

PROFESSIONAL TRAINING REPORT

At

**Sathyabama Institute of Science and
Technology (Deemed to be University)**

Submitted in partial fulfillment of the requirements for the award of Bachelor of
Engineering Degree in INFORMATION TECHNOLOGY

By

BALAJI P
(Reg. No.38120018)



**DEPARTMENT OF INFORMATION TECHNOLOGY
AND
SCHOOL OF COMPUTING**

**SATHYABAMA INSTITUTE OF SCIENCE AND
TECHNOLOGY
JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI – 600119, TAMILNADU**

November 2021



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

(Established under Section 3 of UGC Act, 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI– 600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFAIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **BALAJI P(38120018)** who carried out the project entitled “**Chatbot using Python with nlp**” under my supervision from August 2021 to November 2021.

Internal Guide

Dr.G.MATHIVANAN M.E., Ph.D.,

Head of the Department

Dr.R.SUBHASHINI, M.E., Ph.D.

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I **BALAJI P(38120018)** hereby declare that the Project Report entitled **CHATBOT USING PYTHON WITH NLP** done by me under the guidance of **Dr.G.MATHIVANAN, M.E., Ph.D.** is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering degree in Information Technology.

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management** of **SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean**, School of Computing , **Dr.R.SUBHASHINI M.E., Ph.D.,** Heads of the Department of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr.G.MATHIVANAN M.E.,Ph.D** for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

Messaging and voice-controlled devices are the next big platforms, and conversational computing has a big role to play in creating ,engaging augmented and virtual reality experiences. So this project is aimed to developing a pybot which is nothing but a chatbot. The purpose of chat bots is to support and scale business teams in their relations with customers. Unlike humans who can only communicate with one human at a time, chat bots can simultaneously have conversations with thousands of people. No matter what time of the day it is or how many people are contacting you, every single one of them will be answered immediately. Like that Pybot is also a chatbot ,it can answer all the queries which is related to the python modules. It make a wonderful interaction with the users.

TABLE OF CONTENTS

CHAPTER No.	TITLE	PAGE No
	ABSTRACT	i
	LIST OF FIGURES	ii
	LIST OF TABLES	iii
	LIST OF ABBREVIATIONS	
1	INTRODUCTION	
	1.1 INTRODUCTION ABOUT CHATBOTS	1
	1.2 ABOUT PYTHON	1
	1.3 ABOUT NLP	2
2	AIM AND SCOPE OF THE CHATBOT	
	2.1 AIM OF THE CHATBOT	3
	2.2 SCOPE OF THE CHATBOT	4
	2.3 APPLICATIONS OF THE CHATBOT	5
3	CHATBOT DESIGNS	
	3.1 HOW CHATBOT WORKS	7
	3.2 CHATBOT SPECTRUM	8
	3.3 HOW DOES NLP IN CHATBOTS WORK?	9
4	IMPLEMENTATION OF THE CHATBOT	
	4.1 PACKAGES AND MODULES REQUIRED	14
	4.2 TKINTER PROGRAMMING	21
5	CONCLUSIONS AND FUTURE WORKS	
	REFERENCES	28
	APPENDIX	
	A.SOURCE CODE	28
	B. SCREENSHOTS	53

LIST OF FIGURES

Figure No.	TITLE	Page No.
2.1.1	Applications of chatbot	6
3.2.2	How does chatbot works	7
3.2.3	How does rule based chatbot works	10
3.2.4	How does A.I chatbot works	11
3.2.5	How does nlp in chatbot works	13

LIST OF TABLES

Table No.	TITLE	Page No.
4.1.1	Random modules methods	16
4.2.2	Tkinter widgets	23

LIST OF ABBREVIATIONS

- **AI - Artificial intelligence**
- **NLP- Natural language processing**
- **OCR- Optical character recognition**
- **TTS- Text to speech**
- **NLU- Natural Language Understanding**
- **NLG- Natural Language Generation**
- **NLTK- Natural Language Toolkit**
- **GUI- Graphical user Interface**
- **TK-Tkinter**

CHAPTER - 1

INTRODUCTION

1.1 WHAT IS CHATBOT?

A chatbot is an artificial intelligence (AI) software that can simulate a conversation (or a chat) with a user in natural language through messaging applications, websites, mobile apps or through the telephone.

A chatbot is often described as one of the most advanced and promising expressions of interaction between humans and machines. However, from a technological point of view, a chatbot only represents the natural evolution of a Question Answering system leveraging Natural Language Processing (NLP). Formulating responses to questions in natural language is one of the most typical Examples of Natural Language Processing applied in various enterprises' end-use applications

1.2 ABOUT PYTHON:

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built in data structures, combined with dynamic typing and dynamic binding, make it very attractive for rapid application development, as well as for use as a scripting or glue language to connect existing components together. Python's simple, easy to learn syntax emphasizes readability and therefore reduces the cost of program maintenance. Python supports modules and packages, which encourages program modularity and code reuse. The python interpreter and the extensive standard library are available in source or binary form without charge for all major platforms, and can be freely distributed.

Often, programmers fall in love with python because of the increased productivity it provides. Since there is no compilation step, the edit-test-debug cycle is incredibly fast. Debugging python programs is easy: A bug or badinput will never cause a segmentation fault. Instead, when the interpreter discovers an error, it raises an exception. When the program doesn't catch the exception, the interpreter prints a

stack trace. A source level debugger allows inspection of local and global variables, evaluation of arbitrary expressions, setting breakpoints, stepping through the code a line at a time, and so on. The debugger is written in python itself, testifying to python's introspective power. On the other hand, often the quickest way to debug a program is to add a few print statements to the source: The fast edit-test-debug cycle makes this simple approach very effective.

Python is commonly used in artificial intelligence projects and machine learning projects with the help of libraries like TensorFlow, Keras, Pytorch and Scikit-learn. As a scripting language with modular architecture, simple syntax and rich text processing tools, Python is often used for natural language processing.

1.3 ABOUT NLP:

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. Challenges in natural language processing frequently involve speech recognition, natural language understanding, and natural-language generation.

1.3.1 COMMON NLP TASK

The following is a list of some of the most commonly researched tasks in natural language processing. Some of these tasks have direct real-world applications, while others more commonly serve as subtasks that are used to aid in solving larger tasks. Though natural language processing tasks are closely intertwined, they can be subdivided into categories for convenience. A coarse division is given below.

1.3.2 TEXT AND SPEECH PROCESSING

✧ **Optical character recognition (OCR)**

Given an image representing printed text, determine the corresponding text.

✧ **Speech recognition**

Given a sound clip of a person or people speaking, determine the textual representation of the speech. This is the opposite of [text to speech](#) and is one of the extremely difficult problems colloquially termed "[AI-complete](#)". In [natural speech](#) there are hardly any pauses between successive words, and thus [speech segmentation](#) is a necessary subtask of speech recognition. In most spoken languages, the sounds representing successive letters blend into each other in a process termed [coarticulation](#), so the conversion of the [analog signal](#) to discrete characters can be a very difficult process. Also, given that words in the same language are spoken by people with different accents, the speech recognition software must be able to recognize the wide variety of input as being identical to each other in terms of its textual equivalent.

✧ **Speech segmentation**

Given a sound clip of a person or people speaking, separate it into words. A subtask of [speech recognition](#) and typically grouped with it.

