

## **TABLE OF CONTENTS**

|   | <b>Page No</b> |
|---|----------------|
| <b>CHAPTER 1: ABSTRACT</b>                          | <b>01</b>      |
| 1.1 Introduction                                    |                |
| <b>CHAPTER 2: SYSTEM REQUIREMENT &amp; ANALYSIS</b> | <b>02</b>      |
| 2.1 System Analysis                                 |                |
| 2.1.1 Proposed of Voice Assistant                   |                |
| 2.2 Software & Hardware Requirements                |                |
| 2.2.1 Software used                                 |                |
| 2.2.2 Hardware Specifications                       |                |
| 2.2.3 Software Specifications                       |                |
| <b>CHAPTER 3: SYSTEM DESIGNING</b>                  | <b>07</b>      |
| 3.1 Data Flow Diagram                               |                |
| 3.2 Flow Chart                                      |                |
| <b>CHAPTER 4: SYSTEM IMPLEMENTATION</b>             | <b>11</b>      |
| 4.1 Implementation                                  |                |
| 4.2 Project Modules                                 |                |
| 4.3 Coding details                                  |                |
| 4.4 Screen Shots                                    |                |
| <b>CHAPTER 5: SYSTEM TESTING</b>                    | <b>23</b>      |
| <b>CHAPTER 6: CONCLUSION</b>                        | <b>25</b>      |
| <b>CHAPTER 7: BIBLIOGRAPHY</b>                      | <b>26</b>      |

## **CHAPTER - 1**

### **ABSTRACT**

#### **OBJECTIVE :**

The objective of this code is to provide a simple voice assistant program that can perform various tasks based on user voice commands. The program, called "Carnage", uses several libraries to recognize user voice commands, generate speech responses, play songs on YouTube, send WhatsApp messages, search Wikipedia for information, and tell jokes.

The goal of the program is to make it easy for users to perform tasks without having to type or click on a computer or phone. The program is designed to listen for user voice commands and respond with spoken output, making it more convenient for users who prefer to interact with technology using voice commands.

#### **1.1 INTRODUCTION :**

This is a Python script that implements a virtual assistant named "Carnage" capable of performing a variety of tasks, including playing music on YouTube, sending WhatsApp messages, searching for information on Wikipedia, and telling jokes. The script uses several Python libraries, including `speech_recognition`, `pyttsx3`, `pywhatkit`, `wikipedia`, and `pyjokes`, and contains several functions to handle speech input and output.

This code can serve as the basis for a mini project aimed at building a personal virtual assistant that can assist users in their daily tasks using voice commands. By modifying the existing code and adding new features, students can explore various aspects of natural language processing and artificial intelligence, such as speech recognition, text-to-speech conversion, language understanding, and task execution. The project can be extended to include additional functionalities and can serve as an excellent platform for experimenting with cutting-edge technologies in the field of AI.

## CHAPTER - 2

### SYSTEM REQUIREMENT & ANALYSIS

#### 2.1 SYSTEMS ANALYSIS:

##### **Functional Requirements:**

The system should be able to perform basic tasks such as playing a song on YouTube, sending a message to a phone number, and responding to voice commands. The assistant should be able to recognize the user's voice input and perform the desired task accurately.

##### **User Interface:**

The user interacts with the system through voice commands. The system prompts the user for input using the text-to-speech engine, listens to the user's voice input using the speech recognition engine, and responds to the user's commands through the text-to-speech engine. The user can initiate tasks by saying specific commands, such as "play song" or "send message".

##### **Input/Output:**

The system takes voice input from the user and provides output through the text-to-speech engine. The user provides the input by speaking commands into the microphone, and the system provides the output by speaking responses to the user's commands.

##### **Error Handling:**

The system should be able to handle errors that occur due to inaccuracies in speech recognition or invalid user input. If the system cannot recognize the user's command, it should respond with an error message. Similarly, if the phone number entered by the user is invalid, the system should prompt the user to enter a valid phone number.

Overall, the system provides a basic framework for a voice-activated personal assistant that can perform simple tasks. With additional features and error handling, this system can be improved to be more useful in practical applications.

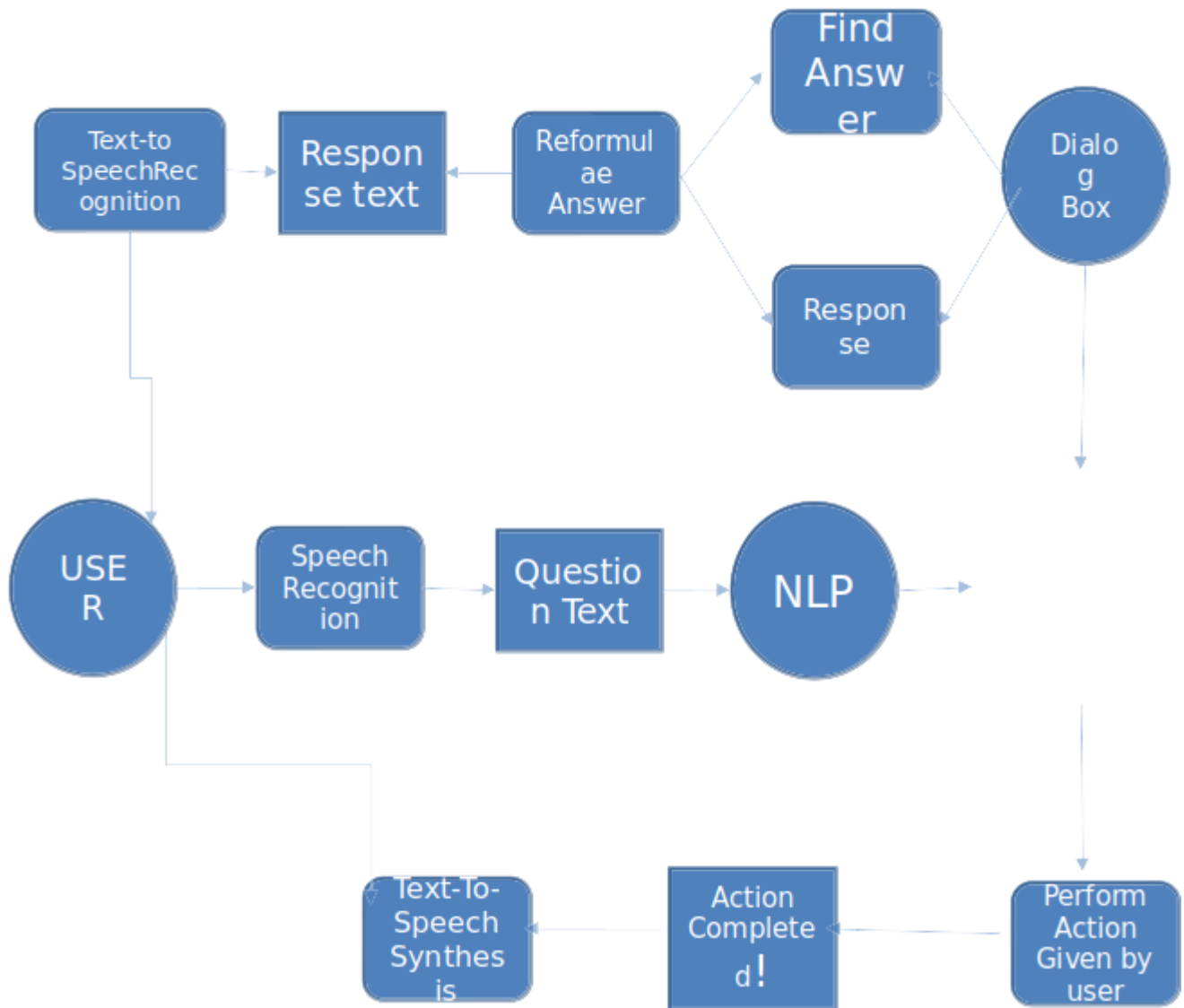
### 2.1.1 PROPOSED MODEL OF VOICE ASSISTANT:

The proposed model of the voice assistant is as shown in the below figure . The model consists of user input through a microphone to accept commands from the user. These commands then go through Speech Recognition, which is the ability of a machine or program to identify words and phrases in spoken languages and convert them to a machine-readable format. On these inputs Natural Language Processing is applied, it is a field which is created by amalgamating computer science and artificial intelligence. Using NLP, we are concerned with interactions between computers and human natural languages.

