. **Consumer service costs are reduced** – Chatbot implementation is an investment in reducing the cost of customer support. You may save extra money by investing in chatbots instead of employing more support staff. A firm may cut expenses in a variety of ways with chatbots:

Salaries

. Infrastructure

Training

Similarly, using AI chatbots for customer care has various advantages;

* Reduce chat availability at peak times.
* Automate common business operations
* Allow for new customer service approaches

Additionally, AI-bots may be expanded without incurring any additional expenditures during business peaks. In addition, bots are cost-saving and improve the bottom line by ensuring that clients have an easier and more consistent brand experience.

1. **Monitor data to get consumer insights** – By tracking user data, a chatbot project is a great way to study customer behavior and follow purchase trends.  Using this information, businesses may sell their products in new ways and reach a wider audience.  Similar to this, bots may be employed to gather input via straightforward questions, improve services, or enhance websites.
2. **It helps with a Conversational Marketing Strategy** – Real-time customer interactions are at the heart of conversational marketing, which aims to assist and drive customers along the sales funnel. It may assist your company in developing genuine consumer interactions and real experiences. By engaging with customers through customized messaging and clever chatbots, firms using chatbots for marketing may increase conversion rates.
3. **Enhance customer services and meet consumer expectations** – Customers anticipate prompt replies to their inquiries or concerns. As chatbots become more intelligent, businesses may utilize them to satisfy customers’ expectations by automating some of the typical interactions. The goal remains the same, whether you employ an Android chatbot: to provide value to consumers through automation. Customers won’t have to wait for humans to communicate with them thanks to automation, which is one of the chatbot’s most important use cases. This will undoubtedly improve their experience with the business and increase retention.
4. **Create a Smooth Customer Experience** – No firm wants to lose prospects midway through a sales funnel since that is never a desirable scenario. Customers may choose not to buy for a variety of reasons, such as a lengthy checkout procedure or unstated fees. Chatbots can be useful in assisting businesses in lowering customer turnover and improving the customer experience. By writing the proper bot scripts, you may successfully interact with clients at various points in their journeys and respond to their questions

**Set Up Your Environment:**

**1**.Ensure you have Python installed (preferably Python 3.x).

Create a virtual environment to manage your project dependencies.

**Install Necessary Libraries:**

**2**.Install the Transformers library for GPT-3 integration (you can use Hugging Face’s transformers library).

**[pip install transformers]**

**Install Flask for web app development.**

**[pip install flask]**

**3.Acquire GPT-3 API Key:**

* **Sign up for GPT-3 access from OpenAI and obtain your API key.**

**4.Create a Flask Web App:**

* **Set up a Flask project by creating a Python script for your application.**
* **Define routes for handling user interactions, such as sending user messages to the GPT-3 model and receiving responses**

**5. Integrate GPT-3:**

* **Use the Transformers library to interact with the GPT-3 model via its API. You’ll need to make POST requests to** [**https://api.openai.com/v1/engines/gpt-3.5-turbo/completions**](https://api.openai.com/v1/engines/gpt-3.5-turbo/completions) **using your API key.**

**6.Process User Input:**

* **Receive and pre-process user input from the web interface.**
* **Prepare the user’s message to be sent as a prompt to the GPT-3 model.**

**7.Send User Input to GPT-3:**

* **Make a POST request to the GPT-3 API with the user’s message as input.**
* **Receive and extract the model’s response.**

**8.Display Responses:**

* **Send the model’s response back to the user through the web interface.**

**9.Iterate and Refine:**

* **Continuously test and improve your chatbot by refining its responses and handling of user interactions.**

**Remember to handle any errors and exceptions, and ensure that you secure your API keys properly. Also, consider implementing user authentication if needed, and think about how to handle different conversation contexts. This is a high-level overview; the specific implementation details may vary depending on your project’s requirements.**

**10.Build one for you using Python**

**Now we have an immense understanding of the theory of chatbots and their advancement in the future. Let’s make our hands dirty by building one simple rule-based chatbot using python for ourselves.**

**We will design a simple GUI using the Python Tkinter module using which we will create a text box and button to submit user intent and on the action, we will build a function where we will match the user intent and respond to him on his intent. If you do not have the Tkinter module install, then first install it using the pip command.**