✧ **Text-to-speech**

Given a text, transform those units and produce a spoken representation.

Text-to-speech can be used to aid the visually impaired.^[17]

✧ **Word segmentation (Tokenization)**

Separate a chunk of continuous text into separate words. For a language like [English](#), this is fairly trivial, since words are usually separated by spaces. However, some written languages like [Chinese](#), [Japanese](#) and [Thai](#) do not mark word boundaries in such a fashion, and in those languages text segmentation is a significant task requiring knowledge of the [vocabulary](#) and [morphology](#) of words in the language. Sometimes this process is also used in cases like [bag of words](#) (BOW) creation in data mining.

CHAPTER 2

AIM AND SCOPE OF THE CHATBOT

2.1 AIM OF THE CHATBOT:

A [chatbot](#) can communicate with a real person behaving like a human. The aim of my chatbot is act as an **education consultant**. It can answer all the **queries related to the python**. As a student I am developing my chatbot related to the education guideness.

You can create chatbots for any business the same as you recruit a person for any department of your company. Whether you are a:

- ✓ Wedding Planner
- ✓ Insurance Assistant
- ✓ Education Consultant
- ✓ Legal Assistant
- ✓ A real estate business
- ✓ Recruiter
- ✓ Travel Agency
- ✓ Hospital or a Beautician! And so on...

2.2 SCOPE OF THE CHATBOT:

Chatbots are everywhere. From online assistants, such as [Microsoft's Cortana](#), to “helper bots” on [messaging applications like Slack](#), to home applications like Amazon.com's Alexa, chatbots have become one of the most visible – and flawed – consumer-facing applications of artificial intelligence and machine learning.

● Using chatbots for specialized tasks

There are several tasks which chatbots can do quickly and more efficiently than their human counterparts. Tasks such as checking the weather, ordering a pizza or hiring cabs can be done more efficiently with chatbots. In a similar vein, businesses can use

[chatbots](#) to automate tasks such as inventory ordering and management.

- **Better service on smartphones and tablets**

The world is moving towards a mobile future. Today, mobile phone users far outnumber the laptop and PC users. Use of chatbots in mobile apps has helped in creating more streamlined app interfaces. The user can browse, compare, buy and get support from a single interface.

- **Enhance your social presence by resolving social issues**

The best example of this the chatbot DoNotPay which saved not only different users from fines amounting to \$160,000 but also helped homeless people apply for housing. It is now turning to help the refugees claim asylum. Besides DoNotPay, there are other bots like U-Report, GYANT, Yeshi, etc., are making sure that they bring about a change in the society.

Chatbots are only going to get smarter with time, allowing diversified usage of this technology. They can become the first point of contact between customers, providing instant assistance, buying advice, and resolving common issues. It is safe to say that chatbot technology is the next big thing in customer service and e-commerce industry.

2.3 APPLICATIONS OF THE CHATBOT:

There is a multitude of [applications](#) of chatbots; from receptionists to salespersons and online assistant they can do almost anything that can help you in satisfying your customers while generating more revenue.

1. Chatbots can enhance sales

It is seen that the customer experience gets even better when they have an assisting agent with them. Your chatbot can assist the clients in shopping while resolving their queries on the go.

Also, the chances of any purchase are higher when the customer is getting better

services throughout the shopping journey.

2. Chatbots can manage accounts

Managing accounts requires one to be precise about the data and the information. And you can use a chatbot to manage accounts and retrieve details whenever needed.

These [artificially intelligent bots](#) are more precise than humans, and there is a small scope of mistakes when you have the right chatbots deployed.

3. Chatbots are customer executives

Most of the websites render online chat option due to their ease and convenience.

Chatbots are perfect for deploying as [customer executives](#) because they are available round the clock and can resolve customer issues seamlessly.

Chatbots have more efficiency in dealing with clients when compared to human customer executives.

The advantages of chatbots are never-ending. They are replacing humans in many fields and are proving them as a better option.

Now you can place food orders, complain about the quality of food, and can claim for a refund without any human intervention as a chatbot will resolve all the queries with its smart sense.

Use cases within various sectors

APPLICATION FIELDS

PIONEERS
DISCOVER



Fig.no.2.3.1 Applications of chatbot

CHAPTER 3

CHATBOT DESIGNS

3.1 HOW CHATBOT WORKS?

There are two different tasks at the core of a chatbot:

- 1) User request analysis
- 2) Returning the response

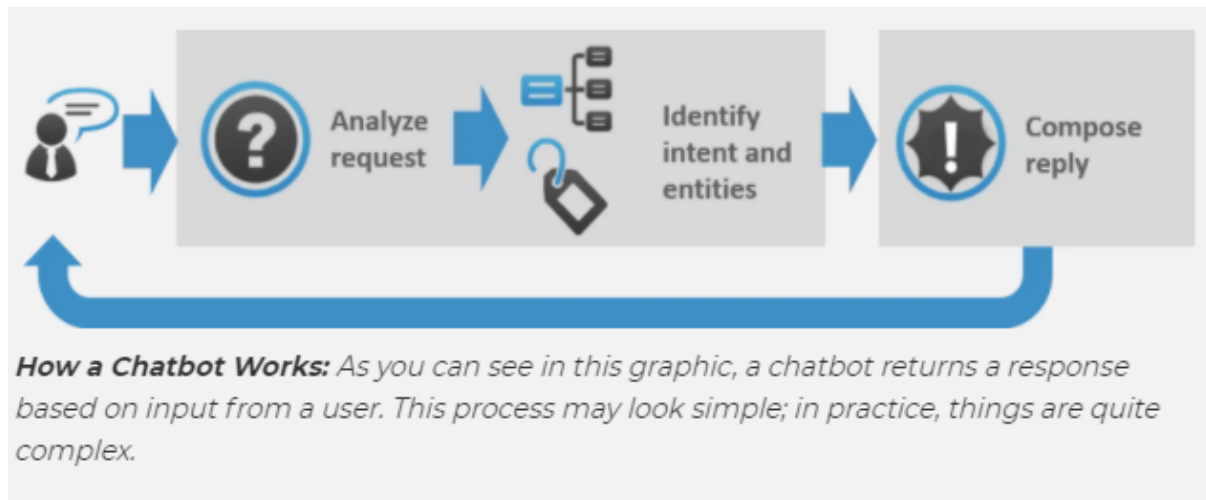


Fig.no.3.1.2 How a chatbot works

1.User request analysis:

If you are not able to correctly understand the user's request, you won't be able to provide the correct answer. This is the first task that a chatbot performs. It analyzes the user's request to identify the user intent and to extract relevant entities.

The ability to identify the user's intent and extract data and relevant entities contained in the user's request is the first condition and the most relevant step at the core of a chatbot

2.Returning the response:

once the user's intent has been identified, the chatbot must provide the most

appropriate response for the user's request. The answer may be:

- a generic and predefined text
- a text retrieved from a knowledge base that contains different answers
- a contextualized piece of information based on data the user has provided
- data stored in enterprise systems
- the result of an action that the chatbot performed by interacting with one or more backend application
- a disambiguating question that helps the chatbot to correctly understand the user's request

3.2 CHATBOT SPECTRUM: TYPES AND USE CASES

Type 1: Scripted/Quick Reply Bots

A scripted chatbot, as the name suggests, is one whose interaction with the end customer happens through a pre-defined knowledge base and technical capabilities that can aptly respond only to specific instructions. In that case, the queries must be aligned with the language programmed in the bot.