Then the BRAIN check whether it is a question or an action, if it is a action than the action is performed by the voice assistant and acknowledgment is given to the user via a synthesis voice or if it is a question than it is search in dialog box or knowledge base and then response via a synthesis voice to the user. Our Voice assistant uses google text-to-speech API to understand all the words spoken by the user, and based on certain conditions that satisfy being a command the voice assistant sends responses to the user.

The proposed model can be further extended to include additional modules and functionalities, such as setting reminders, scheduling appointments, making phone calls, and controlling smart home devices. The model can also be improved by incorporating advanced machine learning and natural language processing techniques to enhance the accuracy and efficiency of the speech recognition and text-to-speech conversion components. Overall, the proposed model provides a powerful platform for building intelligent virtual assistants that can assist users in various domains.

The virtual assistant can understand natural language input from the user using speech recognition, and can respond back to the user using text-to-speech conversion. The model consists of several modules, each responsible for a specific task, such as playing music, sending messages, searching for information, and telling jokes. These modules can be invoked by the user by speaking the corresponding commands, which are then processed by the virtual assistant.



## 2.2 HARDWARE & SOFTWARE REQUIREMENTS:

### 2.2.1 SOFTWARE USED :

#### VISUAL STUDIO CODE VERSION 1.77

\* Visual Studio Code (often referred to as VS Code) is a free, open-source code editor developed by Microsoft. It's a cross-platform editor that can run on Windows, macOS, and Linux, making it accessible to developers across different platforms.

\* VS Code is highly customizable and comes with many features and extensions that make it suitable for various programming languages and frameworks. Some of its key features include:

- **IntelliSense:**

VS Code provides intelligent code completion, code navigation, and code refactoring suggestions to help developers write code more efficiently.

- **Debugging:**

VS Code supports debugging for various programming languages and frameworks, allowing developers to debug their code within the editor.

- **Extensions:**

VS Code has a vast marketplace of extensions that developers can install to enhance their workflow and add support for different programming languages and frameworks

- **Git integration:**

VS Code has built-in Git integration that allows developers to manage their code repositories directly within the editor.

- **Live Share:**

VS Code provides a collaborative coding experience through its Live Share feature, which allows multiple developers to work on the same codebase in real-time.

- **Integrated terminal:**

VS Code comes with an integrated terminal that allows developers to run commands and interact with the command line without leaving the editor.

## **VIRTUAL ASSISTANT**

### **2.2.2 HARDWARE SPECIFICATION:**

**Processor** : Intel Core 2 Duo(Low end device does not support)

**Memory** : Higher than 4GB RAM

**Primary & secondary** : 20 GB Hard disk or higher

**Monitor** : LCD or higher

**Keyboards** : 104 keys

**Pointing device or Mouse** : Two or three button

### **2.2.3 SOFTWARE SPECIFICATIONS:**

**Operating System** : Windows 8/10/11 and Linux

**Supported OS** : Windows , Linux , Mac OS

**Software editor used** : Visual Studio Code Version 1.77

## CHAPTER - 3

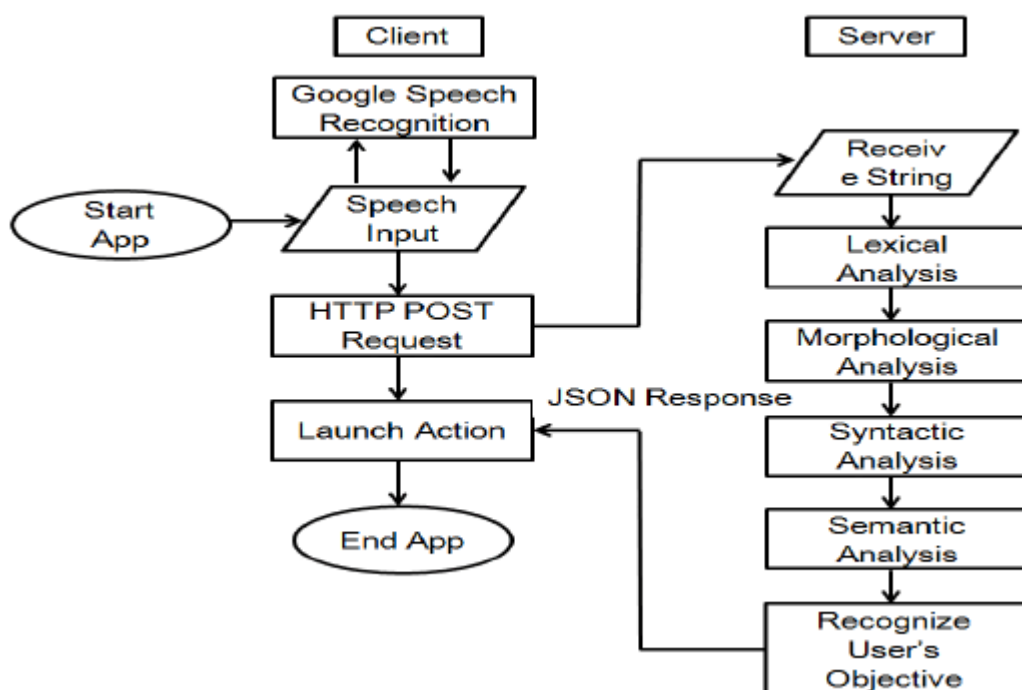
### SYSTEM DESIGNING

#### 3.1 DATA FLOW DIAGRAM:

A data-flow diagram is a way of representing a flow of data through a process or a system (usually an information system). The DFD also provides information about the outputs and inputs of each entity and the process itself. A data-flow diagram has no control flow — there are no decision rules and no loops. The DFD is a network representation of the system. They are excellent mechanisms for communicating with customers during requirement analysis. A DFD, also known as bubble chart, which clarifies system requirements identifying major transformations.

##### Level 0 DFD:

The user gives the input in the form of voice; this voice command is recognized by the application. Then it will check whether it is the authorized user, then action is performed as per the command given by the user. Command given is compared as a form of action and question and responded with the dialog box or search through the knowledge base.





