A PROJECT REPORT

ON

**“VOICE ASSISTANT”**

**DIPLOMA IN**

**COMPUTER ENGINEERING**

SUBMITTED TO

HARYANA STATE BOARD OF TECHNICAL EDUCATION, HARYANA

PANCHKULA

SUBMITTED BY

1. **RACHIT (202500800039)**
2. **AMAAN KHAN (202500800002)**
3. **NIGAM CHAUHAN ( 202500800032)**
4. **ATUL (202500800007)**
5. **CHIRAG (20500800009)**

GUIDED BY



**GOVT. POLYTECHNIC, NANAKPUR**

2023



GOVT. POLYTECHNIC, NANAKPUR

**CERTIFICATION**

This is to certify that the project report entitled “VOICE ASSISTANT ( )” was successfully completed by student(s) of name of 6th semester,

1. RACHIT (202500800039)
2. AMAAN KHAN (202500800002)
3. NIGAM CHAUHAN ( 202500800032)
4. ATUL (202500800007)
5. CHIRAG (202500800009)

in partial fulfilment of the requirements for the award of the **Computer Engineering** Department from the Department of **Computer Engineering** at Govt. Polytechnic, Nanakpur.



GOVT. POLYTECHNIC, NANAKPUR

ACKNOWLEDGMENT

This project is done as a major project titled **“Voice Assistant”**. , a part of course **“Computer Engineering”**.

We are really thankful to the HOD **Mahender Rozera**, **Computer Engineering** and my guide **Ramita Jindal** and **Lecturer**, Govt. Polytechnic, Nanakpur, for their invaluable guidance and assistance, without which the accomplishment of the task would have never been possible.

We also thank Name of the **Govt. Polytechnic, Nanakpur**, for giving us this opportunity to explore the real world and complete my project work in limited time frame.

We are also thankful to our all Teachers of Computer Engineering, for providing us relevant information, necessary clarifications, and great support.



GOVT. POLYTECHNIC, NANAKPUR

**ABSTRACT**

The Python-based Voice Assistant project aims to create a powerful voice-controlled assistant using the Python programming language. By leveraging natural language processing (NLP) and machine learning algorithms, the project develops an intelligent system capable of understanding and responding to user voice commands.

The voice assistant provides a wide range of functionalities, including managing tasks, retrieving web information, controlling smart home devices, delivering weather updates, setting reminders, and initiating voice calls or text messages. Utilizing Python's comprehensive libraries and frameworks, such as speech recognition, text-to-speech conversion, and deep learning, the voice assistant accurately interprets user commands and generates appropriate responses.

To enhance the user experience, the voice assistant incorporates advanced features like context awareness, sentiment analysis, and personalization. Context awareness allows the assistant to understand conversation context and provide relevant information or take appropriate actions. Sentiment analysis enables the assistant to gauge the user's emotional state, responding empathetically and adapting its behaviour accordingly. Personalization ensures the voice assistant learns from user preferences, adapting its responses over time for a tailored and customized experience.

Privacy and security are vital considerations, with the project adhering to industry-standard practices. User data is treated with utmost care, and privacy controls are in place to maintain confidentiality and avoid sharing information without explicit consent.

The Python-based Voice Assistant project holds significant potential across domains, from personal use to smart homes, offices, and customer support systems. Its development contributes to advancing natural language processing and voice recognition technologies, facilitating more intuitive and accessible human-computer interactions.



GOVT. POLYTECHNIC, NANAKPUR

|  |  |
| --- | --- |
|  | Page No. |
| Title page | I |
| Certificate | II |
| Acknowledgement | III |
| Abstract | IV |
|  |  |
| 1. Chapter-1. INTRODUCTION OF PYTHON | 1 |
|  |  |
| 1. Chapter-2. FRONT END CODEING    1. Modules Used in Program | 2-12 |
| 2.2 PIP Installation |
| 2.3 Working of Program |
| 2.4 User Input to run a Program  2.4 exe File CREATION |
| 1. Chapter-3. BACK-END CODING   3.1 GUI (Graphical User Interface) | 13 |
| 1. Chapter-4. Create A .exe file | 14 |
| Flow Chart | 15 |