Type 2: NLP Chatbots

Chatbots, in general, are counted as one of the biggest applications of Artificial Intelligence (AI). At its core, it utilizes an AI technology- Natural Language Processing (NLP) to map user input (text or voice) to an intent. NLP chatbots classify the messages and then language parsing is done to get variables for an answer.

Type 3: Service/Action Chatbots

Service chatbots ask for relevant info from the user to complete their request or take a specific action. This chatbot is heavily used in the airline industry. It helps customers check flight booking, cost of reserving the flights and check the statuses as well.

Type 4: Social Messaging Chatbots

These chatbots are integrated within a social messaging platform (Messenger, Telegram, Whatsapp, Slack etc.), making it easy for customers to directly interact with the bot, just like they do with their friends.

Type 5: Context Enabled Chatbots

Contextual chatbots are the most advanced kind of conversational bots. They utilize Machine Learning and Artificial Intelligence to remember conversations that happened in the past, with specific users, to learn and grow over time. These chatbots learn with their experiences with the user. Siri, Alexa, Google Assistant are some of the examples of contextual chatbots.

Type 6: Voice Enabled Chatbots

Voice enabled chatbots like Alexa, Siri creates personalized experience for the users. These chatbots accept user inputs through voice, act upon user request, answer their queries, and perform a number of creative tasks. Businesses can create their own voice-activated chatbot by using text-to-speech (TTS) and voice recognition APIs.

The three main types of chatbots:

1.Rule-based chatbots

This is the simplest type of chatbots today. People interact with these bots by clicking on buttons and using pre-defined options. To give relevant answers these chatbots require people to make a few selections. As a result, these bots have longer user journey, and they are the slowest to guide the customer to their goal.

These bots are great when it comes to qualifying your leads. The chatbot – asks questions, and people answer them with buttons. The bot analyzes collected data and gives a reply. But, for more advanced scenarios with many conditions or factors, these

chatbots aren't always the best solution.

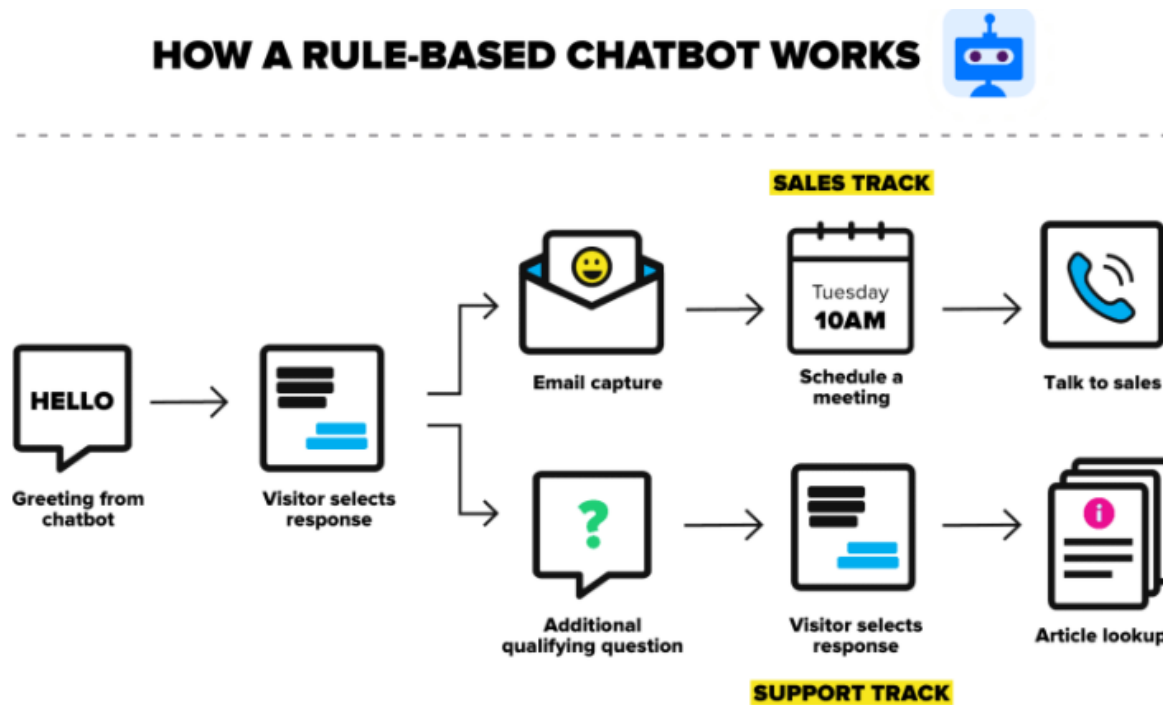


Fig. No. 3.2.3 How a rule based chatbot works

2. Intellectually independent chatbots

These bots use Machine Learning (ML) which helps the chatbot learn from user's inputs and requests.

ML is the ability of the computer to learn by itself from the data, recognize patterns and decide with minimal human interference.

Intellectually independent chatbots are trained to understand specific keywords and phrases that trigger bot's reply. With the time they train themselves to understand more and more questions. You can say they learn and train from experience.

For example, you write to a chatbot: "I have a problem with logging into my account". The bot would understand the words "problem" "logging" "account" and would provide a pre-defined answer based on these phrases.

3.AI- powered chatbots

AI-powered bots combine the best from Rule-based and Intellectually independent.

Artificial Intelligence (AI) is a simulation of human intelligence. AI is the area of computer science that focuses on creating intelligent machines that work and "think" as people.

AI-powered chatbots understand free language, but also have a predefined flow to make sure they solve user's problem. They can remember the context of the conversation and the user's preferences. These chatbots can jump from one point of conversation scenario to another when needed and address random user request at any moment.

These chatbots use Machine Learning, AI and Natural Language Processing (NLP) to understand people.

HOW AN A.I. CHATBOT WORKS

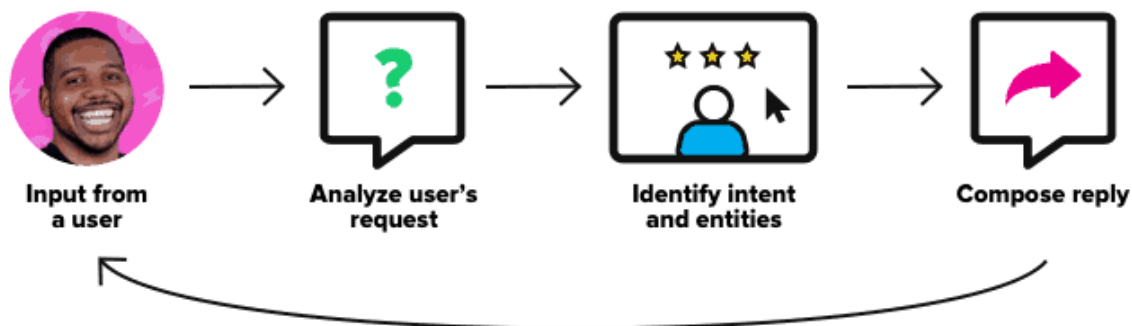


Fig no 3.2.4 How an A.I chatbot works

***NLP** is the ability of the computer to understand and analyze human speech, find the right response and reply in understandable for a human language.*

The goal of NLP is to make the interaction between computers and humans feel like communication between two people. With the help of NLP people can freely interact with chatbots asking a question.