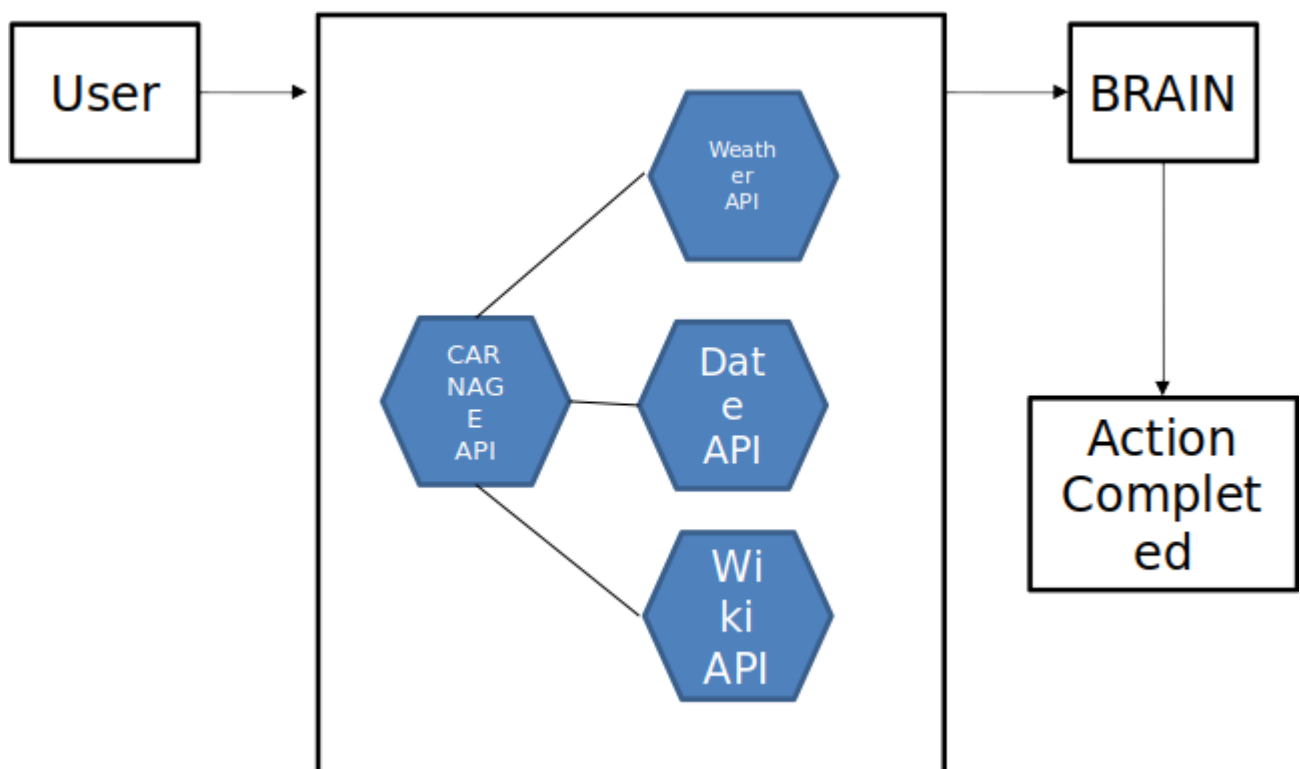
## VIRTUAL ASSISTANT

### Level 1 DFD:

Input is given by the user in the form of voice. GoogleVoiceAPI will convert this voice data in text form and then the action is performed by the voice assistant according to the command given by the user by comparing with the dialog box and knowledge base.

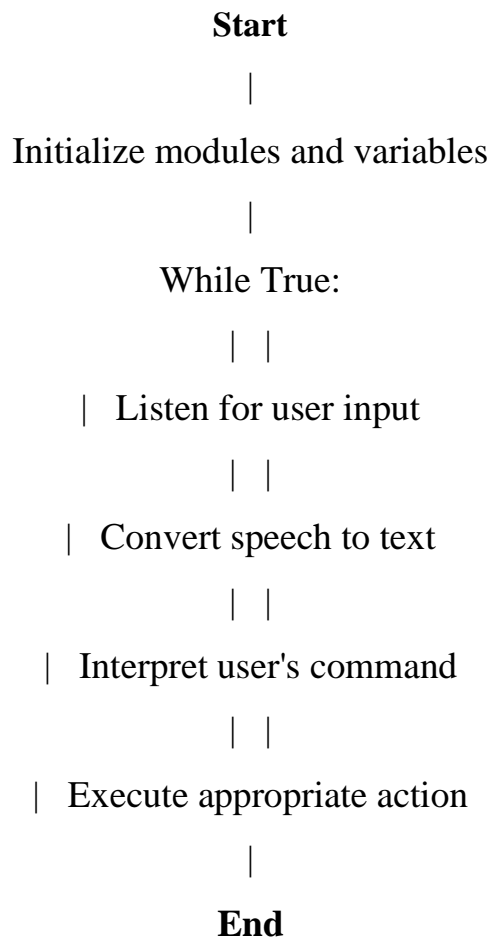
The natural language processing process might receive input from the speech recognition process and send output to the command interpretation process. The command interpretation process might then send data to the response generation process, which would generate a response and send it back to the user.

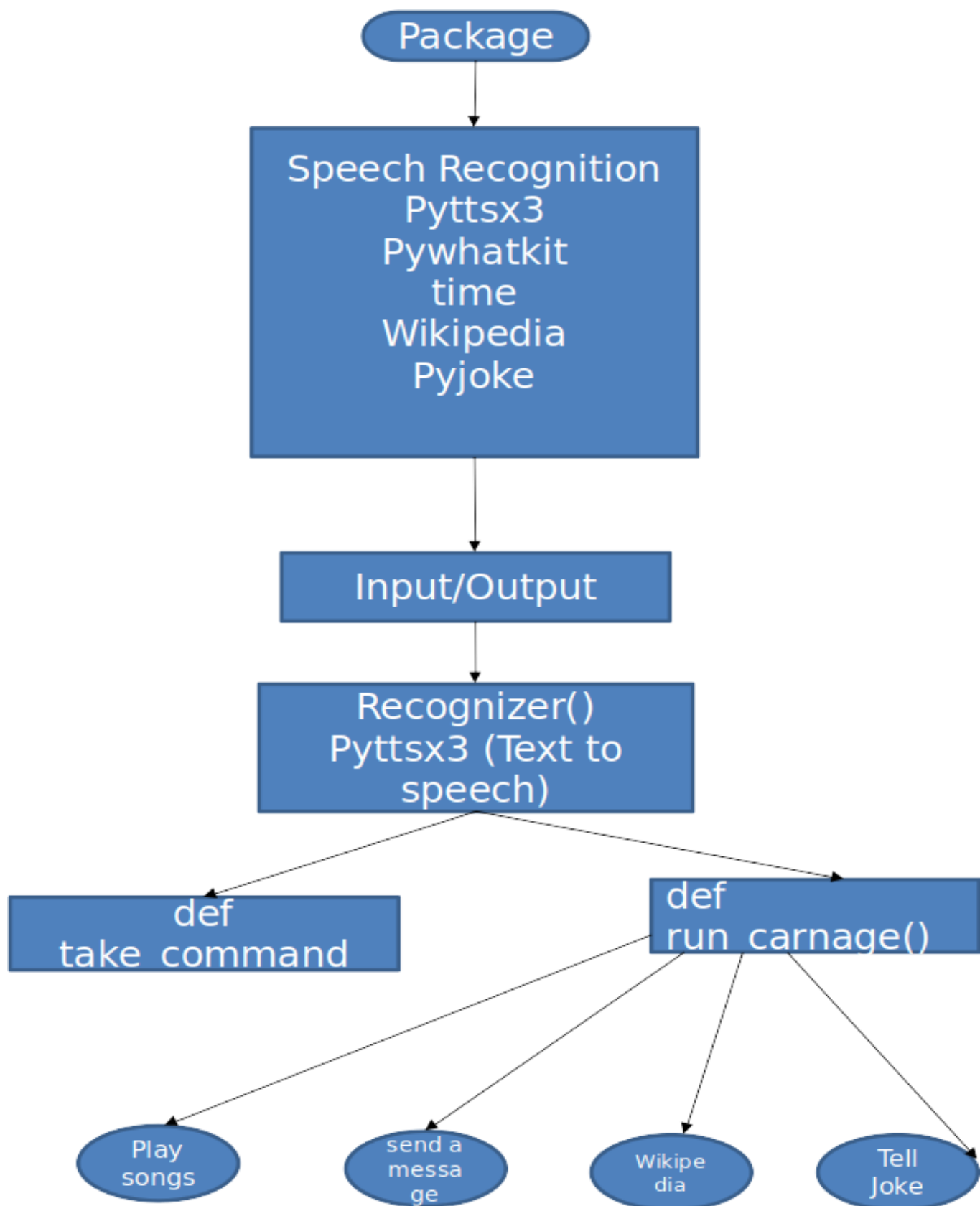
The Level 1 DFD would provide a clear overview of how the different components of the Voice Assistant system interact with each other, making it easier to understand and manage the system as a whole.



### 3.2 FLOWCHART:

Flow chart is the graphical representation of algorithms. Different symbols are used to represent flow charts. As the system is started, it first authenticates the authorized user, then the voice assistant is on running in the background listening for available voice commands; once the user gives a command, based on the conditions provided to the voice assistant, the voice assistant gives the necessary output. This output is sent to the Speech Recognition which converts the speech into machine-readable form. Based on the input received the personal voice assistant then performs the desired task.