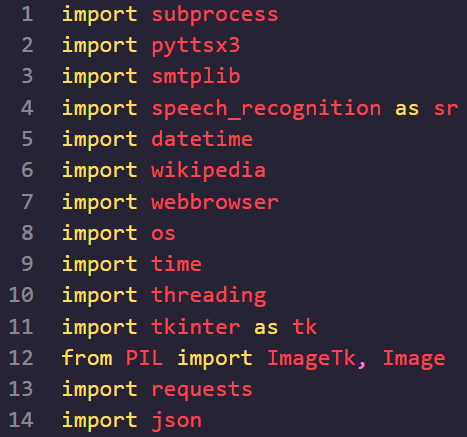
TABLE OF CONTENT

**Chapter-1. INTRODUCTION OF PYTHON**

* Python is a popular, high-level programming language that was developed by Guido van Rossum and initially released in 1991. It is widely recognized for its simplicity and readability, making it an ideal choice for beginners and experienced programmers alike.
* One of Python's notable features is its versatility, supporting multiple programming paradigms such as procedural, object-oriented, and functional programming. This flexibility allows developers to choose the most suitable approach for their projects.
* Python has gained immense popularity and is widely used across various domains including web development, data analysis, machine learning, scientific computing, and automation. Its extensive standard library provides a rich collection of modules and functions that simplify common tasks and reduce the need for additional code.
* The success of Python can be attributed to its strong community support. The Python community actively contributes to the development of libraries and frameworks that extend the language's capabilities. The Package Index (PyPI) serves as a repository for thousands of open-source libraries, making it convenient to find and integrate third-party solutions into Python projects.
* Python's syntax is known for its simplicity and readability. It employs indentation and whitespace to delimit code blocks, fostering visually appealing and consistently formatted code. Combined with Python's English-like keywords, this feature contributes to creating code that is easy to comprehend and maintain.
* Python is an interpreted language, allowing code to be executed line by line at runtime without the need for compilation. This facilitates rapid development and prototyping, as developers can quickly observe the results of their code without the additional step of compilation.
* In conclusion, Python is a versatile and powerful programming language renowned for its simplicity, readability, and extensive library ecosystem. Its broad range of applications continues to expand, making it an excellent choice for programmers of all levels of expertise.

**Chapter-2. Front End Coding**

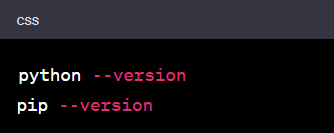
**2.1 Modules Used in Program**

****

1. **subprocess**: Allows executing system commands and interacting with the input/output/error pipes of spawned processes.
2. **pyttsx3**: Enables text-to-speech conversion, allowing you to generate speech from text.
3. **smtplib**: Provides an SMTP client session object for sending emails using the Simple Mail Transfer Protocol (SMTP).
4. **speech\_recognition**: Offers support for speech recognition, allowing you to capture and process audio input.
5. **datetime**: Provides classes for working with dates and times, allowing you to retrieve the current date and time.
6. **wikipedia**: Provides an interface to access articles and information from Wikipedia.
7. **webbrowser**: Offers a high-level interface for opening web-based documents and URLs in the user's default web browser.
8. **os**: Allows interaction with the operating system, providing functionality for tasks such as working with files and directories.
9. **time**: Provides various time-related functions, including pausing or delaying program execution for a specified duration.
10. **threading**: Enables the creation and management of multiple threads, allowing for concurrent execution of different parts of the program.
11. **tkinter**: Provides tools for creating graphical user interfaces (GUIs) using the Tk GUI toolkit.
12. **PIL**: The Python Imaging Library, used for working with images, including opening, manipulating, and saving various image file formats.
13. **ImageTk**: Provides a Tkinter-compatible image class for displaying images in Tkinter GUIs.
14. **Image**: Represents an image object in Python and provides functions for image manipulation and processing.
15. **requests**: Allows sending HTTP requests to interact with web services and retrieve data from APIs.
16. **json**: Provides functions for working with JSON (JavaScript Object Notation) data, including parsing JSON strings and converting them into Python objects, and vice versa.

**2.2 PIP Installation**

To install the necessary Python packages, you can use the pip package manager, which is the standard package manager for Python. Follow the instructions below to install the required packages:

1. Open a command-line interface (e.g., Terminal on macOS/Linux or Command Prompt on Windows).
2. Make sure you have Python and pip installed. You can check by running the following commands: 
3. If Python and pip are not installed, download and install Python from the official Python website ([https://www.python.org](https://www.python.org/)) and make sure to select the option to install pip during the installation process.
4. Once you have confirmed that pip is installed, you can proceed to install the required packages by running the following command:
5. This command installs the **pyttsx3**, **smtplib**, **SpeechRecognition**, **wikipedia**, **requests**, and **Pillow** packages.



1. Wait for the installation process to complete. Pip will automatically download and install the packages and their dependencies.