3.3 HOW DOES NLP IN CHATBOTS WORK?

NLP involves 2 processes: Natural Language Understanding (NLU) and Natural Language Generation (NLG).

***NLU** is the ability of the chatbot to understand a human. It is the process of converting text into structured data for a machine to understand.*

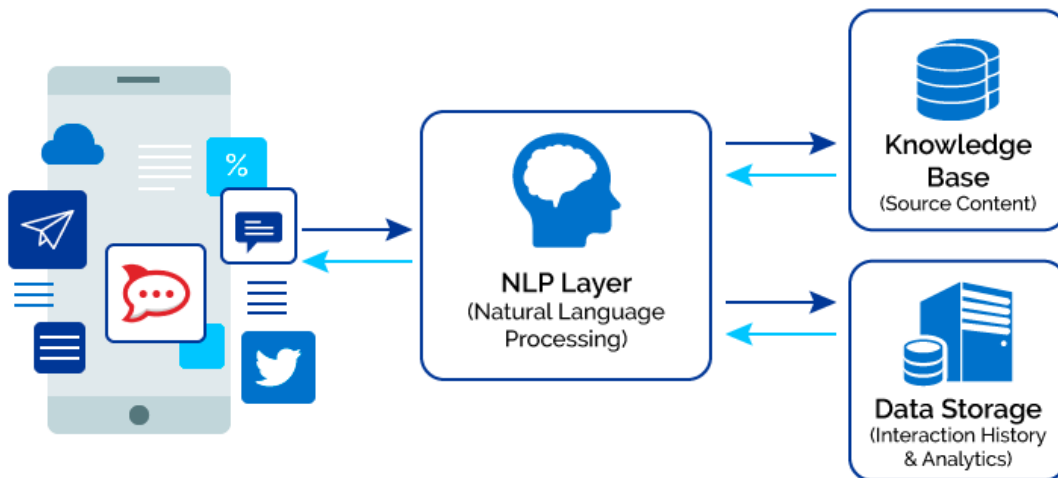
***NLG** transforms structured data into text.*

Imagine, a user writes to a chatbot “*What is the weather in San Francisco today?*”. To give a reply chatbot breaks down the sentence into Intents and Entities.

***An intent** is an action or a request the user wants to perform or information he wants to get.*

***An entity** is a detail that compliments the Intent. It can be a location, date, color, size, flavor, etc.*

So, using the same example the Intent here would be “*Weather*”. The entities would be “*San Francisco*” and “*today*”.



(NLP) Natural Language Processing Chatbots finds a way to convert the user's speech or text into structured data. Which is then utilized to choose a relevant answer. Natural Language Processing includes the following steps;

Fig.no 3.2.5 How does nlp in chatbot works

Few main steps in NLP mechanism:

(NLP) Natural Language Processing Chatbots finds a way to convert the user's speech or text into structured data. Which is then utilized to choose a relevant answer. Natural Language Processing includes the following steps;

- 1 **Tokenization:** The NLP separates a series of words into tokens or pieces that are linguistically representative, with a different value in the application.
- 2 **Sentiment Analysis:** It will study and learn the user's experience, and transfer the inquiry to a human when necessary
- 3 **Normalization:** This program model processes the text to find out the typographical errors and common spelling mistakes that might alter the intended meaning of the user's request.
- 4 **Named Entity Recognition:** The program model of chatbot looks for different categories of words, similar to the name of the particular product, the user's address or name, whichever information is required.
- 5 **Dependency Parsing:** The Chatbot searches for the subjects, verbs, objects,

common phrases and nouns in the user's text to discover related phrases that what users want to convey.

CHAPTER 4

IMPLEMENTATION OF THE CHATBOT

4.1 PACKAGES AND MODULES REQUIRED:

◆ Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to [over 50 corpora and lexical resources](#) such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries, and an active [discussion forum](#).

Library highlights[[edit](#)]

- ❖ [Lexical analysis](#): Word and text tokenizer
- ❖ [n-gram](#) and collocations
- ❖ [Part-of-speech tagger](#)
- ❖ Tree model and Text [chunker](#) for capturing
- ❖ [Named-entity recognition](#)

◆ Import warning

Warning messages are typically issued in situations where it is useful to alert the user of some condition in a program, where that condition (normally) doesn't warrant

raising an exception and terminating the program. For example, one might want to issue a warning when a program uses an obsolete module.

Python programmers issue warnings by calling the [warn\(\)](#) function defined in this module.

◆ Import numpy

NumPy is a library for the [Python programming language](#), adding support for large, multi-dimensional [arrays](#) and [matrices](#), along with a large collection of [high-level mathematical functions](#) to operate on these arrays.

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.

◆ Import random

Python has a built-in module that you can use to make random numbers.

The [random](#) module has a set of methods:

Method	Description
seed()	Initialize the random number generator
getstate()	Returns the current internal state of the random number generator
setstate()	Restores the internal state of the random number generator

<u>getrandbits()</u>	Returns a number representing the random bits
<u>randrange()</u>	Returns a random number between the given range
<u>randint()</u>	Returns a random number between the given range
<u>choice()</u>	Returns a random element from the given sequence
<u>choices()</u>	Returns a list with a random selection from the given sequence
<u>shuffle()</u>	Takes a sequence and returns the sequence in a random order
<u>sample()</u>	Returns a given sample of a sequence
<u>random()</u>	Returns a random float number between 0 and 1
<u>uniform()</u>	Returns a random float number between two given parameters
<u>triangular()</u>	Returns a random float number between two given parameters, you can also set a mode parameter to specify the midpoint between the two other parameters
betavariate()	Returns a random float number between 0 and 1 based on the Beta distribution (used in statistics)
expovariate()	Returns a random float number based on the Exponential distribution (used in statistics)
gammavariate() te()	Returns a random float number based on the Gamma distribution (used in statistics)
gauss()	Returns a random float number based on the Gaussian distribution (used in probability theories)
lognormvariate() ate()	Returns a random float number based on a log-normal distribution (used in probability theories)

normalvariate()	Returns a random float number based on the normal distribution (used in probability theories)
vonmisesvariate()	Returns a random float number based on the von Mises distribution (used in directional statistics)
paretovariate()	Returns a random float number based on the Pareto distribution (used in probability theories)
weibullvariate()	Returns a random float number based on the Weibull distribution (used in statistics)

Table no.4.1.1 Random module methods

◆ Import string

Python String module contains some constants, utility function, and classes for string manipulation.

The **string** module contains a number of useful constants and classes, as well as some deprecated legacy functions that are also available as methods on strings. In addition, Python's built-in string classes support the sequence type methods described in the [Sequence Types — str, bytes, bytearray, list, tuple, range](#) section, and also the string-specific methods described in the [String Methods](#) section. To output formatted strings, see the [String Formatting](#) section. Also, see the **re** module for string functions based on regular expressions.

◆ Punkt Sentence Tokenizer

This tokenizer divides a text into a list of sentences, by using an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences. It must be trained on a large collection of plaintext in the target

language before it can be used.

Punkt is designed to learn parameters (a list of abbreviations, etc.) unsupervised from a corpus similar to the target domain. The pre-packaged models may therefore be unsuitable: use `PunktSentenceTokenizer(text)` to learn parameters from the given text.