## CHAPTER - 4

### SYSTEM IMPLEMENTATION

#### 4.1 IMPLEMENTATION:

The code begins by importing various libraries including time, speech\_recognition, pyttsx3, pywhatkit, wikipedia, and pyjokes.

It then retrieves the current time using the time library, which will be used later for sending messages.

The program uses the speech recognition library to listen for voice commands. It initializes a recognizer object and sets it to adjust for ambient noise. It also initializes a text-to-speech engine using the pyttsx3 library and sets the voice to use.

There is a function defined to speak a given text using the text-to-speech engine. Another function is defined to listen for a command using the speech recognition library.

The program contains a while loop that continuously listens for voice commands. When a command is recognized, it executes the corresponding action.

- If the command contains the word "play", the program removes the word "play" and "playing" from the command and uses the pywhatkit library to search and play the song on YouTube.
- If the command contains the word "message", the program prompts the user to speak the phone number of the recipient, followed by the message content. It then uses the pywhatkit library to send the message to the recipient.
- If the command contains the word "Wikipedia", the program searches for the given topic using the wikipedia library and speaks a summary of the topic.
- If the command contains the word "joke", the program uses the pyjokes library to tell a random joke.
- If the command does not match any of the defined actions, the program will apologize and ask the user to repeat the command.

### 4.2 PROJECT MODULES:

#### SPEECH RECOGNITION:

- Recognizer() Function

The Recognizer() function initializes the speech recognition engine.

- listen(source) Function

The listen(source) function records audio from a source, such as a microphone, and returns an audio data.

- recognize\_google(audio) Function

The recognize\_google(audio) function converts audio data to text using the Google Web Speech API.

- adjust\_for\_ambient\_noise(source) Function

The adjust\_for\_ambient\_noise(source) function adjusts the energy threshold for capturing audio from a noisy environment.

#### PYTTSX3 MODULE:

- init() Function

The init() function initializes the pyttsx3 text-to-speech engine.

- getProperty(name) Function

The getProperty(name) function gets the value of a pyttsx3 engine property.

- setProperty(name, value) Function

The setProperty(name, value) function sets the value of a pyttsx3 engine property.

- say(text) Function

The say(text) function speaks the specified text using the pyttsx3 engine.

- runAndWait() Function

## VIRTUAL ASSISTANT

The `runAndWait()` function blocks the program until all currently queued commands are executed.

### PYWHATKIT MODULE:

- `playonyt(query)` Function

The `playonyt(query)` function opens a YouTube video with the specified query in the default web browser.

- `sendwhatmsg(phone_no, message, time_hour, time_min)` Function

The `sendwhatmsg(phone_no, message, time_hour, time_min)` function sends a WhatsApp message to the specified phone number with the specified message at the specified time.

### TIME MODULE:

- `localtime()` Function

The `localtime()` function gets the current time in the local timezone.

- `strftime(format, time)` Function

The `strftime(format, time)` function converts a time object into a string with a specified format.

`time.strftime("%H",t)` is used to get the current hour in 24-hour format and `time.strftime("%M",t)` is used to get the current minute. `int(min) + int(1))%60` is used to get the minute one minute ahead of the current time, accounting for cases where the current time is near the end of an hour.

### WIKIPEDIA MODULE:

The `wikipedia` module is a Python library that makes it easy to access and search Wikipedia articles programmatically. It provides a simple interface to search Wikipedia articles, get summaries, and extract other useful information from the articles. In this code, the `wikipedia` module is used to perform a search on Wikipedia and get a summary of the search results based on the user's command. The summary is then spoken out loud to the user using the `talk` function.

## VIRTUAL ASSISTANT

### 4.3 CODING DETAILS:

```
import time
import speech_recognition as sr
import pyttsx3
import pywhatkit
import wikipedia
import pyjokes
import wikipedia

##time section
t = time.localtime()
hour = time.strftime("%H",t)
minut = time.strftime("%M",t)
send = (int(minut) + int(1))%60
print(hour,send)
##listen sec
listener = sr.Recognizer()
engine = pyttsx3.init()
voices = engine.getProperty('voices')
engine.setProperty('voice', voices[1].id)

## talk section
def talk(text):
    engine.say(text)
    engine.runAndWait()

def take_command():
    try:
        with sr.Microphone() as source:
            print("i am listening....")
            listener.adjust_for_ambient_noise(source)
            voice = listener.listen(source)
            command = listener.recognize_google(voice)
            command = command.lower()

            if 'carnage' in command:
                command = command.replace('carnage','')
                print(command)
                talk('sure ,chief')

    except sr.UnknownValueError:
        talk("Sorry, I didn't understand that.")
        command = " "
    except sr.RequestError:
        talk("Sorry, my speech service is down.")
```

## VIRTUAL ASSISTANT

```
command = "  
return command
```

```
##request executable block for all
```

```
def run_carnage():
```

```
    while True:
```

```
        command = take_command()
```

```
        print(command)
```

```
    if 'play' in command:
```

```
        song = command.replace('playing', '').replace('play', '').strip()
```

```
        if song:
```

```
            talk('playing' + song)
```

```
            pywhatkit.playonyt(song)
```

```
        else:
```

```
            talk('Sorry, I didn\'t catch the name of the song can you repeat again.')
```

```
    ##msg block
```

```
    elif 'message' in command:
```

```
        talk('To whom should I send the message?')
```

```
        with sr.Microphone() as source:
```

```
            voice = listener.listen(source)
```

```
            phone_number = listener.recognize_google(voice)
```

```
            phone_number = phone_number.strip()
```

```
    ##checking stage for msg phno
```

```
    if len(phone_number) == 10 or not phone_number.isdigit():
```

```
        message_content = "
```

```
        talk('What should i want to send?')
```

```
        with sr.Microphone() as source:
```

```
            voice = listener.listen(source)
```

```
            message_content = listener.recognize_google(voice)
```

```
            message_content = message_content.strip()
```

```
    ##msg not received
```

```
    if not message_content:
```

```
        talk('Sorry, I didn\'t catch the message. Please try again.')
```

```
        continue
```

```
    ##msg to deliver
```

```
    try:
```

```
        pywhatkit.sendwhatmsg(f"+91{phone_number}", message_content, int(hour), int(send))
```

```
        talk('Message sent successfully.')
```

```
    except:
```

```
        talk('Sorry, an error occurred while sending the message.')
```

```
    ##wikipedia block
```

```
    elif 'wikipedia' in command:
```

```
        search = command.replace('search', '').replace('wikipedia', '').strip()
```

```
        try:
```