After executing the above steps, you should have the necessary packages installed and can proceed with using them in your Python code

* 1. **Working of Programing**
* **Listen and Speak Module**

1. Listen Module
2. The **listen()** function uses the SpeechRecognition library to capture audio from a microphone and convert it into text. Here's a simplified explanation:

Create a recognizer object: **r = sr.Recognizer()**

This creates an instance of the Recognizer class from the SpeechRecognition library. It will be used to recognize speech.

1. Access the microphone as the audio source: **with sr.Microphone() as source:**

This sets up a context manager to access the microphone and ensures it is properly released after use.

1. Capture audio input: **audio = r.listen(source)**

This records audio from the microphone using the **listen()** method of the Recognizer object. The captured audio is stored in the **audio** variable.

1. Recognize speech:

* Try block:
* Attempt to recognize the speech in the captured audio using the **recognize\_google()** method: **text = r.recognize\_google(audio)**
* The recognized text is stored in the **text** variable.
* Clear the contents of a text entry widget: **text\_entry.delete(0, tk.END)**
* Insert the recognized text into the text entry widget: **text\_entry.insert(tk.END, text)**
* Except sr.UnknownValueError block:
* Handle the case when speech recognition fails to understand or recognize the speech.
* Clear the text entry widget: **text\_entry.delete(0, tk.END)**
* Insert the message "Unable to recognize speech" into the text entry widget.
* Except sr.RequestError block:
* Handle the case when there is an error in making the speech recognition request.
* Clear the text entry widget: **text\_entry.delete(0, tk.END)**
* Insert the message "Speech recognition request error" into the text entry widget.
* This function captures audio from the microphone, converts it to text using the Google Web Speech API, and displays the recognized text in a Tkinter text entry widget (**text\_entry**). It also handles potential errors that may occur during the speech recognition process.

1. Speak Module

**speak()** function and initializes the pyttsx3 text-to-speech engine. Here's a breakdown of the code:

1. **speak()** function:

* **text = text\_entry.get()**: Retrieves the text content from a text entry widget (**text\_entry**) in a GUI and assigns it to the **text** variable.
* **engine = pyttsx3.init()**: Initializes the pyttsx3 text-to-speech engine using the default settings.
* **engine.say(text)**: Passes the **text** variable as input to the **say()** method of the pyttsx3 engine. This method adds the text to the speech queue.
* **engine.runAndWait()**: Initiates the speech synthesis process and waits for the speech to be spoken. This method blocks the program execution until all the text in the speech queue is spoken.

1. Engine initialization and voice selection:

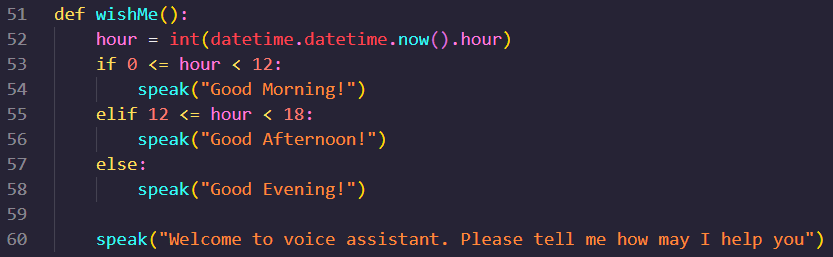
* **engine = pyttsx3.init('sapi5')**: Initializes the pyttsx3 engine using the 'sapi5' speech API provided by the Microsoft Speech Platform.
* **voices = engine.getProperty('voices')**: Retrieves the available voices supported by the text-to-speech engine.
* **engine.setProperty('voice', voices[1].id)**: Sets the voice property of the engine to the second voice in the **voices** list. This line selects a specific voice for speech synthesis. In this case, it selects the second voice (index 1) from the available voices list.

The purpose of this code is to define a function (**speak()**) that uses the pyttsx3 library to convert text into speech. It retrieves the text from a text entry widget in a GUI, initializes the pyttsx3 engine, adds the text to the speech queue, and then executes the speech synthesis process. The code also sets the voice property of the engine to use a specific voice from the available voices.

* **Wish me Module**

This module wishes the person according to the time of the day.

The **wishMe()** function in the code you provided is used to greet the user based on the current time and welcome them to the voice assistant. Here's an explanation of how the code works:



1. The **datetime.datetime.now().hour** retrieves the current hour of the day as an integer.

2. The **if-elif-else** statements are used to determine the appropriate greeting based on the current hour:

* If the hour is greater than or equal to 0 and less than 12, it is considered morning, and the voice assistant says "Good Morning!".
* If the hour is greater than or equal to 12 and less than 18, it is considered afternoon, and the voice assistant says "Good Afternoon!".
* If the hour is not in the previous ranges, it is considered evening, and the voice assistant says "Good Evening!".

