

# ChatGPTChatGPT



```
import speech_recognition as sr
import pyttsx3
import webbrowser
import os
import subprocess
from datetime import datetime
import tkinter as tk
from tkinter import scrolledtext
import threading
import time
import re # For regex operations

# Initialize recognizer and TTS engine
recognizer = sr.Recognizer()
tts_engine = pyttsx3.init()

# List of available commands
AVAILABLE_COMMANDS = [
    "time - Say the current time",
    "open browser - Open the web browser",
    "open calculator - Open the calculator",
    "open youtube - Open YouTube",
    "search google <query> - Search Google",
    "shutdown - Shutdown the system",
    "take a note - Save a voice note",
    "set timer for <duration> - Set a timer",
    "perform <math_expression> - Perform a math operation",
    "exit - Exit the application"
]

# Global flag to control listening
listening = False

def speak(text):
    """Convert text to speech."""
    tts_engine.say(text)
    tts_engine.runAndWait()

def take_command():
    """Listen for a voice command and return it as text."""
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        display_message("Listening...")
        try:
            audio = recognizer.listen(source, timeout=5)
```

```

        command = recognizer.recognize_google(audio).lower()
        display_message(f"You said: {command}")
        return command
    except sr.UnknownValueError:
        speak("Sorry, I didn't catch that.")
        display_message("Sorry, I didn't catch that.")
        return ""
    except sr.RequestError:
        speak("Could not connect to the speech recognition service.")
        display_message("Could not connect to the speech recognition service.")
        return ""
    except sr.WaitTimeoutError:
        display_message("Listening timed out.")
        return ""

def run_command(command):
    """Perform actions based on recognized commands."""
    if "time" in command:
        now = datetime.now().strftime("%H:%M")
        speak(f"The time is {now}.")
        display_message(f"The time is {now}.")

    elif "open browser" in command:
        speak("Opening browser.")
        display_message("Opening browser...")
        webbrowser.open("https://www.google.com")

    elif "open calculator" in command:
        speak("Opening calculator.")
        display_message("Opening calculator...")
        if os.name == "nt":
            subprocess.Popen("calc.exe")
        elif os.name == "posix":
            subprocess.Popen(["gnome-calculator"])

    elif "open youtube" in command:
        speak("Opening YouTube.")
        display_message("Opening YouTube...")
        webbrowser.open("https://www.youtube.com")

    elif "search google" in command:
        query = command.replace("search google", "").strip()
        speak(f"Searching Google for {query}.")
        display_message(f"Searching Google for: {query}...")
        webbrowser.open(f"https://www.google.com/search?q={query}")

    elif "shutdown" in command:
        speak("Shutting down the system.")
        display_message("Shutting down the system...")
        if os.name == "nt":
            os.system("shutdown /s /t 1")

```

```

elif os.name == "posix":
    os.system("sudo shutdown now")

elif "take a note" in command:
    speak("What would you like to note down?")
    note = take_command()
    if note:
        with open("notes.txt", "a") as file:
            file.write(f"{note}\n")
        speak("Your note has been saved.")
        display_message("Your note has been saved.")

elif "set timer for" in command:
    duration = re.search(r"(\d+)\s*(seconds|minutes)", command)
    if duration:
        time_value = int(duration.group(1))
        unit = duration.group(2)
        total_seconds = time_value if unit == "seconds" else time_value * 60

        display_message(f"Setting a timer for {time_value} {unit}.")
        speak(f"Timer set for {time_value} {unit}.")
        time.sleep(total_seconds)
        speak("Time's up!")
        display_message("Time's up!")
    else:
        speak("Please specify a valid duration.")
        display_message("Please specify a valid duration.")

elif "perform" in command:
    expression = command.replace("perform", "").strip()
    try:
        result = eval(expression)
        speak(f"The result is {result}.")
        display_message(f"The result of {expression} is {result}.")
    except Exception as e:
        speak("There was an error performing the calculation.")
        display_message("There was an error performing the calculation.")

elif "exit" in command:
    speak("Goodbye!")
    display_message("Goodbye!")
    root.quit() # Exit the GUI

else:
    speak("I'm not sure how to do that.")
    display_message("I'm not sure how to do that.")

def display_message(message):
    """Display messages and commands in the text area."""
    text_area.insert(tk.END, f"{message}\n")
    text_area.yview(tk.END) # Auto-scroll to the bottom