[nltk.tokenize.punkt.PunktTrainer](#) learns parameters such as a list of abbreviations (without supervision) from portions of text. Using a `PunktTrainer` directly allows for incremental training and modification of the hyper-parameters used to decide what is considered an abbreviation, etc.

◆ wordnet

[WordNet](#) is a lexical database for the English language, which was created by Princeton, and is part of the NLTK corpus. `WordNet` is just another NLTK corpus reader, and can be imported like this:

```
>>> from nltk.corpus import wordnet
```

For more compact code, we recommend:

```
>>> from nltk.corpus import wordnet as wn
```

◆ Python | Lemmatization with NLTK

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is similar to stemming but it brings context to the words. So it links words with similar meaning to one word.

Text preprocessing includes both [Stemming](#) as well as Lemmatization. Many times people find these two terms confusing. Some treat these two as same. Actually, lemmatization is preferred over Stemming because lemmatization does morphological analysis of the words.

Applications of lemmatization are:

- a) Used in comprehensive retrieval systems like search engines.
- b) Used in compact indexing

◆ `sklearn.feature_extraction.text.TfidfVectorizer`

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to `CountVectorizer` followed by `TfidfTransformer`.

◆ Cosine similarity

`cosine_similarity` computes the L2-normalized dot product of vectors. That is, if x and y are row vectors, their cosine similarity k is defined as:

$$k(x,y) = \frac{xy^T}{\|x\| \|y\|}$$

This is called cosine similarity, because Euclidean (L2) normalization projects the vectors onto the unit sphere, and their dot product is then the cosine of the angle between the points denoted by the vectors.

This kernel is a popular choice for computing the similarity of documents represented as tf-idf vectors. `cosine_similarity` accepts `scipy.sparse` matrices. (Note that the tf-idf functionality in `sklearn.feature_extraction.text` can produce normalized vectors, in which case `cosine_similarity` is equivalent to `linear_kernel`, only slower.)

◆ Import time

Python has a module named `time` to handle time-related tasks. To use functions defined in the module, we need to import the module first. Here's how:

```
import time
```

Python time.sleep()

The sleep() function suspends (delays) execution of the current thread for the given number of seconds.

Import threading

Threading in python is used to run multiple threads (tasks, function calls) at the same time. Python threads are used in cases where the execution of a task involves some waiting. One example would be interaction with a service hosted on another computer, such as a webserver. Threading allows python to execute other code while waiting; this is easily simulated with the sleep function.

In Python, the threading module is a built-in module which is known as threading and can be directly imported.

Since almost everything in Python is represented as an object, threading also is an object in Python. A thread is capable of

- Holding data,
- Stored in data structures like dictionaries, lists, sets, etc.

Can be passed as a parameter to a function.

A thread can also be executed as a process.

A thread in Python can have various states like:

- Wait,
- Locked.

◆ |Text to Speech by using pyttsx3

pyttsx3 is a text-to-speech conversion library in Python. Unlike alternative libraries, it works offline and is compatible with both Python 2 and 3. An application invokes the pyttsx3.init() factory function to get a reference to a pyttsx3. Engine

instance. it is a very easy to use tool which converts the entered text into speech. The pyttsx3 module supports two voices first is female and the second is male which is provided by “sapi5” for windows.

It supports three TTS engines :

- A. *sapi5* – SAPI5 on Windows
- B. *nsss* – NSSpeechSynthesizer on Mac OS X
- C. *espeak* – eSpeak on every other platform

Installataion

To install the pyttsx3 module, first of all, you have to open the terminal and write

```
pip install pyttsx3
```

4.2 TKINTER PROGRAMMING

Tkinter is the standard GUI library for Python. Python when combined with Tkinter provides a fast and easy way to create GUI applications. Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit.

Creating a GUI application using Tkinter is an easy task. All you need to do is perform the following steps –

- i. Import the *Tkinter* module.
- ii. Create the GUI application main window.
- iii. Add one or more of the above-mentioned widgets to the GUI application.
- iv. Enter the main event loop to take action against each event triggered by the user.

Example

```
#!/usr/bin/python
```

```
import Tkinter
```

```
top = Tkinter.Tk()

# Code to add widgets will go here...

top.mainloop()
```

Tkinter Modules

Most of the time, [tkinter](#) is all you really need, but a number of additional modules are available as well. The Tk interface is located in a binary module named `_tkinter`. This module contains the low-level interface to Tk, and should never be used directly by application programmers. It is usually a shared library (or DLL), but might in some cases be statically linked with the Python interpreter.

In addition to the Tk interface module, [tkinter](#) includes a number of Python modules, `tkinter.constants` being one of the most important. Importing [tkinter](#) will automatically import `tkinter.constants`, so, usually, to use Tkinter all you need is a simple import statement:

```
import tkinter
```

Or, more often:

```
from tkinter import *
```

Other modules that provide Tk support include:

[tkinter.scrolledtext](#)

Text widget with a vertical scroll bar built in.

`tkinter.colorchooser`

Dialog to let the user choose a color.

`tkinter.commondialog`

Base class for the dialogs defined in the other modules listed here.

`tkinter.filedialog`

Common dialogs to allow the user to specify a file to open or save.

`tkinter.font`

Utilities to help work with fonts.

`tkinter.messagebox`

Access to standard Tk dialog boxes.

`tkinter.simpledialog`

Basic dialogs and convenience functions.

`tkinter.dnd`

Drag-and-drop support for [tkinter](#). This is experimental and should become deprecated when it is replaced with the Tk DND.

This would create a following window –

Tkinter Widgets

Tkinter provides various controls, such as buttons, labels and text boxes used in a GUI application. These controls are commonly called widgets.

There are currently 15 types of widgets in Tkinter. We present these widgets as well as a brief description in the following table –

Sr.No.	Operator & Description
--------	------------------------

1	Button The Button widget is used to display buttons in your application.
2	Canvas The Canvas widget is used to draw shapes, such as lines, ovals, polygons and rectangles, in your application.
3	Checkbutton The Checkbutton widget is used to display a number of options as checkboxes. The user can select multiple options at a time.
4	Entry The Entry widget is used to display a single-line text field for accepting values from a user.
5	Frame The Frame widget is used as a container widget to organize other widgets.
6	Label The Label widget is used to provide a single-line caption for other widgets. It can also contain images.
7	Listbox The Listbox widget is used to provide a list of options to a user.
8	Menubutton The Menubutton widget is used to display menus in your application.
9	Menu

	The Menu widget is used to provide various commands to a user. These commands are contained inside Menubutton.
10	Message The Message widget is used to display multiline text fields for accepting values from a user.
11	Radiobutton The Radiobutton widget is used to display a number of options as radio buttons. The user can select only one option at a time.
12	Scale The Scale widget is used to provide a slider widget.
13	Scrollbar The Scrollbar widget is used to add scrolling capability to various widgets, such as list boxes.
14	Text The Text widget is used to display text in multiple lines.
15	Toplevel The Toplevel widget is used to provide a separate window container.
16	Spinbox The Spinbox widget is a variant of the standard Tkinter Entry widget, which can be used to select from a fixed number of values.
17	PanedWindow

	A PanedWindow is a container widget that may contain any number of panes, arranged horizontally or vertically.
18	LabelFrame A labelframe is a simple container widget. Its primary purpose is to act as a spacer or container for complex window layouts.
19	tkMessageBox This module is used to display message boxes in your applications.