**Program**

pip install tkinter

from tkinter import \* root = Tk() root.title("Chatbot") def send():

send = "You -> "+e.get() txt.insert(END, "n"+send) user = e.get().lower() if(user == "hello"):

txt.insert(END, "n" + "Bot -> Hi") elif(user == "hi" or user == "hii" or user == "hiiii"):

txt.insert(END, "n" + "Bot -> Hello") elif(e.get() == "how are you"):

txt.insert(END, "n" + "Bot -> fine! and you") elif(user == "fine" or user == "i am good" or user == "i am doing good"):

txt.insert(END, "n" + "Bot -> Great! how can I help you.") else:

txt.insert(END, "n" + "Bot -> Sorry! I dind't got you")

e.delete(0, END) txt = Text(root)

txt.grid(row=0, column=0, columnspan=2) e = Entry(root, width=100)

e.grid(row=1, column=0)

send = Button(root, text="Send", command=send).grid(row=1, column=1) root.mainloop()

**Explanation** – First we have created a blank window, After that, we created a text field using the entry method and a Button widget which on triggering calls the function send, and in return, it gets the chatbot response. We have used a basic If-else control statement to build a simple rule-based chatbot. And you can interact with the chatbot by running the application from the interface and you can see the output as below figure.



Implementing Chatbot using Python NLTK Library

NLTK stands for Natural language toolkit used to deal with NLP applications and chatbot is one among them. Now we will advance our Rule-based chatbots using the NLTK library. Please install the NLTK library first before working using the pip command.

**[pip instal nltk]**

First thing is to import the library and classes we need to use.

First thing is to import the library and classes we need to use.

import nltk from nltk.chat.util import Chat, reflections

1. ***Chat –*** Chat is a class that contains complete logic for processing the text 2 data which the chatbot receives and find useful information out of it.

***reflections –*** Another import we have done is reflections which is a dictionary containing basic input and corresponding outputs. You can also create your own dictionary with more responses you want. if you print reflections it will be something like this.

reflections = {

"i am" : "you are",

"i was" : "you were",

"i" : "you",

"i'm" : "you are",

"i'd" : "you would",

"i've" : "you have",

"i'll" : "you will",

"my" : "your",

"you are" : "I am",

"you were" : "I was",

"you've" : "I have",

"you'll" : "I will",

"your" : "my",

"yours" : "mine",

"you" : "me", "me" : "you"

} let’s start building logic for the NLTK chatbot.

After importing the libraries, First, we have to create rules. The lines of code given below create a simple set of rules. the first line describes the user input which we have taken as raw string input and the next line is our chatbot response. You can modify these pairs as per the questions and answers you want.

pairs = [ [

r"my name is (.\*)",

["Hello %1, How are you today ?",]

], [

r"hi|hey|hello",

["Hello", "Hey there",]

], [

r"what is your name ?",

["I am a bot created by Analytics Vidhya. you can call me crazy!",] ],

[

r"how are you ?",

["I'm doing goodnHow about You ?",]

], [

r"sorry (.\*)",

["Its alright","Its OK, never mind",]

], [

r"I am fine",

["Great to hear that, How can I help you?",]

], [

r"i'm (.\*) doing good",

["Nice to hear that","How can I help you?:)",]

], [

r"(.\*) age?",

["I'm a computer program dudenSeriously you are asking me this?",] ], [

r"what (.\*) want ?",

["Make me an offer I can't refuse",]

], [

r"(.\*) created ?",

["Raghav created me using Python's NLTK library ","top secret ;)",]

],

[

r"(.\*) (location|city) ?",

['Indore, Madhya Pradesh',]

], [

r"how is weather in (.\*)?",

["Weather in %1 is awesome like always","Too hot man here in %1","Too cold man here in %1","Never even heard about %1"]

], [

r"i work in (.\*)?",

["%1 is an Amazing company, I have heard about it. But they are in huge loss these days.",]

], [

r"(.\*)raining in (.\*)",

["No rain since last week here in %2","Damn its raining too much here in %2"]

], [

r"how (.\*) health(.\*)",

["I'm a computer program, so I'm always healthy ",]

], [

r"(.\*) (sports|game) ?",

["I'm a very big fan of Football",]

], [

r"who (.\*) sportsperson ?",

["Messy","Ronaldo","Roony"]

], [

r"who (.\*) (moviestar|actor)?",

["Brad Pitt"]

], [

r"i am looking for online guides and courses to learn data science, can you suggest?",

["Crazy\_Tech has many great articles with each step explanation along with code, you can explore"]

], [ r"quit",

["BBye take care. See you soon :) ","It was nice talking to you. See you soon :)"]

],

]

After creating pairs of rules, we will define a function to initiate the chat process. The function is very simple which first greet the user, and ask for any help. And the conversation starts from here by calling a Chat class and passing pairs and reflections to it. def chat():