```

```
def display_available_commands():
    """Display all available commands in the command list area."""
    command_list_area.insert(tk.END, "Available Commands:\n")
    for cmd in AVAILABLE_COMMANDS:
        command_list_area.insert(tk.END, f"- {cmd}\n")
    command_list_area.config(state=tk.DISABLED) # Make it read-only

def continuous_listen():
    """Continuously listen for commands with pauses between executions."""
    global listening
    while True:
        if listening:
            command = take_command()
            if command:
                run_command(command)
                time.sleep(2) # Pause for 2 seconds after each command

def toggle_listening():
    """Start or stop the listening process."""
    global listening
    listening = not listening
    if listening:
        speak("Listening started.")
        display_message("Listening started...")
        start_listening_button.config(text="Stop Listening")
    else:
        speak("Listening stopped.")
        display_message("Listening stopped...")
        start_listening_button.config(text="Start Listening")

def start_listening_thread():
    """Start the listening loop in a separate thread."""
    listening_thread = threading.Thread(target=continuous_listen)
    listening_thread.daemon = True # Ensure thread exits when the main program does
    listening_thread.start()

# Set up the Tkinter GUI
root = tk.Tk()
root.title("Real-Time Voice Assistant")
root.geometry("600x500")

# Text area to display commands and messages
text_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=10, width=60)
text_area.pack(pady=10)

# Text area to display available commands (read-only)
command_list_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=6, width=60)
command_list_area.pack(pady=10)

# Button to start/stop listening
```

```
start_listening_button = tk.Button(root, text="Start Listening", command=toggle_listening, font=
("Arial", 14))
start_listening_button.pack(pady=10)

# Display available commands on startup
display_available_commands()

# Start the assistant with a greeting
speak("How can I help you?")
display_message("How can I help you?")

# Start the continuous listening thread
start_listening_thread()

# Run the GUI main loop
root.mainloop()

explain this code line by line
```



Certainly! Let's break down the provided Python code line by line to understand how it works.

## Importing Libraries

python

```
import speech_recognition as sr
import pyttsx3
import webbrowser
import os
import subprocess
from datetime import datetime
import tkinter as tk
from tkinter import scrolledtext
import threading
import time
import re # For regex operations
```

1. `import speech_recognition as sr`: Imports the SpeechRecognition library, allowing the program to recognize spoken words.
2. `import pyttsx3`: Imports the Text-to-Speech (TTS) engine, enabling the application to convert text to spoken words.
3. `import webbrowser`: Allows the program to open web browsers and navigate to URLs.
4. `import os`: Provides a way to interact with the operating system, such as opening applications or shutting down the computer.
5. `import subprocess`: Enables spawning new processes, connecting to their input/output/error pipes, and obtaining their return codes.
6. `from datetime import datetime`: Imports the `datetime` class, which is used to retrieve and format the current date and time.
7. `import tkinter as tk`: Imports the Tkinter library for creating a graphical user interface (GUI).

8. `from tkinter import scrolledtext`: Imports the `ScrolledText` widget from Tkinter for displaying scrollable text areas.
9. `import threading`: Allows the program to run multiple threads, enabling simultaneous operations like listening for voice commands while processing.
10. `import time`: Provides time-related functions, such as sleeping for a given duration.
11. `import re`: Imports the regular expressions module, which allows for searching and manipulating strings based on patterns.

## Initializing Components

python

```
# Initialize recognizer and TTS engine
recognizer = sr.Recognizer()
tts_engine = pyttsx3.init()
```

12. `recognizer = sr.Recognizer()`: Creates an instance of the speech recognizer, which will handle converting speech to text.
13. `tts_engine = pyttsx3.init()`: Initializes the TTS engine, preparing it for converting text to speech.

## List of Commands

python

```
# List of available commands
AVAILABLE_COMMANDS = [
    "time - Say the current time",
    "open browser - Open the web browser",
    "open calculator - Open the calculator",
    "open youtube - Open YouTube",
    "search google <query> - Search Google",
    "shutdown - Shutdown the system",
    "take a note - Save a voice note",
    "set timer for <duration> - Set a timer",
    "perform <math_expression> - Perform a math operation",
    "exit - Exit the application"
]
```

14. `AVAILABLE_COMMANDS`: Defines a list of commands that the assistant can recognize and execute. Each command has a brief description.

## Global Flag

python

```
# Global flag to control listening
listening = False
```

15. `listening = False`: Initializes a global variable to control whether the assistant is actively listening for commands.