4.2.2 tkinter widgets

Standard attributes

Let us take a look at how some of their common attributes, such as sizes, colors and fonts are specified.

- Dimensions
- Colors
- Fonts
- Anchors
- Relief styles
- Bitmaps
- Cursors

Geometry Management

All Tkinter widgets have access to specific geometry management methods, which have the purpose of organizing widgets throughout the parent widget area. Tkinter exposes the following geometry manager classes: pack, grid, and place.

The *pack()* Method – This geometry manager organizes widgets in blocks before placing them in the parent widget.

The *grid()* Method – This geometry manager organizes widgets in a table-like structure in the parent widget.

The *place()* Method – This geometry manager organizes widgets by placing them in a specific position in the parent widget.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

From my perspective, chatbots or smart assistants with artificial intelligence are dramatically changing businesses. There is a wide range of chatbot building platforms that are available for various enterprises, such as e-commerce, retail, banking, leisure, travel, healthcare, and so on.

Chatbots can reach out to a large audience on messaging apps and be more effective than humans. They may develop into a capable information-gathering tool in the near future.

5.2 FUTURE WORK

According to my project, I am planning to do a chatbot and get uploaded to medical

institute. It may be useful for the medical institute those who have queries about medicines. I hope it will work greatly.

REFERENCES

- ✓ <https://www.drift.com/learn/chatbot/why-are-chatbots-important>
- ✓ <https://botscrew.com/blog/what-are-bots/>
- ✓ <https://en.wikipedia.org/wiki/Chatbot>
- ✓ <https://developer.kore.ai/docs/bots/chatbot-overview/chatbot-overview/>
- ✓ <https://www.nltk.org/>
- ✓ <https://www.geeksforgeeks.org/python-gui-tkinter/>
- ✓ <https://medium.com/@randerson112358/build-a-simple-chat-bot-graphical-user-interface>
- ✓ <https://www.youtube.com/watch?v=Z9V2gxc3Z5Q>

APPENDIX

A. code

B. Implemented screenshots

A.Code

```
import nltk
import warnings
warnings.filterwarnings("ignore")
# nltk.download() # for downloading packages
#import tensorflow as tf
import numpy as np
import random
import string # to process standard python strings

f=open('nlp python answer finals.txt','r',errors = 'ignore')
m=open('modules pythons.txt','r',errors = 'ignore')
checkpoint = "./chatbot_weights.ckpt"
#session = tf.InteractiveSession()
#session.run(tf.global_variables_initializer())
#saver = tf.train.Saver()
#saver.restore(session, checkpoint)

raw=f.read()
rawone=m.read()
raw=raw.lower()# converts to lowercase
rawone=rawone.lower()# converts to lowercase
nltk.download('punkt') # first-time use only
```

```

nltk.download('wordnet') # first-time use only
sent_tokens = nltk.sent_tokenize(raw)# converts to list of sentences
word_tokens = nltk.word_tokenize(raw)# converts to list of words
sent_tokensone = nltk.sent_tokenize(rawone)# converts to list of sentences
word_tokensone = nltk.word_tokenize(rawone)# converts to list of words


sent_tokens[:2]
sent_tokensone[:2]


word_tokens[:5]
word_tokensone[:5]


lemmer = nltk.stem.WordNetLemmatizer()
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))


Introduce_Ans = ["My name is PyBot.", "My name is PyBot you can called me pi.", "Im
PyBot :) ", "My name is PyBot. and my nickname is pi and i am happy to solve your
queries :) "]
GREETING_INPUTS = ("hello", "hi", "hiii", "hii", "hiiii", "hiiii", "greetings", "sup", "what's
up", "hey",)
GREETING_RESPONSES = ["hi", "hey", "hii there", "hi there", "hello", "I am glad! You
are talking to me"]
Basic_Q = ("what is python ?", "what is python", "what is python?", "what is python.")
Basic_Ans = "Python is a high-level, interpreted, interactive and object-oriented
scripting programming language python is designed to be highly readable It uses
English keywords frequently where as other languages use punctuation, and it has
fewer syntactical constructions than other languages."

```

```
Basic_Om = ("what is module","what is module.", "what is module ", "what is module  
?", "what is module?", "what is module in python", "what is module in python.", "what is  
module in python?", "what is module in python ?")
```

```
Basic_AnsM = ["Consider a module to be the same as a code library.", "A file  
containing a set of functions you want to include in your application.", "A module can  
define functions, classes and variables. A module can also include runnable code.  
Grouping related code into a module makes the code easier to understand and use."]
```

```
# 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)
```

```
# Checking for Basic_Q
```

```
def basic(sentence):
```

```
    for word in Basic_Q:
```

```
        if sentence.lower() == word:
```

```
            return Basic_Ans
```

```
# Checking for Basic_QM
```

```
def basicM(sentence):
```

```
    """If user's input is a greeting, return a greeting response"""
```

```
    for word in Basic_Om:
```

```
        if sentence.lower() == word:
```

```
            return random.choice(Basic_AnsM)
```

```
# Checking for Introduce
```

```
def IntroduceMe(sentence):
```

```
    return random.choice(Introduce_Ans)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity
```

```
# Generating response
```

```
def response(user_response):
```

```
    robo_response=""
```

```
    sent_tokens.append(user_response)
```

```
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
```

```
    tfidf = TfidfVec.fit_transform(sent_tokens)
```

```
    vals = cosine_similarity(tfidf[-1], tfidf)
```

```
    idx=vals.argsort()[0][-2]
```

```
    flat = vals.flatten()
```

```
    flat.sort()
```

```
    req_tfidf = flat[-2]
```

```
    if(req_tfidf==0):
```

```
        robo_response=robo_response+"I am sorry! I don't understand you"
```

```
        return robo_response
```

```
    else:
```

```
        robo_response = robo_response+sent_tokens[idx]
```

```
        return robo_response
```

```
# Generating response
```

```
def responseone(user_response):
```

```
    robo_response=""
```

```
    sent_tokensone.append(user_response)
```

```
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize, stop_words='english')
```

```
    tfidf = TfidfVec.fit_transform(sent_tokensone)
```

```
    vals = cosine_similarity(tfidf[-1], tfidf)
```

```
    idx=vals.argsort()[0][-2]
```

```

flat = vals.flatten()
flat.sort()
req_tfidf = flat[-2]
if(req_tfidf==0):
    robo_response=robo_response+"I am sorry! I don't understand you"
    return robo_response
else:
    robo_response = robo_response+sent_tokenone[idx]
    return robo_response

def chat(user_response):
    user_response=user_response.lower()
    keyword = " module "
    keywordone = " module"
    keywordsecond = "module "

    if(user_response!='bye'):
        if(user_response=='thanks' or user_response=='thank you' ):
            flag=False
            #print("ROBO: You are welcome..")
            return "You are welcome.."
        elif(basicM(user_response)!=None):
            return basicM(user_response)
        else:
            if(user_response.find(keyword) != -1 or user_response.find(keywordone)
            != -1 or user_response.find(keywordsecond) != -1):
                #print("ROBO: ",end="")
                #print(responseone(user_response))
                return responseone(user_response)
                sent_tokenone.remove(user_response)
            elif(greeting(user_response)!=None):

```

```

        #print("ROBO: "+greeting(user_response))
        return greeting(user_response)
    elif(user_response.find("your name") != -1 or user_response.find(" your
name") != -1 or user_response.find("your name ") != -1 or user_response.find(" your
name ") != -1):
        return IntroduceMe(user_response)
    elif(basic(user_response)!=None):
        return basic(user_response)
    else:
        #print("ROBO: ",end="")
        #print(response(user_response))
        return response(user_response)
        sent_tokens.remove(user_response)

else:
    flag=False
    #print("ROBO: Bye! take care..")
    return "Bye! take care.."