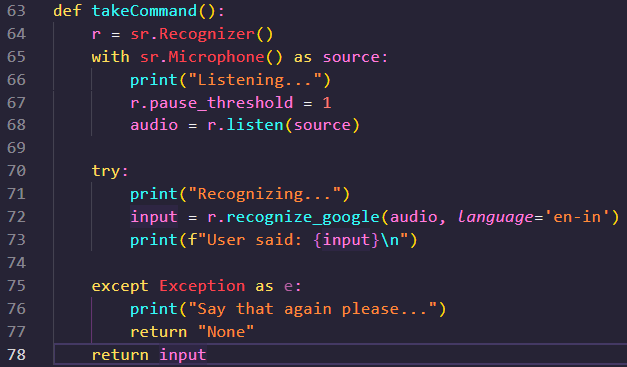
3. After determining the appropriate greeting, the voice assistant uses the **speak()** function (assuming it is defined elsewhere) to audibly speak the greeting.

4. Following the greeting, the voice assistant says "Welcome to voice assistant. Please tell me how may I help you".

Overall, the **wishMe()** function sets the tone and welcomes the user to the voice assistant based on the current time. It adds a personalized touch and prepares the user to interact with the voice assistant.

* **Take Command Module**

The **takeCommand()** function in the provided code is used to capture voice input from the user using the SpeechRecognition library (sr). Here's an explanation of how the code works:

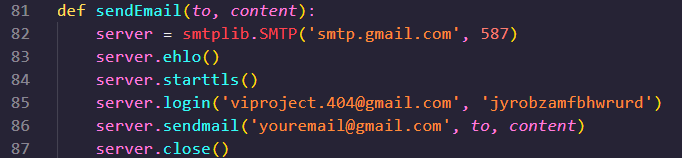


1. It creates a new instance of the Recognizer class from the sr module: **r = sr.Recognizer()**. This object will be used to recognize speech.
2. It sets up a context for audio capture using the user's microphone: **with sr.Microphone() as source**. The Microphone class represents the audio source (microphone in this case) and is used as the source of audio data.
3. It prints "Listening..." to indicate that it is ready to capture voice input.
4. It sets the pause threshold to 1 second: **r.pause\_threshold = 1**. This means that if there is a pause in speech longer than 1 second, the recording will stop and the captured audio will be processed.
5. It captures the audio input from the microphone using the **listen()** method of the Recognizer object: **audio = r.listen(source)**. This method listens to the audio input from the specified source (microphone) until it detects silence or the timeout is reached.
6. It prints "Recognizing..." to indicate that it is processing the captured audio.
7. It uses the **recognize\_google()** method of the Recognizer object to convert the captured audio into text: **input = r.recognize\_google(audio, language='en-in')**. The **recognize\_google()** method sends the audio data to the Google Speech Recognition service and attempts to convert it into text. The recognized text is assigned to the **INPUT** variable.
8. If the speech recognition is successful and the text is recognized, it prints the recognized query: **print(f"User said: {input}\n").**
9. If there is an exception during speech recognition (e.g., speech not recognized), it prints "Say that again please..." and returns the string "None".
10. If speech recognition is successful, it returns the recognized query as a string.

The **takeCommand()** function is designed to be used as part of a voice assistant or speech recognition system. It captures the user's voice input, processes it using speech recognition, and returns the recognized text for further processing or interaction with the voice assistant.

* **Send Email Module**

The `sendEmail` function provided is a Python function that allows you to send an email through a Gmail account using the Simple Mail Transfer Protocol (SMTP). Here's a breakdown of how the function works:



1. \*\*SMTP Server Setup\*\*:

- The `smtplib.SMTP` class is used to create an instance of an SMTP server, connecting to the Gmail SMTP server at `'smtp.gmail.com'` on port `587`. This establishes the connection between your code and the Gmail server.

- The `ehlo()` method is called to greet the server and introduce the client.

- The `starttls()` method is invoked to initiate a secure TLS (Transport Layer Security) connection. This step enables encryption for the subsequent communication.

2. \*\*Authentication\*\*:

- The `login` method is used to authenticate the user by providing the Gmail email address (`viproject.404@gmail.com`) and the corresponding password (`jyrobzamfbhwrurd`). This step ensures that the email is sent from a valid and authorized account.

3. \*\*Sending the Email\*\*:

- The `sendmail` method is used to send the email.

- The sender's email address is specified as `'youremail@gmail.com'`.