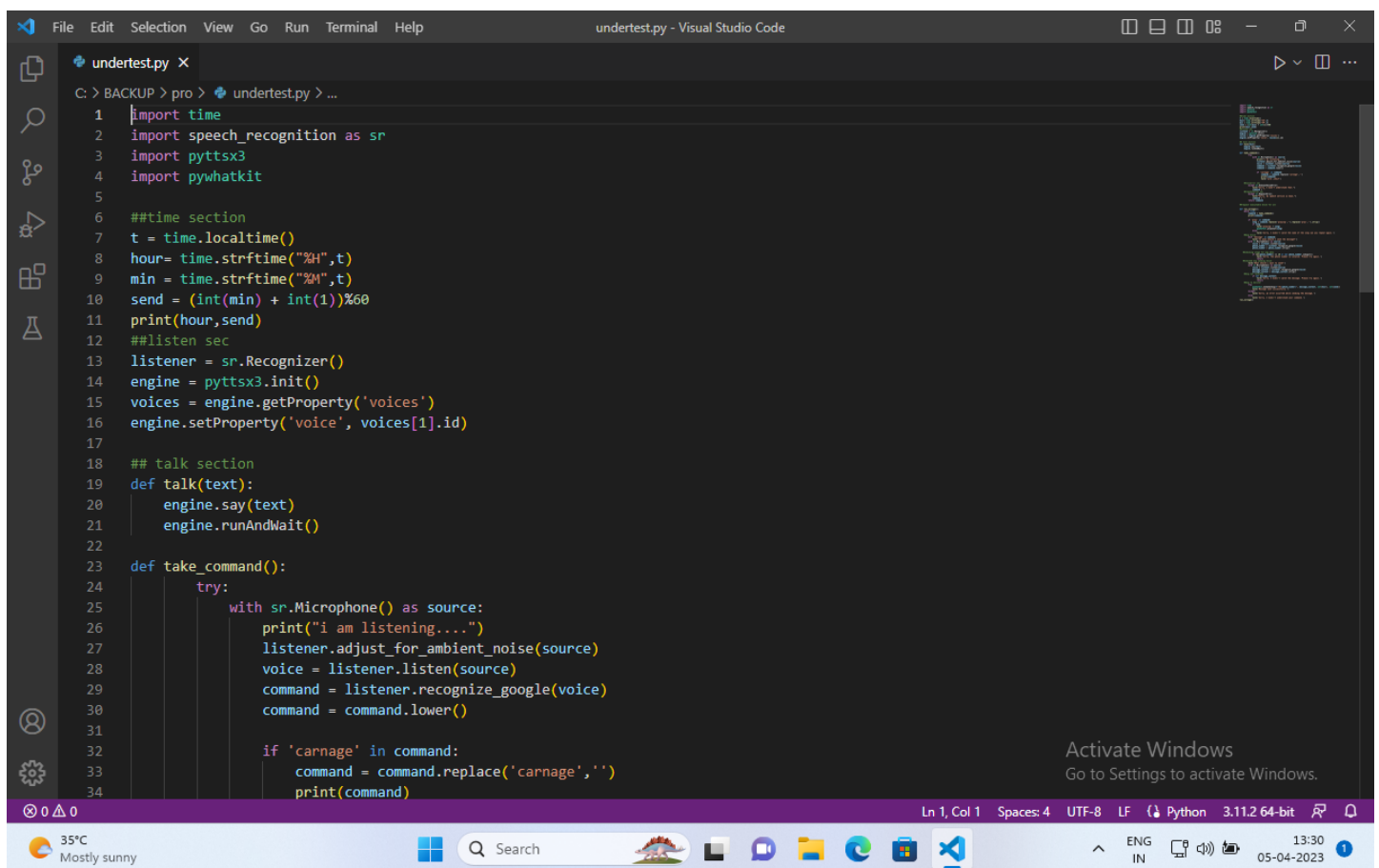
## VIRTUAL ASSISTANT

```
        info = wikipedia.summary(search, sentences=1)
        talk(f'According to Wikipedia, {info}')
    except:
        talk('Sorry, I couldn\'t find any information on that topic.')

##pyjokes block
elif 'joke' in command:
    talk(pyjokes.get_joke())
else:
    talk('Sorry, I didn\'t understand your command.')

run_carnage()
```

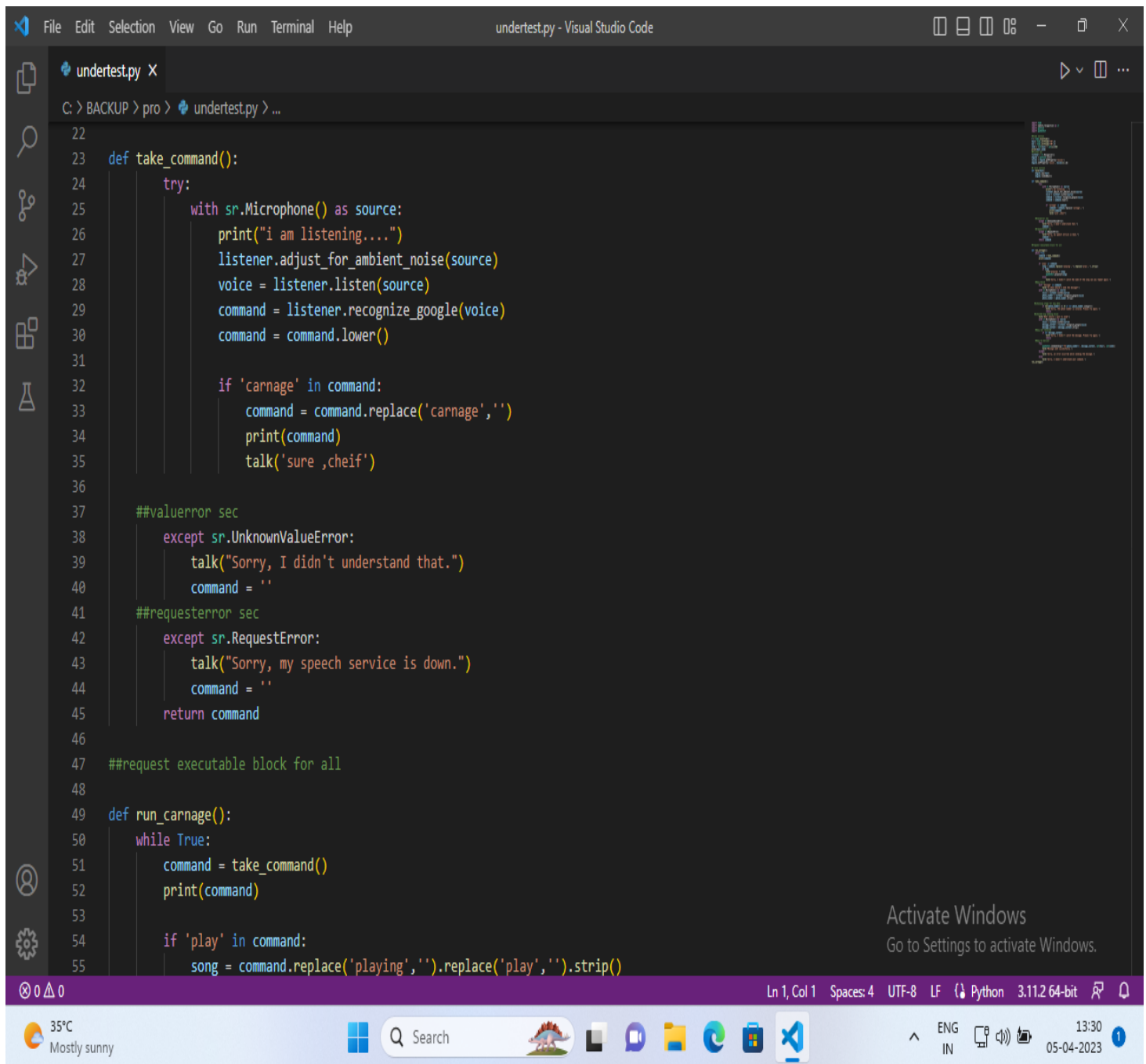
## 4.4 SCREEN SHOTS:



```
File Edit Selection View Go Run Terminal Help
undertest.py - Visual Studio Code

undertest.py X
C: > BACKUP > pro > undertest.py > ...
1  import time
2  import speech_recognition as sr
3  import pyttsx3
4  import pywhatkit
5
6  ##time section
7  t = time.localtime()
8  hour= time.strftime("%H",t)
9  min = time.strftime("%M",t)
10 send = (int(min) + int(1))%60
11 print(hour,send)
12
13 ##listen sec
14 listener = sr.Recognizer()
15 engine = pyttsx3.init()
16 voices = engine.getProperty('voices')
17 engine.setProperty('voice', voices[1].id)
18
19 ## talk section
20 def talk(text):
21     engine.say(text)
22     engine.runAndWait()
23
24 def take_command():
25     try:
26         with sr.Microphone() as source:
27             print("i am listening....")
28             listener.adjust_for_ambient_noise(source)
29             voice = listener.listen(source)
30             command = listener.recognize_google(voice)
31             command = command.lower()
32
33             if 'carnage' in command:
34                 command = command.replace('carnage','')
35                 print(command)
```