```

*******gui*******

```

from tkinter import *
import time
import tkinter.messagebox
from bot import chat
import pyttsx3
import threading

```

```

saved_username = ["You"]
#ans=["PyBot"]

```

```
window_size="400x400"
```

```
class ChatInterface(Frame):
```

```
    def __init__(self, master=None):
```

```
        Frame.__init__(self, master)
```

```
        self.master = master
```

```
        # sets default bg for top level windows
```

```
        self.tl_bg = "#EEEEEE"##hexadecimal number code
```

```
        self.tl_bg2 = "#EEEEEE"
```

```
        self.tl_fg = "#000000"
```

```
        self.font = "Verdana 10"
```

```
        menu = Menu(self.master)
```

```
        self.master.config(menu=menu, bd=5)
```

```
# Menu bar
```

```
# File
```

```
    file = Menu(menu, tearoff=0)
```

```
    menu.add_cascade(label="File", menu=file)
```

```
# file.add_command(label="Save Chat Log", command=self.save_chat)
```

```
    file.add_command(label="Clear Chat", command=self.clear_chat)
```

```
# file.add_separator()
```

```
    file.add_command(label="Exit",command=self.chatexit)
```

```
# Options
```

```
    options = Menu(menu, tearoff=0)
```

```
    menu.add_cascade(label="Options", menu=options)
```

```
# username
```

```

# font
font = Menu(options, tearoff=0)
options.add_cascade(label="Font", menu=font)
font.add_command(label="Default",command=self.font_change_default)
font.add_command(label="Times",command=self.font_change_times)
font.add_command(label="System",command=self.font_change_system)
font.add_command(label="Helvetica",command=self.font_change_helvetica)
font.add_command(label="Fixedsys",command=self.font_change_fixedsys)

# color theme
color_theme = Menu(options, tearoff=0)
options.add_cascade(label="Color Theme", menu=color_theme)

color_theme.add_command(label="Default",command=self.color_theme_default)
# color_theme.add_command(label="Night",command=self.)
color_theme.add_command(label="Grey",command=self.color_theme_grey)

color_theme.add_command(label="Blue",command=self.color_theme_dark_blue)

color_theme.add_command(label="Torque",command=self.color_theme_turquoise)

color_theme.add_command(label="Hacker",command=self.color_theme_hacker)
# color_theme.add_command(label='Mkbhd',command=self.MKBHD)

help_option = Menu(menu, tearoff=0)
menu.add_cascade(label="Help", menu=help_option)

```

```

#help_option.add_command(label="Features", command=self.features_msg)
help_option.add_command(label="About PyBot", command=self.msg)
help_option.add_command(label="Developer", command=self.about)

self.text_frame = Frame(self.master, bd=6)
self.text_frame.pack(expand=True, fill=BOTH)

# scrollbar for text box
self.text_box_scrollbar = Scrollbar(self.text_frame, bd=0)
self.text_box_scrollbar.pack(fill=Y, side=RIGHT)

# contains messages
self.text_box = Text(self.text_frame,
yscrollcommand=self.text_box_scrollbar.set, state=DISABLED,
                    bd=1, padx=6, pady=6, spacing3=8, wrap=WORD,
bg=None, font="Verdana 10", relief=GROOVE,
                    width=10, height=1)
self.text_box.pack(expand=True, fill=BOTH)
self.text_box_scrollbar.config(command=self.text_box.yview)

# frame containing user entry field
self.entry_frame = Frame(self.master, bd=1)
self.entry_frame.pack(side=LEFT, fill=BOTH, expand=True)

# entry field
self.entry_field = Entry(self.entry_frame, bd=1, justify=LEFT)
self.entry_field.pack(fill=X, padx=6, pady=6, ipady=3)
# self.users_message = self.entry_field.get()

# frame containing send button and emoji button
self.send_button_frame = Frame(self.master, bd=0)
self.send_button_frame.pack(fill=BOTH)

```

```

# send button
self.send_button = Button(self.send_button_frame, text="Send", width=5,
relief=GROOVE, bg='white',
                           bd=1, command=lambda:
self.send_message_insert(None), activebackground="#FFFFFF",
                           activeforeground="#000000")
self.send_button.pack(side=LEFT, ipady=8)
self.master.bind("<Return>", self.send_message_insert)

self.last_sent_label(date="No messages sent.")
#t2 = threading.Thread(target=self.send_message_insert(, name='t1')
#t2.start()

```

```

def playResponce(self, response):
    x=pyttsx3.init()
    #print(response)
    li = []
    if len(response) > 100:
        if response.find('--') == -1:
            b = response.split('--')
            #print(b)

    x.setProperty('rate',120)
    x.setProperty('volume',100)
    x.say(response)
    x.runAndWait()
    #print("Played Successfully.....")

```

```

def last_sent_label(self, date):

```

```

try:
    self.sent_label.destroy()
except AttributeError:
    pass

self.sent_label = Label(self.entry_frame, font="Verdana 7", text=date,
bg=self.tl_bg2, fg=self.tl_fg)
self.sent_label.pack(side=LEFT, fill=X, padx=3)

def clear_chat(self):
    self.text_box.config(state=NORMAL)
    self.last_sent_label(date="No messages sent.")
    self.text_box.delete(1.0, END)
    self.text_box.delete(1.0, END)
    self.text_box.config(state=DISABLED)

def chatexit(self):
    exit()

def msg(self):
    tkinter.messagebox.showinfo("PyBOT v1.0",'PyBOT is a chatbot for
answering python queries\nIt is based on retrival-based NLP using pythons NLTK
tool-kit module\nGUI is based on Tkinter\nIt can answer questions regarding python
language for new learners')

def about(self):
    tkinter.messagebox.showinfo("PyBOT Developers","1.Abhishek
Ezhava\n2.Mayur Kadam\n3.Monis Khot\n4.Raj Vishwakarma")

def send_message_insert(self, message):
    user_input = self.entry_field.get()

```

```

pr1 = "Human : " + user_input + "\n"
self.text_box.configure(state=NORMAL)
self.text_box.insert(END, pr1)
self.text_box.configure(state=DISABLED)
self.text_box.see(END)
#t1 = threading.Thread(target=self.playResponse, args=(user_input,))
#t1.start()
#time.sleep(1)
ob=chat(user_input)
pr="PyBot : " + ob + "\n"
self.text_box.configure(state=NORMAL)
self.text_box.insert(END, pr)
self.text_box.configure(state=DISABLED)
self.text_box.see(END)
self.last_sent_label(str(time.strftime( "Last message sent: " + '%B %d, %Y' + '
at ' + '%l:%M %p'))))
self.entry_field.delete(0,END)
time.sleep(0)
t2 = threading.Thread(target=self.playResponse, args=(ob,))
t2.start()
#return ob