- The recipient's email address is passed through the `to` parameter of the `sendEmail` function.

- The content of the email is passed through the `content` parameter of the `sendEmail` function.

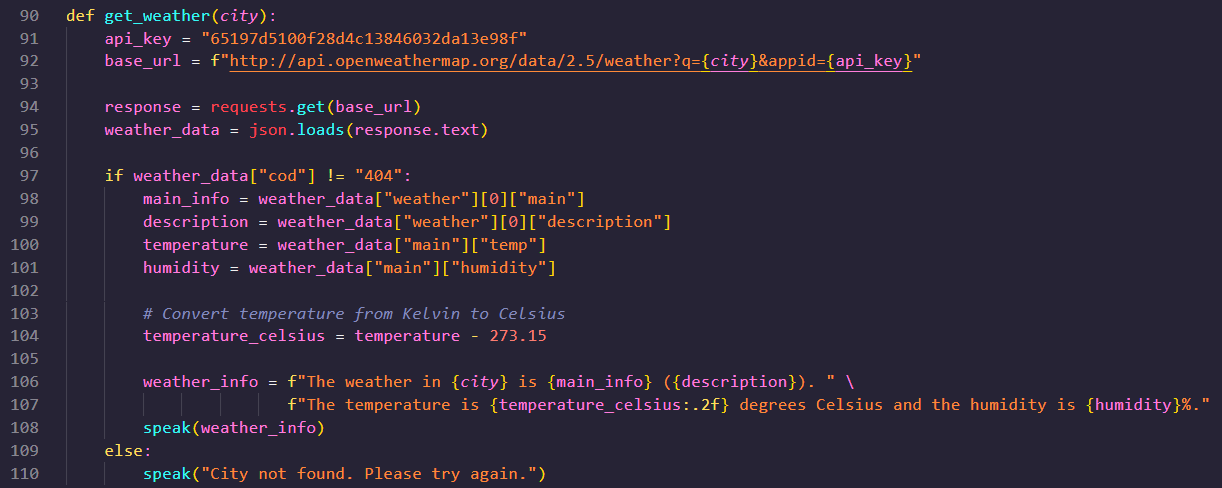
4. \*\*Closing the Connection\*\*:

- After the email is sent, the `close()` method is called to terminate the connection with the SMTP server. This ensures that resources are released and the connection is properly closed.

To use this function, you need to provide the recipient's email address (`to`) and the content of the email (`content`) when calling the `sendEmail` function. Additionally, you should ensure that you have provided valid Gmail credentials (`viproject.404@gmail.com` and its corresponding password) and have allowed access for less secure apps or generated an application-specific password. Without proper authentication, the email sending process will fail.

* **Weather Module**

The **get\_weather(city)** function in the provided code is used to fetch weather information for a given city using the OpenWeatherMap API. Here's an explanation of how the code works:

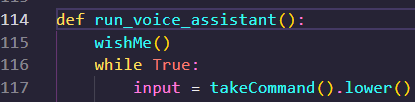


1. It defines the API key required to access the OpenWeatherMap API: **api\_key = "65197d5100f28d4c13846032da13e98f"**. Make sure to replace this API key with your own valid API key.
2. It constructs the base URL for the API request using the provided city: **base\_url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={api\_key}"**.
3. It sends an HTTP GET request to the OpenWeatherMap API using the constructed URL: **response = requests.get(base\_url)**.
4. It parses the response data as JSON: **weather\_data = json.loads(response.text)**.
5. It checks if the API response contains valid weather data by verifying the **"cod"** value. If the value is not **"404"**, it means the city is found and weather data is available.
6. It extracts relevant weather information from the API response, such as the main weather condition, description, temperature, and humidity.
7. It converts the temperature from Kelvin to Celsius by subtracting 273.15: **temperature\_celsius = temperature - 273.15**.
8. It constructs a string containing the weather information for the given city.
9. It uses the **speak()** function (not shown in the provided code) to speak out the weather information.
10. If the API response indicates that the city was not found (**"cod"** value is **"404"**), it uses the **speak()** function to inform the user that the city was not found.

The **get\_weather(city)** function can be called with a city name as an argument to fetch the current weather information for that city using the OpenWeatherMap API. The retrieved weather information is then spoken out using the **speak()** function.