Print def chat():

print("Hi! I am a chatbot created by Analytics Vidhya for your service")

chat = Chat(pairs, reflections) chat.converse() #initiate the conversation if \_\_name\_\_ == "\_\_main\_\_":

chat()

Hi! I am a chatbot created by Analytics Vidhya for your service”)

Chat = Chat(pairs, reflections)

Chat.converse()

#initiate the conversation

If \_\_name\_\_ == “\_\_main\_\_”:

Chat()

Dataset :Kaggle https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

**ChatBot Using Python**

**Definition** :

Chat Bot is a Python library that is designed to deliver automated responses to user inputs. It makes use of a combination of ML algorithms to generate many different types of responses .At the most basic level, a chat bot is a computer program that simulates and processes human conversation (either written or spoken), allowing humans to interact with digital devices as if they were communicating with a real person.

Prepare the Date :

**import tensorflow as tf**

**import matplotlib.pyplot as plt import matplotlib.ticker as ticker**

**from sklearn.model\_selection import train\_test\_split**

**import unicodedata import re**

**import numpy as np import os import io import time**  **import warnings**

**warnings.filterwarnings('ignore')**

file = open('../input/simple-dialogs-forchatbot/dialogs.txt','r').read()

qna\_list = [f.split('\t') for f in file.split('\n')] questions = [x[0] for x in qna\_list]answers = [x[1] for x in qna\_list]

print("Question: ", questions[0])print("Answer: ", answers[0])

Preprocess sentences : def unicode\_to\_ascii(s):

return ''.join(c for c in unicodedata.normalize('NFD', s) if unicodedata.category(c) != 'Mn')

def preprocess\_sentence(w):

w = unicode\_to\_ascii(w.lower().strip()) w = re.sub(r"([?.!,¿])", r" \1 ", w) w = re.sub(r'[" "]+', " ", w) w = re.sub(r"[^a-zA-Z?.!,¿]+", " ", w) w = w.strip()

w = '<start> ' + w + ' <end>' return w print(preprocess\_sentence(questions[0]))print(preprocess\_sentence(a nswers[0]))

pre\_questions = [preprocess\_sentence(w) for w in questions]pre\_answers = [preprocess\_sentence(w) for w in answers]

Tokenize :

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 def load\_dataset(data, num\_examples=None): # creating cleaned input, output pairs if(num\_examples != None):

targ\_lang, inp\_lang, = data[:num\_examples] else:

targ\_lang, inp\_lang, = data

input\_tensor, inp\_lang\_tokenizer = tokenize(inp\_lang) target\_tensor, targ\_lang\_tokenizer = tokenize(targ\_lang) return input\_tensor, target\_tensor, inp\_lang\_tokenizer, targ\_lang\_tokenizer

num\_examples = 30000data = pre\_answers, pre\_questionsinput\_tensor, target\_tensor, inp\_lang, targ\_lang = load\_dataset(data, num\_examples)

# Calculate max\_length of the target tensorsmax\_length\_targ, max\_length\_inp = target\_tensor.shape[1], input\_tensor.shape[1] In [10]:

# Creating training and validation sets using an 80-20 splitinput\_tensor\_train, input\_tensor\_val, target\_tensor\_train, target\_tensor\_val = train\_test\_split(input\_tensor, target\_tensor, test\_size=0.2)

# Show lengthprint(len(input\_tensor\_train), len(target\_tensor\_train), len(input\_tensor\_val), len(target\_tensor\_val))

Word to index :

def convert(lang, tensor): for t in tensor: if t!=0:

print ("%d ----> %s" % (t, lang.index\_word[t])) print ("Input Language; index to word mapping")convert(inp\_lang, input\_tensor\_train[0])print ()print ("Target Language; index to word mapping")convert(targ\_lang, target\_tensor\_train[0])

Create Tensorflow dataset :

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

Encoder/Decoder with attention equations :

Encoder 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): 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)) encoder = Encoder(vocab\_inp\_size, embedding\_dim, units, BATCH\_SIZE) # sample inputsample\_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))

Attention :

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\_length, 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, hiddn\_size)

context\_vector = attention\_weights \* values context\_vector = tf.reduce\_sum(context\_vector, axis=1) return context\_vector, attention\_weights attention\_layer = BahdanauAttention(10)attention\_result, attention\_weights = attention\_layer(sample\_hidden, sample\_output) print("Attention result shape: (batch size, units)

{}".format(attention\_result.shape))print("Attention weights shape:

(batch\_size, sequence\_length, 1)

{}".format(attention\_weights.shape))

Decoder :

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)

# x shape after concatenation == (batch\_size, 1, embedding\_dim + hidden\_size)

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

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\_output.shape))

Training :

optimizer = tf.keras.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\_)

@tf.functiondef 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['<start>']] \* 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 EPOCHS = 40 for epoch in range(1, EPOCHS + 1):

enc\_hidden = encoder.initialize\_hidden\_state()

total\_loss = 0 for (batch, (inp, targ)) in enumerate(dataset.take(steps\_per\_epoch)):

batch\_loss = train\_step(inp, targ, enc\_hidden) total\_loss += batch\_loss if(epoch % 4 == 0):

print('Epoch:{:3d} Loss:{:.4f}'.format(epoch, total\_loss / steps\_per\_epoch))

Evaluate :

def remove\_tags(sentence):

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

sentence = preprocess\_sentence(sentence)

inputs = [inp\_lang.word\_index[i] for i in sentence.split(' ')] inputs = tf.keras.preprocessing.sequence.pad\_sequences([inputs], maxlen=max\_length\_inp,padding='post') inputs = tf.convert\_to\_tensor(inputs) result = ''

hidden = [tf.zeros((1, units))] enc\_out, enc\_hidden = encoder(inputs, hidden) dec\_hidden = enc\_hidden

dec\_input = tf.expand\_dims([targ\_lang.word\_index['<start>']],

0) for t in range(max\_length\_targ):

predictions, dec\_hidden, attention\_weights = decoder(dec\_input,dec\_hidden,enc\_out)

# storing the attention weights to plot later on attention\_weights = tf.reshape(attention\_weights, (-1, )) predicted\_id = tf.argmax(predictions[0]).numpy() result += targ\_lang.index\_word[predicted\_id] + ' ' if targ\_lang.index\_word[predicted\_id] == '<end>': return remove\_tags(result), remove\_tags(sentence) # the predicted ID is fed back into the model dec\_input = tf.expand\_dims([predicted\_id], 0) return remove\_tags(result), remove\_tags(sentence)

Answer question : def ask(sentence):

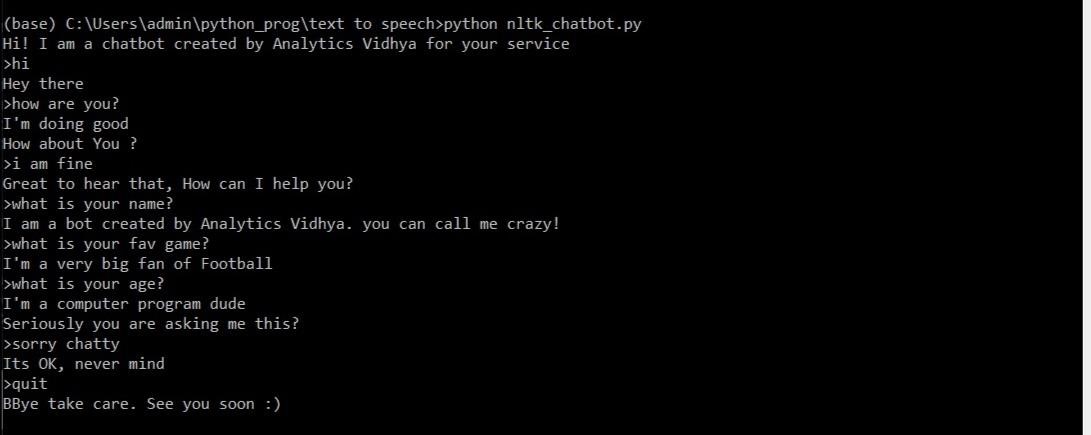
result, sentence = evaluate(sentence)

print('Question: %s' % (sentence)) print('Predicted answer: {}'.format(result))

**We have created an amazing Rule-based chatbot just by using Python and NLTK library. The nltk.chat works on various regex patterns present in user Intent and corresponding to it, presents the output to a user. Let’s run the application and chat with your created chatbot.**

**End Notes**

**Chatbots are the top application of Natural Language processing and today it is simple to create and integrate with various social medial handle and websites. Today most Chatbots are created using tools like Dialogflow, RASA, etc. This was a quick introduction to chatbots to present an understanding of how businesses are transforming using Data science and artificial Intelligence.**



**Design and Development of CHATBOT: A Review**

**Abstract.** This paper focuses on a newly emerging tool for learning from CHATBOT, whichis a learning-cum-assisted tool. A CHATBOT is an artificially created virtual entity that interacts with users using interactive textual or speech skills. This CHATBOT directly chats with the people using artificial intelligence and Machine Learning concepts. This paper reviews the technique, terminology, and different platforms used to design and develop the CHATBOT. It also presents some actual practical life typical applications and examples of CHATBOT. The utility of the CHATBOT tool for Computer-Aided Design (CAD) applications is proposed from this review.