## VIRTUAL ASSISTANT



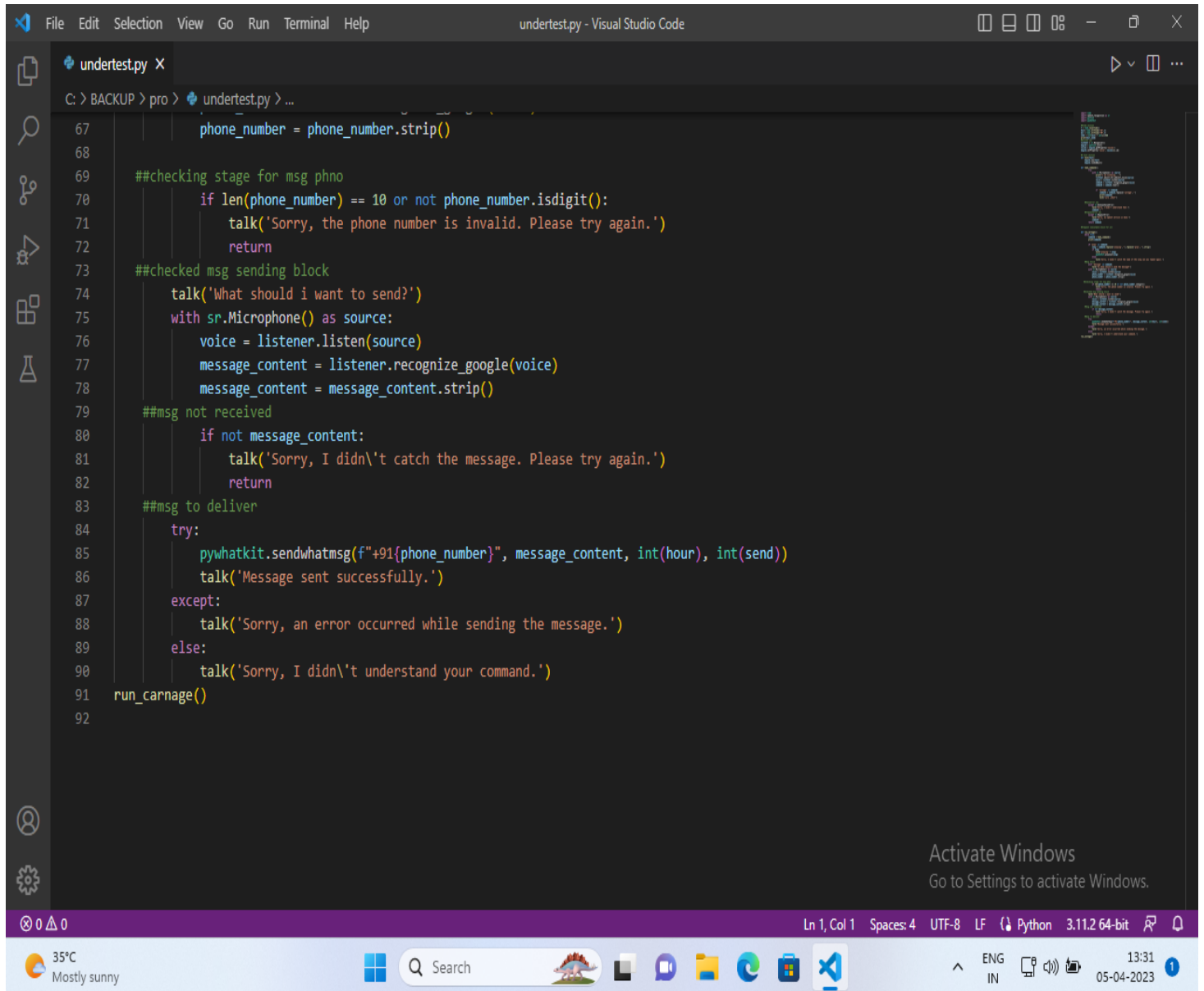
```
22
23 def take_command():
24     try:
25         with sr.Microphone() as source:
26             print("i am listening...")
27             listener.adjust_for_ambient_noise(source)
28             voice = listener.listen(source)
29             command = listener.recognize_google(voice)
30             command = command.lower()
31
32             if 'carnage' in command:
33                 command = command.replace('carnage','')
34                 print(command)
35                 talk('sure ,cheif')
36
37     ##valuererror sec
38     except sr.UnknownValueError:
39         talk("Sorry, I didn't understand that.")
40         command = ''
41     ##requesterror sec
42     except sr.RequestError:
43         talk("Sorry, my speech service is down.")
44         command = ''
45     return command
46
47     ##request executable block for all
48
49 def run_carnage():
50     while True:
51         command = take_command()
52         print(command)
53
54         if 'play' in command:
55             song = command.replace('playing','').replace('play','').strip()
```

Activate Windows  
Go to Settings to activate Windows.

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.11.2 64-bit

35°C Mostly sunny Search 13:30 05-04-2023

## VIRTUAL ASSISTANT



```
67     phone_number = phone_number.strip()
68
69     ##checking stage for msg phno
70     if len(phone_number) == 10 or not phone_number.isdigit():
71         talk('Sorry, the phone number is invalid. Please try again.')
72         return
73     ##checked msg sending block
74     talk('What should i want to send?')
75     with sr.Microphone() as source:
76         voice = listener.listen(source)
77         message_content = listener.recognize_google(voice)
78         message_content = message_content.strip()
79     ##msg not received
80     if not message_content:
81         talk('Sorry, I didn\'t catch the message. Please try again.')
82         return
83     ##msg to deliver
84     try:
85         pywhatkit.sendwhatmsg(f"+91{phone_number}", message_content, int(hour), int(send))
86         talk('Message sent successfully.')
87     except:
88         talk('Sorry, an error occurred while sending the message.')
89     else:
90         talk('Sorry, I didn\'t understand your command.')
91 run_carnage()
92
```

Activate Windows  
Go to Settings to activate Windows.