* **Run Voice Assistant Modules**

The **run\_voice\_assistant()** function is the main function that runs the voice assistant. Here's an explanation of how it works:



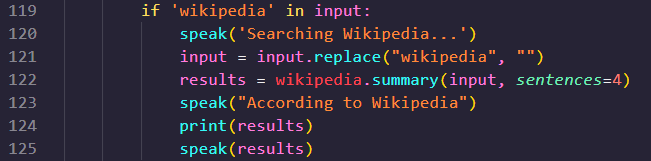
1. It starts by calling the **wishMe()** function, which greets the user based on the current time and welcomes them to the voice assistant.
2. It enters an infinite loop using **while True:** to continuously listen for user commands.
3. Inside the loop, it calls the **takeCommand()** function to capture the user's voice command and converts it to text. The returned command is converted to lowercase using the **lower()** method and assigned to the **query** variable. The **takeCommand()** function uses the SpeechRecognition library to listen to the user's voice input through the microphone and convert it to text.
4. The **query** variable now contains the user's command in lowercase format, which can be used to perform different actions or trigger specific functionalities of the voice assistant. You can add conditional statements to check the value of **query** and determine what action to take based on the user's command.
5. The loop continues to listen for user commands until the program is interrupted or terminated.

By continuously capturing user commands and processing them inside the loop, the voice assistant can provide real-time responses and interact with the user based on their voice input. You can extend the functionality of the voice assistant by adding more conditional statements and incorporating various features according to your requirements.

* **INPUT TO RUN A PROGRAM**

1. WIKIPEDIA INPUT

This block of code handles the case when the user mentions "wikipedia" in their query. Here's how it works:



1. The condition **'wikipedia' in query** checks if the word "wikipedia" is present in the user's query.
2. If the condition is true, the voice assistant speaks "Searching Wikipedia..." using the **speak()** function to provide feedback to the user.
3. The line **query = query.replace("wikipedia", "")** removes the word "wikipedia" from the user's query, so that only the specific search query remains. For example, if the user says "search wikipedia for Albert Einstein", the resulting query will be "Albert Einstein".
4. The line **results = wikipedia.summary(query, sentences=4)** uses the Wikipedia API and the **summary()** function from the **wikipedia** library to fetch a summary of the topic mentioned in the query. It retrieves 4 sentences from the Wikipedia page related to the query.
5. The voice assistant speaks "According to Wikipedia" using the **speak()** function to provide an introductory phrase.
6. The line **print(results)** prints the fetched summary of the topic to the console.

Finally, the voice assistant speaks the fetched summary using the **speak()** function, allowing the user to hear the information.