## Text-to-Speech Function

python

```
def speak(text):
    """Convert text to speech."""
    tts_engine.say(text)
    tts_engine.runAndWait()
```

16. `def speak(text):`: Defines a function that takes text as input and converts it to speech.
17. `tts_engine.say(text)`: Queues the text for speech output.
18. `tts_engine.runAndWait()`: Processes the speech commands and waits until all speech has been spoken.

## Voice Command Function

python

```
def take_command():
    """Listen for a voice command and return it as text."""
    with sr.Microphone() as source:
        recognizer.adjust_for_ambient_noise(source)
        display_message("Listening...")
        try:
            audio = recognizer.listen(source, timeout=5)
            command = recognizer.recognize_google(audio).lower()
            display_message(f"You said: {command}")
            return command
        except sr.UnknownValueError:
            speak("Sorry, I didn't catch that.")
            display_message("Sorry, I didn't catch that.")
            return ""
        except sr.RequestError:
            speak("Could not connect to the speech recognition service.")
            display_message("Could not connect to the speech recognition service.")
            return ""
        except sr.WaitTimeoutError:
            display_message("Listening timed out.")
            return ""
```

19. `def take_command():`: Defines a function to listen for voice commands and return the recognized text.
20. `with sr.Microphone() as source:`: Opens the microphone as the audio source.
21. `recognizer.adjust_for_ambient_noise(source)`: Adjusts the recognizer sensitivity to ambient noise levels.
22. `display_message("Listening...")`: Displays a message in the GUI indicating the assistant is listening.
23. `audio = recognizer.listen(source, timeout=5)`: Listens for audio input for a maximum of 5 seconds.
24. `command = recognizer.recognize_google(audio).lower()`: Uses Google's speech recognition service to convert the audio to text and converts it to lowercase.
25. `display_message(f"You said: {command}")`: Displays the recognized command in the GUI.
26. `return command`: Returns the recognized command as a string.
27. **Exception Handling:**

- `sr.UnknownValueError`: Catches errors when the speech is not recognized and informs the user.

- `sr.RequestError`: Catches connection errors to the speech recognition service.
- `sr.WaitTimeoutError`: Catches timeout errors if no speech is detected.

## Command Execution Function

python

```
def run_command(command):
    """Perform actions based on recognized commands."""
```

28. `def run_command(command):`: Defines a function that performs different actions based on the recognized voice command.

python

```
if "time" in command:
    now = datetime.now().strftime("%H:%M")
    speak(f"The time is {now}.")
    display_message(f"The time is {now}.")
```

29. `if "time" in command:`: Checks if the command includes "time".
30. `now = datetime.now().strftime("%H:%M")`: Gets the current time in HH format.
31. `speak(f"The time is {now}.")`: Speaks out the current time.
32. `display_message(f"The time is {now}.")`: Displays the current time in the GUI.

python

```
elif "open browser" in command:
    speak("Opening browser.")
    display_message("Opening browser...")
    webbrowser.open("https://www.google.com")
```

33. `elif "open browser" in command:`: Checks if the command includes "open browser".
34. `speak("Opening browser.")`: Informs the user that the browser is opening.
35. `display_message("Opening browser...")`: Updates the GUI to reflect the action.
36. `webbrowser.open("https://www.google.com")`: Opens the default web browser to Google.

python

```
elif "open calculator" in command:
    speak("Opening calculator.")
    display_message("Opening calculator...")
    if os.name == "nt":
        subprocess.Popen("calc.exe")
    elif os.name == "posix":
        subprocess.Popen(["gnome-calculator"])
```

37. `elif "open calculator" in command:`: Checks if the command includes "open calculator".
38. `speak("Opening calculator.")`: Informs the user that the calculator is opening.



- 39. `display_message("Opening calculator...")`: Updates the GUI.
- 40. `if os.name == "nt":`: Checks if the operating system is Windows.
- 41. `subprocess.Popen("calc.exe")`: Opens the calculator on Windows.
- 42. `elif os.name == "posix":`: Checks if the operating system is Unix-like (e.g., Linux).
- 43. `subprocess.Popen(["gnome-calculator"])`: Opens the GNOME calculator on Unix-like systems.

python

```
elif "open youtube" in command:
    speak("Opening YouTube.")
    display_message("Opening YouTube...")
    webbrowser.open("https://www.youtube.com")
```