35°C Mostly sunny

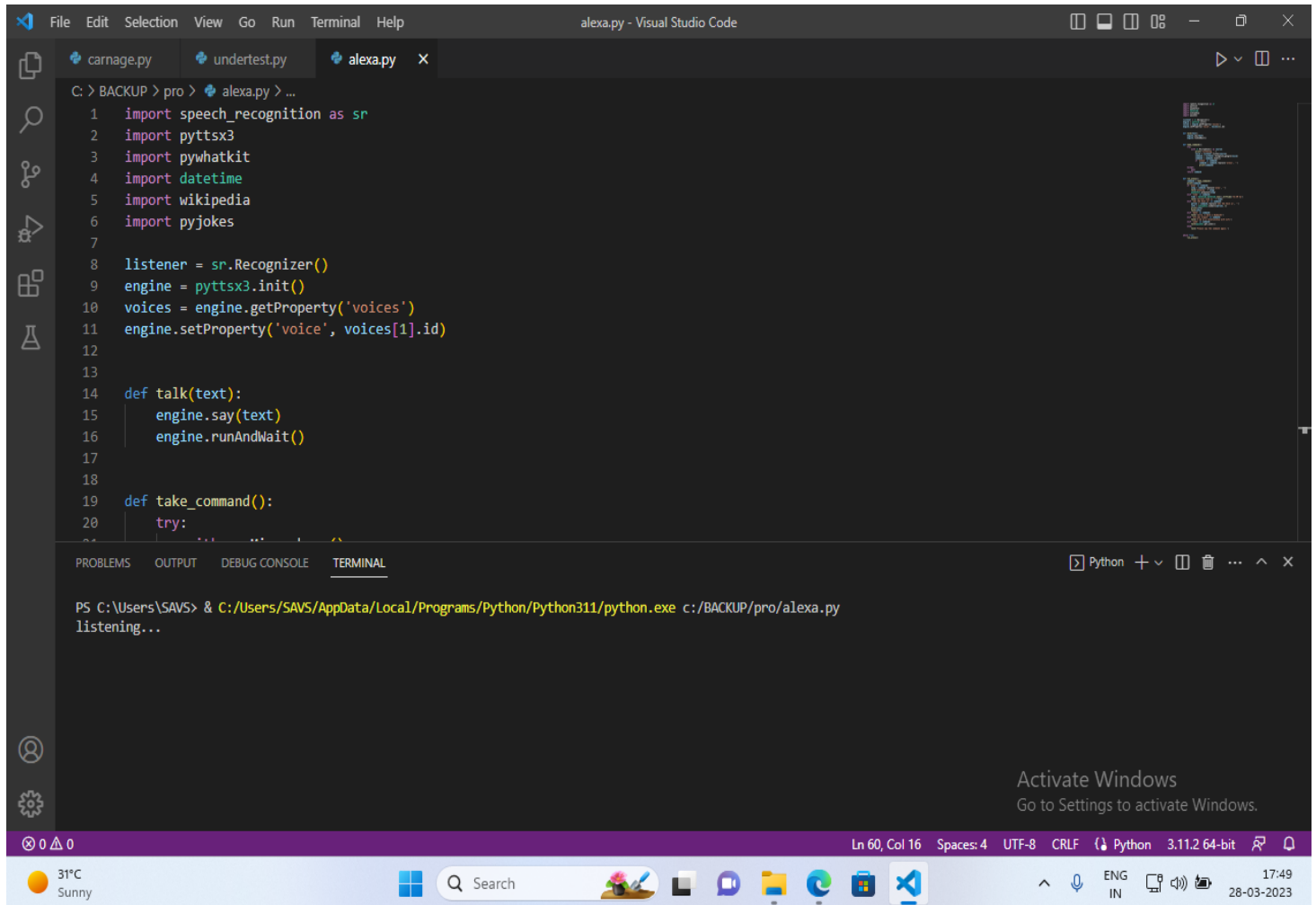
Search

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.11.2 64-bit

ENG IN 13:31 05-04-2023

# VIRTUAL ASSISTANT

## Listening to commands



The screenshot displays the Visual Studio Code interface. The editor window shows the file `alexa.py` with the following Python code:

```
C: > BACKUP > pro > alexa.py > ...
1  import speech_recognition as sr
2  import pyttsx3
3  import pywhatkit
4  import datetime
5  import wikipedia
6  import pyjokes
7
8  listener = sr.Recognizer()
9  engine = pyttsx3.init()
10 voices = engine.getProperty('voices')
11 engine.setProperty('voice', voices[1].id)
12
13
14 def talk(text):
15     engine.say(text)
16     engine.runAndWait()
17
18
19 def take_command():
20     try:
```

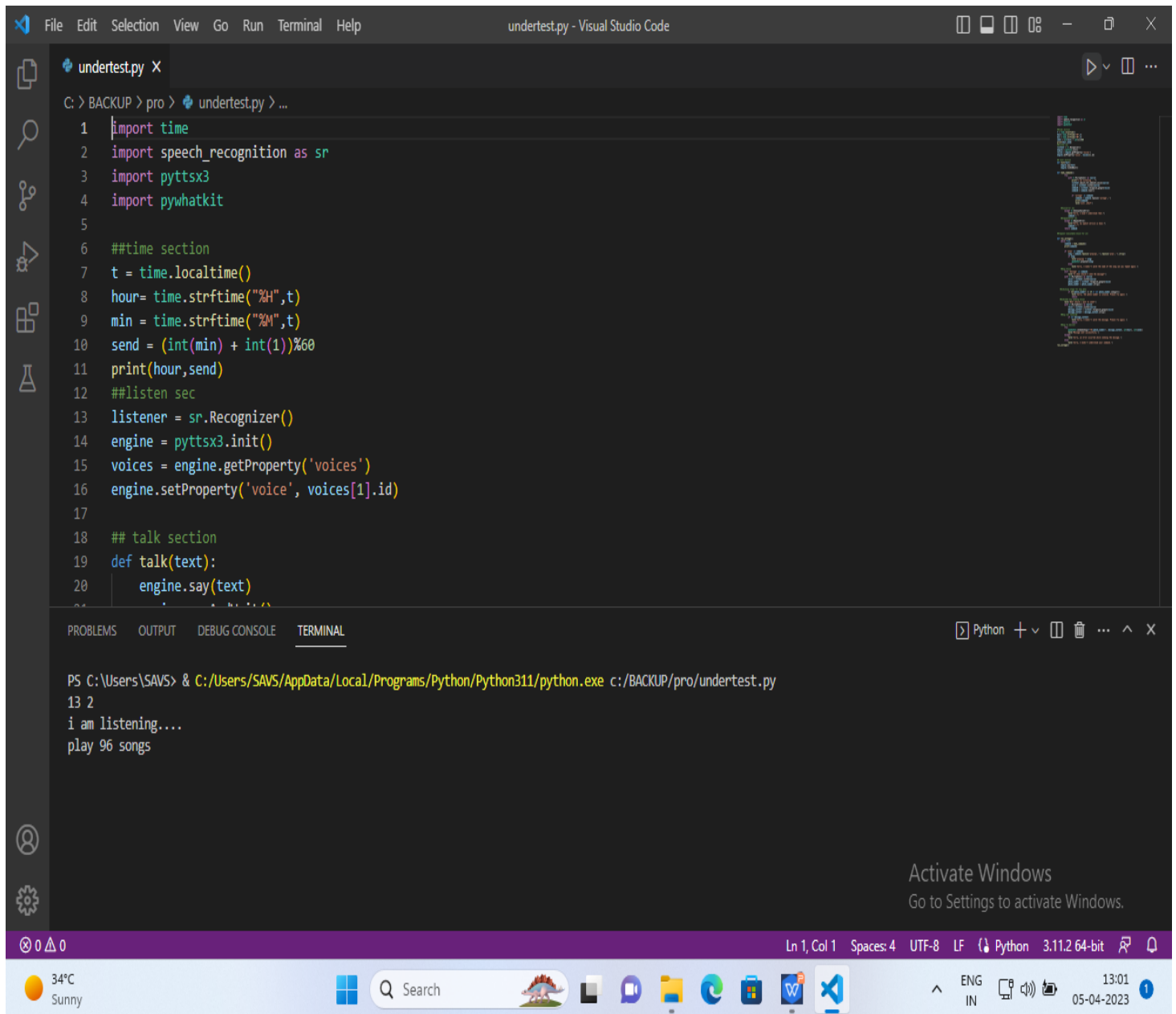
The terminal window at the bottom shows the command prompt running the script:

```
PS C:\Users\SAVS> & C:/Users/SAVS/AppData/Local/Programs/Python/Python311/python.exe c:/BACKUP/pro/alexa.py
listening...
```

The status bar at the bottom indicates the current position is Line 60, Column 16, with 4 spaces. The encoding is UTF-8 and the line ending is CRLF. The Python version is 3.11.2 64-bit. The system tray shows the date and time as 17:49 on 28-03-2023.