1. OPEN YOUTUBE



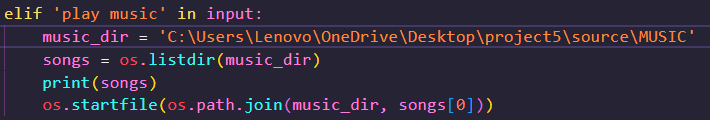
1. OPEN GOOGLE



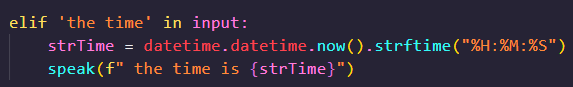
1. OPEN STACKOVERFLOW



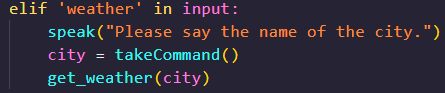
1. PLAY MUSIC



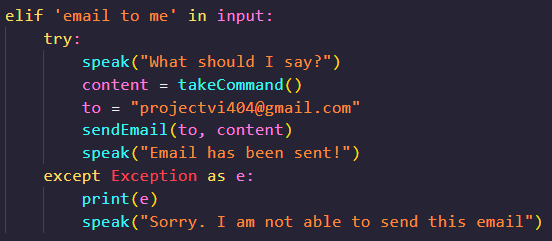
1. THE TIME



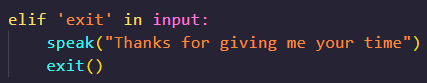
1. WEATHER



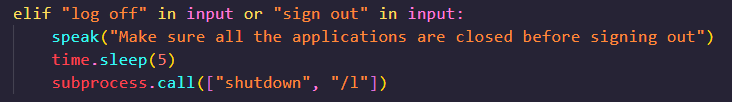
1. EMAIL



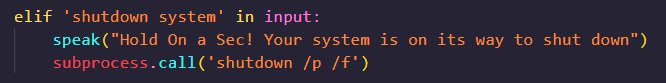
1. EXIT



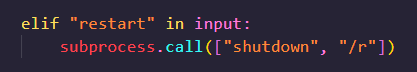
1. LOG OFF



1. SHUTDOWN SYSTEM



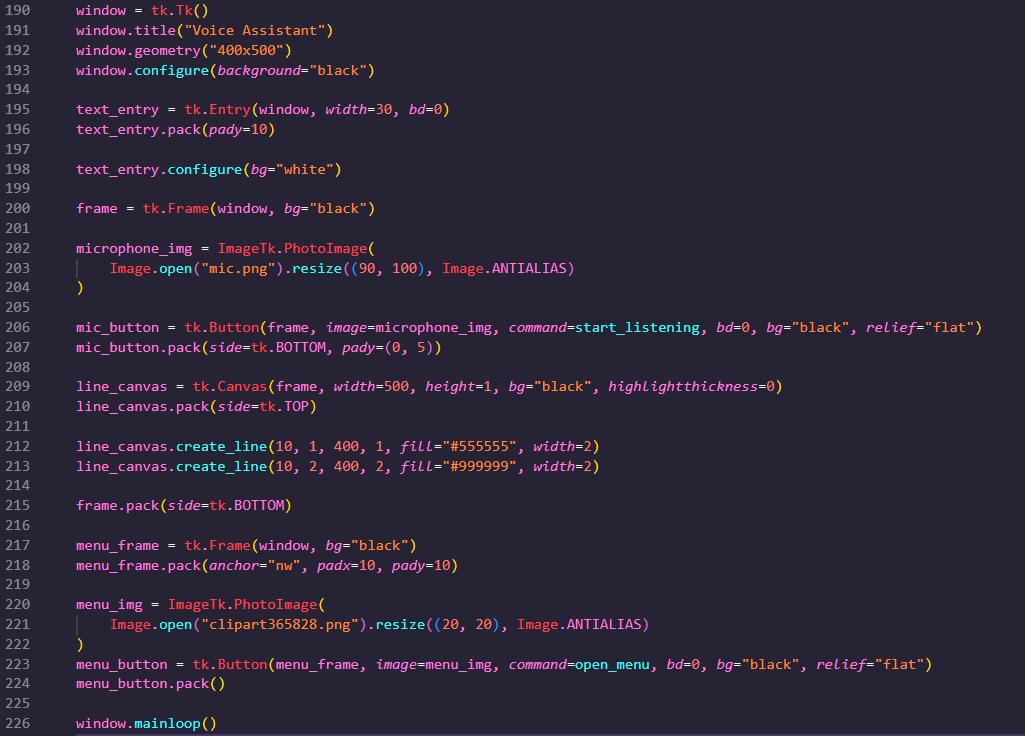
1. RESTART



**Chapter-3. Front End Coding**

3.1 GUI(Graphical User Interface)

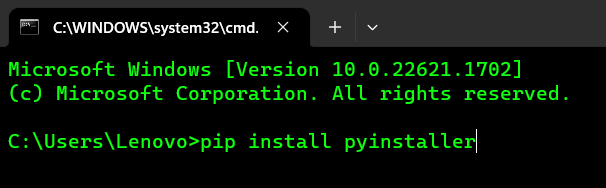
The code you provided is an example of creating a graphical user interface (GUI) using the Tkinter library in Python. Here's an explanation of how it works:



1. Create a new window using **tk.Tk()** and set its title to "Voice Assistant".
2. Set the window size to 400x500 pixels and configure its background color to black.
3. Create a text entry widget using **tk.Entry()** and set its width to 30 characters with no border (**bd=0**).
4. Pack the text entry widget into the window and add some vertical padding.
5. Modify the background color of the text entry widget to white.
6. Create a frame to hold the microphone button and line using **tk.Frame()** and set its background color to black.
7. Load the microphone image using **Image.open()** from a file named "mic.png" and resize it to 90x100 pixels using **Image.resize()**. Convert the image to **ImageTk.PhotoImage()** format.
8. Create a button for the microphone using **tk.Button()**, set its image to the resized microphone image, and specify the command to be executed when clicked as **start\_listening()**.
9. Pack the microphone button into the frame, align it to the bottom, and add some vertical padding.
10. Create a canvas using **tk.Canvas()** within the frame, set its width and height to 500 and 1 respectively, and specify its background color and highlight thickness.
11. Pack the canvas into the frame, align it to the top.
12. Draw two lines on the canvas using the **create\_line()** method of the canvas. These lines are used to create a visual separator.
13. Pack the frame containing the microphone button and line into the window, aligning it to the bottom.
14. Create another frame to hold the menu button and set its background color to black.
15. Load the menu icon using **Image.open()** from a file named "clipart365828.png" and resize it to 20x20 pixels using **Image.resize()**. Convert the image to **ImageTk.PhotoImage()** format.
16. Create a button for the menu using **tk.Button()**, set its image to the resized menu icon, and specify the command to be executed when clicked as **open\_menu()**.
17. Pack the menu button into the menu frame.
18. Pack the menu frame into the window, aligning it to the northwest and adding some padding.
19. Start the main loop of the GUI using **window.mainloop()**. This loop continuously listens for user events and updates the GUI accordingly.