- 44. `elif "open youtube" in command:`: Checks if the command includes "open YouTube".
- 45. `speak("Opening YouTube.")`: Informs the user that YouTube is opening.
- 46. `display_message("Opening YouTube...")`: Updates the GUI.
- 47. `webbrowser.open("https://www.youtube.com")`: Opens YouTube in the web browser.

python

```
elif "search google" in command:
    query = command.replace("search google", "").strip()
    speak(f"Searching Google for {query}.")
    display_message(f"Searching Google for: {query}...")
    webbrowser.open(f"https://www.google.com/search?q={query}")
```

- 48. `elif "search google" in command:`: Checks if the command includes "search Google".
- 49. `query = command.replace("search google", "").strip()`: Extracts the search query from the command.
- 50. `speak(f"Searching Google for {query}.")`: Informs the user of the search query.
- 51. `display_message(f"Searching Google for: {query}...")`: Updates the GUI with the search action.
- 52. `webbrowser.open(f"https://www.google.com/search?q={query}")`: Opens Google search results for the query.

python

```
elif "shutdown" in command:
    speak("Shutting down the system.")
    display_message("Shutting down the system...")
    if os.name == "nt":
        os.system("shutdown /s /t 1")
    elif os.name == "posix":
        os.system("sudo shutdown now")
```

- 53. `elif "shutdown" in command:`: Checks if the command includes "shutdown".
- 54. `speak("Shutting down the system.")`: Informs the user that the system is shutting down.
- 55. `display_message("Shutting down the system...")`: Updates the GUI.
- 56. `if os.name == "nt":`: Checks if the operating system is Windows.
- 57. `os.system("shutdown /s /t 1")`: Executes the shutdown command on Windows.
- 58. `elif os.name == "posix":`: Checks if the operating system is Unix-like.

59. ``os.system("sudo shutdown now")``: Executes the shutdown command on Unix-like systems.

python

```
elif "take a note" in command:
    speak("What would you like to note down?")
    note = take_command()
    if note:
        with open("notes.txt", "a") as file:
            file.write(f"{note}\n")
        speak("Your note has been saved.")
        display_message("Your note has been saved.")
```

- 60. ``elif "take a note" in command:``: Checks if the command includes "take a note".
- 61. ``speak("What would you like to note down?")``: Prompts the user to provide a note.
- 62. ``note = take_command()``: Calls ``take_command()`` to listen for the note input.
- 63. ``if note:``: Checks if the note was successfully captured.
- 64. ``with open("notes.txt", "a") as file:``: Opens (or creates) a text file named ``notes.txt`` in append mode.
- 65. ``file.write(f"{note}\n")``: Writes the note to the file.
- 66. ``speak("Your note has been saved.")``: Confirms to the user that the note was saved.
- 67. ``display_message("Your note has been saved.")``: Updates the GUI with confirmation.

python

```
elif "set timer for" in command:
    duration = re.search(r"(\d+)\s*(seconds|minutes)", command)
    if duration:
        time_value = int(duration.group(1))
        unit = duration.group(2)
        total_seconds = time_value if unit == "seconds" else time_value * 60

        display_message(f"Setting a timer for {time_value} {unit}.")
        speak(f"Timer set for {time_value} {unit}.")
        time.sleep(total_seconds)
        speak("Time's up!")
        display_message("Time's up!")
    else:
        speak("Please specify a valid duration.")
        display_message("Please specify a valid duration.")
```

- 68. ``elif "set timer for" in command:``: Checks if the command includes "set timer for".
- 69. ``duration = re.search(r"(\d+)\s*(seconds|minutes)", command)``: Uses regex to search for a numeric duration followed by either "seconds" or "minutes".
- 70. ``if duration:``: Checks if a valid duration was found.
- 71. ``time_value = int(duration.group(1))``: Extracts the numeric value of the duration.
- 72. ``unit = duration.group(2)``: Extracts whether the duration is in seconds or minutes.
- 73. ``total_seconds = time_value if unit == "seconds" else time_value * 60``: Converts the duration to total seconds.
- 74. ``display_message(f"Setting a timer for {time_value} {unit}.")``: Updates the GUI with timer details.
- 75. ``speak(f"Timer set for {time_value} {unit}.")``: Informs the user about the timer.
- 76. ``time.sleep(total_seconds)``: Pauses the execution for the specified duration.
- 77. ``speak("Time's up!")``: Notifies the user that the timer has finished.
- 78. ``display_message("Time's up!")``: Updates the GUI with the timer completion message.