## VIRTUAL ASSISTANT

Asking to play music



The screenshot shows the Visual Studio Code editor with a file named `undertest.py` open. The code is a Python script that uses the `speech_recognition` and `pyttsx3` libraries to create a simple virtual assistant. It includes a time section to print the current time and a listen section to recognize voice input. The terminal shows the command to run the script, and the output indicates that the assistant is listening and has received the command "play 96 songs".

```
1 import time
2 import speech_recognition as sr
3 import pyttsx3
4 import pywhatkit
5
6 ##time section
7 t = time.localtime()
8 hour= time.strftime("%H",t)
9 min = time.strftime("%M",t)
10 send = (int(min) + int(1))%60
11 print(hour,send)
12
13 ##listen sec
14 listener = sr.Recognizer()
15 engine = pyttsx3.init()
16 voices = engine.getProperty('voices')
17 engine.setProperty('voice', voices[1].id)
18
19 ## talk section
20 def talk(text):
21     engine.say(text)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

PS C:\Users\SAVS> & C:/Users/SAVS/AppData/Local/Programs/Python/Python311/python.exe c:/BACKUP/pro/undertest.py

13 2

i am listening....

play 96 songs

Activate Windows  
Go to Settings to activate Windows.

Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.11.2 64-bit

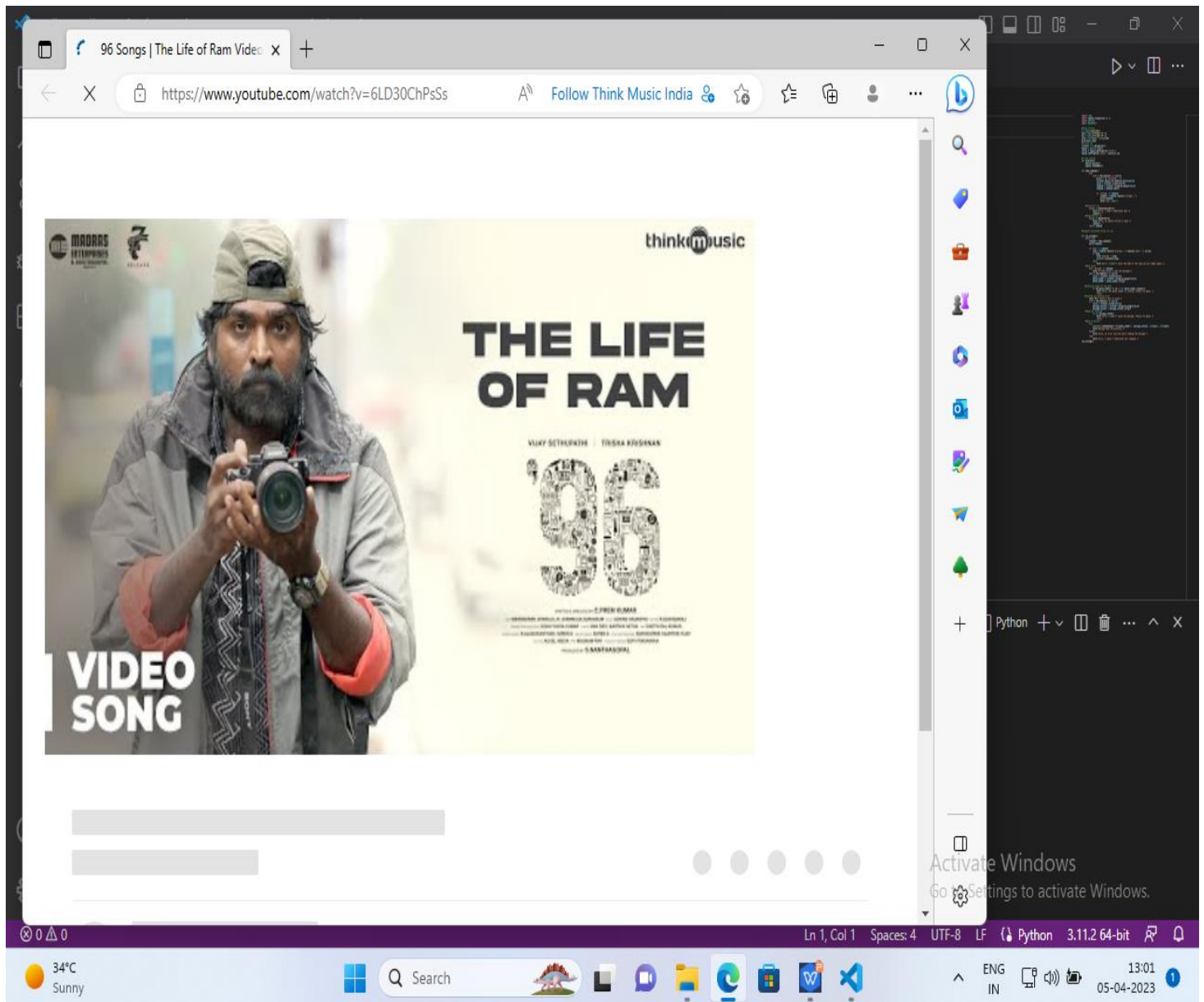
34°C Sunny

Search

ENG IN 13:01 05-04-2023

## VIRTUAL ASSISTANT

Playing song from commands



## Playing song from Youtube



## CHAPTER – 5

### SYSTEM TESTING

#### **Method Accepted for system testing:**

The performance is evaluated as per the number of platform interactions and query-based responses to be measured and optimized. For a virtual assistant's performance testing, its three main components –

- The bot framework
- Bot service
- Channel tests

Load testing of Advanced Virtual Assistants (AVAs) can be done using JMeter while testing the others depends on how they have been deployed.

#### **Testing objectives:**

Here are a few key points that you should consider for VA performance testing:

- Check out the performance of virtual assistants with multiple connections of different bandwidths including 2G, 3G, 4G, 5G and Wi-Fi.
- Make sure the application is working smoothly while it switches from one network to another.
- Make sure the messages are delivered instantly to the user.
- Ensure the videos and images shared by users are loading fast with no resolution issues
- Check whether the quality of video and voice calls are smooth and clear. Load test the application to identify how many concurrent customers can access the application. You can use JMeter for this testing.
- Find out the maximum number of users that can be active in a group chat concurrently.



## **VIRTUAL ASSISTANT**

### **Key metrics to test virtual assistants:**

It is essential to keep in mind the below mentioned key metrics when testing virtual assistants:

#### **1. Response time:**

Normally, AI-enabled virtual assistants are expected to reply instantly after an input is received from the user. Any lag in response can affect the user experience. Hence, it is business-critical to test the exact response time of the VA to ensure a customer delight experience.

#### **2. Answering/Response accuracy:**

Test the responses or answers by VAs to inputs or queries to ensure they are meeting the business objective efficiently.

#### **3. Error management:**

AI-enabled VAs are equipped to handle errors and address them. For instance, if a virtual assistant is unable to process user inputs, it should be able to think alternately and ask different questions from the user to eliminate dead air in the conversation. In the worst case scenario, it should transfer the call to a live agent.

#### **4. Personality:**

Virtual assistants should have a unique tone that is pleasant for users. To ensure a great customer experience, its tone should be tested and rectified if needed.

#### **5. Navigation:**

Customers often feel lost while communicating with a virtual assistant. Make sure the navigation of the VA is user-friendly for a seamless experience.

#### **6. Wrapping Up**

With increasing adoption of advanced virtual assistants in multiple businesses to offer better customer experience, performance testing of VAs has become business critical. Only robust performance testing can help you launch AI-enabled virtual assistants that are reliable, humane and scalable.

## **CHAPTER - 6**

### **CONCLUSION**

This system is designed in such a method wherein the user can accommodate it effortlessly. Our proposed system BRAIN – The A.I. A personal voice assistant can be implemented using the face recognition and using speech recognition module thus makes the system more secure and robust. The contributions of Smart Voice Assistant are twofold. First, the face recognition technique makes it more secure and robust to use. Secondly, it is the voice control application that provides enhancements to all applications running on a system by synthesizing commands set from onscreen context. BRAIN can benefit a large number of users with universal eyes free and hands free voice control of their system. Speech recognition technology is a key technology which will provide a new way of human interaction with machines or tools. The advantage of voice commands over multi-touch when interacting with a screen non-visually is that it does not require targets to be located and thus avoids the problems with pointing, it saves time. The sending of Email, and reading of News can be possible by the blind people also. This can do a variety of tasks like tell you the time, open applications, organize files, give updates of matches, play games, tell you the location, tell some jokes, open hackathons, do calculations, updates about the stock and the endless tasks for the user. Thus making one's life comfortable and at the same time remotely accessible via voice command.

## **CHAPTER 7**

### **BIBLIOGRAPHY**

1. Pankaj Jalote, “Software Engineering”, First Edition.
2. Pankaj Jalote, “An Integrated Approach to Software Engineering”, Narosa Publications.
3. Roger s Pressman, “Software Engineering”, Tata McGraw Hill.
4. Ian Sommer Villa, “Software Engineering”, Pearson Education.
5. Shari Lawrence, “Software Engineering Theory and Practice”, Pearson Education Asia.
6. Rajib Mall,” Fundamentals of Software Engineering”, PHI.
7. Carlo Ghezzi, Mehdi Jazayeri, “Fundamentals of Software Engineering”, PHI.

### **WEBSITES**

[www.asp.net](http://www.asp.net)

[www.csharp.com](http://www.csharp.com)

[www.msdn.com](http://www.msdn.com)