**Chapter-4. Create A .exe File**

1. Install the required tools: Before creating an executable, you need to install the necessary tools. One popular tool is PyInstaller, which bundles Python applications into standalone executables. You can install PyInstaller using pip by running the following command in your terminal or command prompt:



1. Write your Python script: Create your Python script or ensure you have an existing script that you want to convert into an executable. Make sure your script is functioning as expected and does not have any dependencies that are not included in the standard Python library.
2. Generate the executable: Once you have installed PyInstaller and prepared your Python script, you can generate the executable by running the following command in your terminal or command prompt:

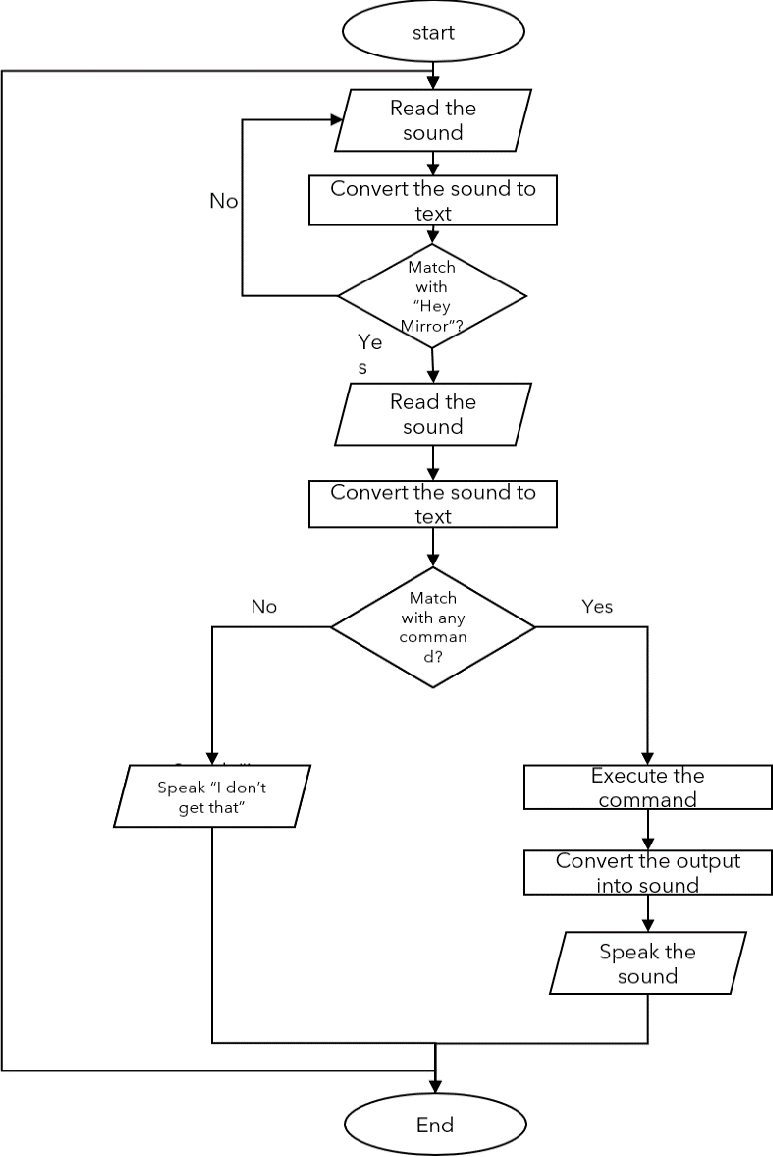


Replace **your\_script.py** with the actual name of your Python script.

PyInstaller will analyze your script, collect the required dependencies, and create a bundled executable file in a new **dist** directory.

1. Test the executable: After generating the executable, you can test it to ensure it runs properly. Navigate to the **dist** directory and run the executable file corresponding to your Python script.
2. Distribute the executable: Once you have confirmed that the executable works as expected, you can distribute it to others. You can share the entire **dist** directory, which contains the standalone executable and any necessary files or dependencies.

Keep in mind that PyInstaller attempts to bundle all the required dependencies, but there might be certain cases where additional configuration is needed, especially if your script depends on non-standard libraries or external resources. Refer to the PyInstaller documentation for advanced usage and troubleshooting.





****