- 79. ``else:``: Handles cases where the duration is not specified correctly.
- 80. ``speak("Please specify a valid duration.")``: Informs the user to provide a valid duration.
- 81. ``display_message("Please specify a valid duration.")``: Updates the GUI.

python

```
elif "perform" in command:
    expression = command.replace("perform", "").strip()
    try:
        result = eval(expression)
        speak(f"The result is {result}.")
        display_message(f"The result of {expression} is {result}.")
    except Exception as e:
        speak("There was an error performing the calculation.")
        display_message("There was an error performing the calculation.")
```

- 82. ``elif "perform" in command:``: Checks if the command includes "perform".
- 83. ``expression = command.replace("perform", "").strip()``: Extracts the mathematical expression from the command.
- 84. ``try:``: Starts a try block to handle potential errors.
- 85. ``result = eval(expression)``: Evaluates the mathematical expression.
- 86. ``speak(f"The result is {result}.")``: Informs the user of the result.
- 87. ``display_message(f"The result of {expression} is {result}.")``: Displays the result in the GUI.
- 88. ``except Exception as e:``: Catches any errors during evaluation.
- 89. ``speak("There was an error performing the calculation.")``: Informs the user of the error.
- 90. ``display_message("There was an error performing the calculation.")``: Updates the GUI.

python

```
elif "exit" in command:
    speak("Goodbye!")
    display_message("Goodbye!")
    root.quit() # Exit the GUI
```

- 91. ``elif "exit" in command:``: Checks if the command includes "exit".
- 92. ``speak("Goodbye!")``: Informs the user that the application is closing.
- 93. ``display_message("Goodbye!")``: Updates the GUI.
- 94. ``root.quit()``: Exits the Tkinter main loop and closes the application.

python

```
else:
    speak("I'm not sure how to do that.")
    display_message("I'm not sure how to do that.")
```

- 95. ``else:``: Handles any unrecognized commands.
- 96. ``speak("I'm not sure how to do that.")``: Informs the user that the command was not recognized.
- 97. ``display_message("I'm not sure how to do that.")``: Updates the GUI.

## Displaying Messages

python

```
def display_message(message):
    """Display messages and commands in the text area."""
    text_area.insert(tk.END, f"{message}\n")
    text_area.yview(tk.END) # Auto-scroll to the bottom
```

- 98. `def display_message(message):`: Defines a function to display messages in the GUI.
- 99. `text_area.insert(tk.END, f"{message}\n")`: Inserts the message at the end of the text area.
- 100. `text_area.yview(tk.END)`: Auto-scrolls the text area to the bottom to show the latest message.

## Displaying Available Commands

python

```
def display_available_commands():
    """Display all available commands in the command list area."""
    command_list_area.insert(tk.END, "Available Commands:\n")
    for cmd in AVAILABLE_COMMANDS:
        command_list_area.insert(tk.END, f"- {cmd}\n")
    command_list_area.config(state=tk.DISABLED) # Make it read-only
```

- 101. `def display_available_commands():`: Defines a function to show available commands in the GUI.
- 102. `command_list_area.insert(tk.END, "Available Commands:\n")`: Inserts a header for the commands.
- 103. `for cmd in AVAILABLE_COMMANDS:`: Loops through each command in the `AVAILABLE_COMMANDS` list.
- 104. `command_list_area.insert(tk.END, f"- {cmd}\n")`: Inserts each command into the command list area.
- 105. `command_list_area.config(state=tk.DISABLED)`: Sets the command list area to read-only mode.

## Continuous Listening Function

python

```
def continuous_listen():
    """Continuously listen for commands with pauses between executions."""
    global listening
    while True:
        if listening:
            command = take_command()
            if command:
                run_command(command)
                time.sleep(2) # Pause for 2 seconds after each command
```

- 106. `def continuous_listen():`: Defines a function to continuously listen for commands while the listening flag is set.
- 107. `global listening`: Declares the `listening` variable to be global.
- 108. `while True:`: Creates an infinite loop for continuous listening.
- 109. `if listening:`: Checks if listening is enabled.
- 110. `command = take_command()`: Calls `take_command()` to get a command from the user.
- 111. `if command:`: Checks if a command was received.
- 112. `run_command(command)`: Executes the recognized command.

113. `time.sleep(2)`: Pauses execution for 2 seconds before the next command.