```

```

def font_change_default(self):
    self.text_box.config(font="Verdana 10")
    self.entry_field.config(font="Verdana 10")
    self.font = "Verdana 10"

```

```

def font_change_times(self):
    self.text_box.config(font="Times")
    self.entry_field.config(font="Times")
    self.font = "Times"

def font_change_system(self):
    self.text_box.config(font="System")
    self.entry_field.config(font="System")
    self.font = "System"

def font_change_helvetica(self):
    self.text_box.config(font="helvetica 10")
    self.entry_field.config(font="helvetica 10")
    self.font = "helvetica 10"

def font_change_fixedsys(self):
    self.text_box.config(font="fixedsys")
    self.entry_field.config(font="fixedsys")
    self.font = "fixedsys"

def color_theme_default(self):
    self.master.config(bg="#EEEEEE")
    self.text_frame.config(bg="#EEEEEE")
    self.entry_frame.config(bg="#EEEEEE")
    self.text_box.config(bg="#FFFFFF", fg="#000000")
    self.entry_field.config(bg="#FFFFFF", fg="#000000",
insertbackground="#000000")
    self.send_button_frame.config(bg="#EEEEEE")
    self.send_button.config(bg="#FFFFFF", fg="#000000",
activebackground="#FFFFFF", activeforeground="#000000")
    #self.emoji_button.config(bg="#FFFFFF", fg="#000000",
activebackground="#FFFFFF", activeforeground="#000000")

```

```

self.sent_label.config(bg="#EEEEEE", fg="#000000")

self.tl_bg = "#FFFFFF"
self.tl_bg2 = "#EEEEEE"
self.tl_fg = "#000000"

# Dark
def color_theme_dark(self):
    self.master.config(bg="#2a2b2d")
    self.text_frame.config(bg="#2a2b2d")
    self.text_box.config(bg="#212121", fg="#FFFFFF")
    self.entry_frame.config(bg="#2a2b2d")
    self.entry_field.config(bg="#212121", fg="#FFFFFF",
insertbackground="#FFFFFF")
    self.send_button_frame.config(bg="#2a2b2d")
    self.send_button.config(bg="#212121", fg="#FFFFFF",
activebackground="#212121", activeforeground="#FFFFFF")
    # self.emoji_button.config(bg="#212121", fg="#FFFFFF",
activebackground="#212121", activeforeground="#FFFFFF")
    self.sent_label.config(bg="#2a2b2d", fg="#FFFFFF")

    self.tl_bg = "#212121"
    self.tl_bg2 = "#2a2b2d"
    self.tl_fg = "#FFFFFF"

# Grey
def color_theme_grey(self):
    self.master.config(bg="#444444")
    self.text_frame.config(bg="#444444")
    self.text_box.config(bg="#4f4f4f", fg="#ffffff")
    self.entry_frame.config(bg="#444444")
    self.entry_field.config(bg="#4f4f4f", fg="#ffffff", insertbackground="#ffffff")

```

```

self.send_button_frame.config(bg="#444444")
self.send_button.config(bg="#4f4f4f", fg="#ffffff", activebackground="#4f4f4f",
activeforeground="#ffffff")
self.emoji_button.config(bg="#4f4f4f", fg="#ffffff",
activebackground="#4f4f4f", activeforeground="#ffffff")
self.sent_label.config(bg="#444444", fg="#ffffff")

self.tl_bg = "#4f4f4f"
self.tl_bg2 = "#444444"
self.tl_fg = "#ffffff"

```

```

def color_theme_turquoise(self):
    self.master.config(bg="#003333")
    self.text_frame.config(bg="#003333")
    self.text_box.config(bg="#669999", fg="#FFFFFF")
    self.entry_frame.config(bg="#003333")
    self.entry_field.config(bg="#669999", fg="#FFFFFF",
insertbackground="#FFFFFF")
    self.send_button_frame.config(bg="#003333")
    self.send_button.config(bg="#669999", fg="#FFFFFF",
activebackground="#669999", activeforeground="#FFFFFF")
    self.emoji_button.config(bg="#669999", fg="#FFFFFF",
activebackground="#669999", activeforeground="#FFFFFF")
    self.sent_label.config(bg="#003333", fg="#FFFFFF")

    self.tl_bg = "#669999"
    self.tl_bg2 = "#003333"
    self.tl_fg = "#FFFFFF"

```

Blue

```

def color_theme_dark_blue(self):

```

```

self.master.config(bg="#263b54")
self.text_frame.config(bg="#263b54")
self.text_box.config(bg="#1c2e44", fg="FFFFFF")
self.entry_frame.config(bg="#263b54")
self.entry_field.config(bg="#1c2e44", fg="FFFFFF",
insertbackground="FFFFFF")
self.send_button_frame.config(bg="#263b54")
self.send_button.config(bg="#1c2e44", fg="FFFFFF",
activebackground="#1c2e44", activeforeground="FFFFFF")
#self.emoji_button.config(bg="#1c2e44", fg="FFFFFF",
activebackground="#1c2e44", activeforeground="FFFFFF")
self.sent_label.config(bg="#263b54", fg="FFFFFF")

self.tl_bg = "#1c2e44"
self.tl_bg2 = "#263b54"
self.tl_fg = "FFFFFF"

```

```

# Torque
def color_theme_turquoise(self):
    self.master.config(bg="#003333")
    self.text_frame.config(bg="#003333")
    self.text_box.config(bg="#669999", fg="FFFFFF")
    self.entry_frame.config(bg="#003333")
    self.entry_field.config(bg="#669999", fg="FFFFFF",
insertbackground="FFFFFF")
    self.send_button_frame.config(bg="#003333")
    self.send_button.config(bg="#669999", fg="FFFFFF",
activebackground="#669999", activeforeground="FFFFFF")
    #self.emoji_button.config(bg="#669999", fg="FFFFFF",

```

```

activebackground="#669999", activeforeground="#FFFFFF")
    self.sent_label.config(bg="#003333", fg="#FFFFFF")

    self.tl_bg = "#669999"
    self.tl_bg2 = "#003333"
    self.tl_fg = "#FFFFFF"

# Hacker
def color_theme_hacker(self):
    self.master.config(bg="#0F0F0F")
    self.text_frame.config(bg="#0F0F0F")
    self.entry_frame.config(bg="#0F0F0F")
    self.text_box.config(bg="#0F0F0F", fg="#33FF33")
    self.entry_field.config(bg="#0F0F0F", fg="#33FF33",
insertbackground="#33FF33")
    self.send_button_frame.config(bg="#0F0F0F")
    self.send_button.config(bg="#0F0F0F", fg="#FFFFFF",
activebackground="#0F0F0F", activeforeground="#FFFFFF")
    #self.emoji_button.config(bg="#0F0F0F", fg="#FFFFFF",
activebackground="#0F0F0F", activeforeground="#FFFFFF")
    self.sent_label.config(bg="#0F0F0F", fg="#33FF33")

    self.tl_bg = "#0F0F0F"
    self.tl_bg2 = "#0F0F0F"
    self.tl_fg = "#33FF33"

# Default font and color theme
def default_format(self):
    self.font_change_default()
    self.color_theme_default()

```

```
root=Tk()
```

```
a = ChatInterface(root)
root.geometry(window_size)
root.title("PyBot")
root.iconbitmap('i.ico')
root.mainloop()
```

B.IMPLEMENTED SCREENSHOTS