**Keywords.** CHATBOT, Artificial Intelligence, Machine Learning, CAD.

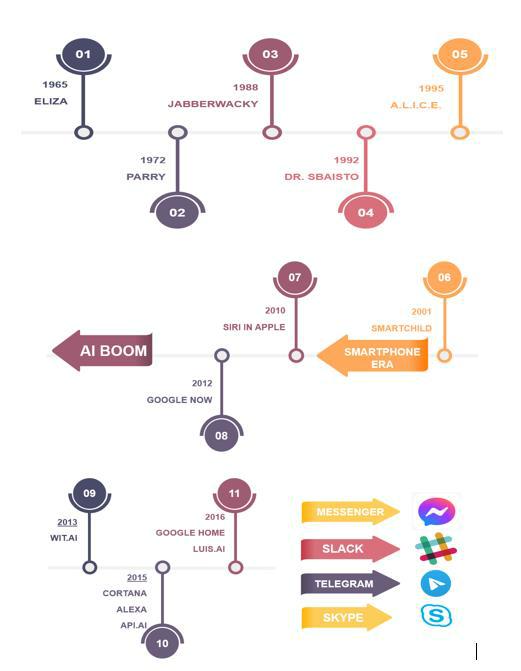
**1.** **Introduction**

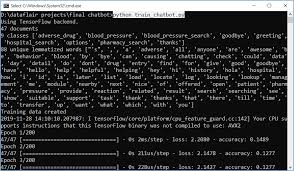
Computer-aided design packages are the primary software to help Mechanical Designers throughout the world. In the beginning, it used to replace hand-drawn technical drawings. The technology of the 21st century led engineers to develop software and hardware so that CAD [1] software and its tools integrate with computers and mobile, easy to use, affordable that way, the organization and higher institutes started incorporating these type of courses into their education curriculum. The user uses these 2D drafting and 3D models developed by the software in the form of technical design to present their design ideas and concepts to other users [2]. The most commonly used CAD software is Autodesk AutoCAD, Dassault Catia, Dassault Solidwork, Autodesk Inventor, Autodesk Fusion 360, PTC Creo, and Siemens NX. Technical drafting generates after model completion submitted for evaluation by instructors. The user mainly focuses on learning the CAD software through procedural-based knowledge, which means knowing the associated commands useful for a particular problem.

While learning CAD software for a specific CAD problem, users may face many problems or get stuck when a new CAD design problem is issue for assessment. For a new CAD problem, users try to implement the procedural manner for solving, using the same commands and tools repeatedly, which results in the neglection of new commands and tools through which the same CAD problem is dealt with efficiently.

Speech and textual forms of information play a vital role in communicating among peoples. Nowadays, speech and textual conversation are primary communication forms between humans and computers that occur through web applications. The purpose of a CHATBOT is to help answer user queries [3]. CHATBOT is a computer program that processes a user’s natural-language input and generates relatively smart, affluent, and intelligent responses sent back to the user [4]. CHATBOT help with human request and allow conversation 24 hours out of every day and improve productivity by assuming control over all activities where people are not required. However, the most significant advantage of CHATBOT is that it can reach a wide-ranging audience on a messaging system and automate customized messages [5]. Some common examples of CHATBOT are ASK DISHA for IRCTC inquiry, Amazon chat customer service, HDFC bank bot EVA, etc. CHATBOT has been utilized in several industries to convey specific information or perform tasks, like telling the weather of Delhi, making flight reservations from Mumbai to Kolkata, answering educational-based queries, or purchasing

products and groceries. Telegram, WhatsApp messenger, Signal, IBM, Microsoft Cortana, Slack, Google Assistant, Siri, WeChat, Facebook Messenger [5], etc. popular famous applications are using these technologies.

**1.1 A Brief Historical Account for Chatbot**



## 

## **Conclusion**

What we’ve illustrated here is just one among the many ways of **how to make a chatbot in Python.** You can also use NLTK, another resourceful Python library to create a Python chatbot. And although what you learned here is a very basic**chatbot in Python** having hardly any cognitive skills, it should be enough to help you understand the anatomy of chatbots.