## Toggle Listening

python

```
def toggle_listening():
    """Start or stop the listening process."""
    global listening
    listening = not listening
    if listening:
        speak("Listening started.")
        display_message("Listening started...")
        start_listening_button.config(text="Stop Listening")
    else:
        speak("Listening stopped.")
        display_message("Listening stopped...")
        start_listening_button.config(text="Start Listening")
```

114. `def toggle_listening()`: Defines a function to start or stop the listening process.

115. `global listening`: Declares `listening` to be a global variable.

116. `listening = not listening`: Toggles the value of `listening`.

117. `if listening:`: Checks if listening has started.

118. `speak("Listening started.")`: Informs the user that listening has begun.

119. `display_message("Listening started...")`: Updates the GUI.

120. `start_listening_button.config(text="Stop Listening")`: Changes the button text to "Stop Listening".

121. `else:`: Handles the case where listening is stopped.

122. `speak("Listening stopped.")`: Informs the user that listening has ended.

123. `display_message("Listening stopped...")`: Updates the GUI.

124. `start_listening_button.config(text="Start Listening")`: Changes the button text back to "Start Listening".

## Start Listening Thread

python

```
def start_listening_thread():
    """Start the listening loop in a separate thread."""
    listening_thread = threading.Thread(target=continuous_listen)
    listening_thread.daemon = True # Ensure thread exits when the main program does
    listening_thread.start()
```

125. `def start_listening_thread()`: Defines a function to start the listening loop in a separate thread.

126. `listening_thread = threading.Thread(target=continuous_listen)`: Creates a new thread targeting the `continuous_listen` function.

127. `listening_thread.daemon = True`: Marks the thread as a daemon, allowing it to exit when the main program does.

128. `listening_thread.start()`: Starts the listening thread.

## GUI Setup

python

```
# Set up the Tkinter GUI
root = tk.Tk()
root.title("Real-Time Voice Assistant")
root.geometry("600x500")
```

129. `root = tk.Tk()`: Initializes the main Tkinter window.

130. `root.title("Real-Time Voice Assistant")`: Sets the window title.

131. `root.geometry("600x500")`: Sets the window size to 600x500 pixels.

## Text Area for Messages

python

```
# Text area to display commands and messages
text_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=10, width=60)
text_area.pack(pady=10)
```

132. `text_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=10, width=60)`: Creates a scrolled text area for displaying messages.

133. `text_area.pack(pady=10)`: Adds the text area to the window with vertical padding.

## Text Area for Available Commands

python

```
# Text area to display available commands (read-only)
command_list_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=6, width=60)
command_list_area.pack(pady=10)
```

134. `command_list_area = scrolledtext.ScrolledText(root, wrap=tk.WORD, height=6, width=60)`: Creates a scrolled text area for displaying available commands.

135. `command_list_area.pack(pady=10)`: Adds the command list area to the window with vertical padding.

## Start/Stop Listening Button

python

```
# Button to start/stop listening
start_listening_button = tk.Button(root, text="Start Listening", command=toggle_listening,
font=("Arial", 14))
start_listening_button.pack(pady=10)
```

136. `start_listening_button = tk.Button(root, text="Start Listening", command=toggle_listening, font=("Arial", 14))`: Creates a button to start/stop listening, linked to the `toggle_listening` function.

137. `start_listening_button.pack(pady=10)`: Adds the button to the window with vertical padding.

## Display Available Commands on Startup

python

```
# Display available commands on startup
display_available_commands()
```

138. `display_available_commands()`: Calls the function to display the list of available commands when the application starts.

## Start Assistant with Greeting

python

```
# Start the assistant with a greeting
speak("How can I help you?")
display_message("How can I help you?")
```

139. `speak("How can I help you?")`: Greets the user through text-to-speech.

140. `display_message("How can I help you?")`: Displays the greeting message in the GUI.

## Start Continuous Listening Thread

python

```
# Start the continuous listening thread
start_listening_thread()
```

141. `start_listening_thread()`: Calls the function to start the listening process in a separate thread.

## Run the GUI Main Loop

python

```
# Run the GUI main loop
root.mainloop()
```

142. `root.mainloop()`: Enters the Tkinter main loop, waiting for events (like button clicks or voice commands) to occur.

## Summary

This code creates a voice-activated assistant using Python libraries such as `speech_recognition` and `pyttsx3`. The assistant listens for commands and executes actions like opening applications, performing calculations, setting timers, and taking notes while providing audio feedback and displaying messages in a graphical interface.

ChatGPT can make mistakes. Check important info.