**OVER VIEW OF JARVIS APP**

1. **PYTTSX3**

DESCRIPTION:

PYTTSX3 IS A TEXT-TO-SPEECH (TTS) CONVERSION LIBRARY IN PYTHON. IT ALLOWS YOU TO CONVERT TEXT INTO SPOKEN WORDS, ENABLING APPLICATIONS TO HAVE SPEECH OUTPUT CAPABILITIES.

KEY FEATURES:

OFFLINE CAPABILITY: UNLIKE SOME TTS LIBRARIES THAT REQUIRE AN INTERNET CONNECTION, PYTTSX3 WORKS OFFLINE.

PLATFORM SUPPORT:IT WORKS ACROSS DIFFERENT PLATFORMS, INCLUDING WINDOWS, MACOS, AND LINUX.

CUSTOMIZABLE VOICE PROPERTIES: YOU CAN CHANGE PROPERTIES LIKE SPEECH RATE, VOLUME, AND VOICE TYPE (MALE, FEMALE).

USE CASE:

IDEAL FOR APPLICATIONS THAT REQUIRE VOCAL FEEDBACK OR AUDIO PROMPTS, SUCH AS VOICE ASSISTANTS, EDUCATIONAL TOOLS, OR ACCESSIBILITY APPLICATIONS FOR VISUALLY IMPAIRED USERS.

**2. SPEECH\_RECOGNITION**

DESCRIPTION:

THE SPEECH\_RECOGNITION LIBRARY PROVIDES FUNCTIONALITIES TO CONVERT SPOKEN LANGUAGE INTO TEXT. IT WORKS WITH VARIOUS SPEECH RECOGNITION APIS, INCLUDING GOOGLE SPEECH RECOGNITION.

KEY FEATURES:

MULTIPLE RECOGNIZERS: SUPPORTS SEVERAL RECOGNITION ENGINES, INCLUDING GOOGLE WEB SPEECH API, SPHINX, AND MORE.

MICROPHONE INPUT: CAN CAPTURE AUDIO DIRECTLY FROM THE MICROPHONE AND CONVERT IT TO TEXT.

ERROR HANDLING: PROVIDES MECHANISMS TO HANDLE ERRORS IN SPEECH RECOGNITION, LIKE UNCLEAR AUDIO.

USE CASE:

USEFUL FOR VOICE-CONTROLLED APPLICATIONS, DICTATION SOFTWARE, OR ANY APPLICATION THAT REQUIRES UNDERSTANDING SPOKEN COMMANDS.

**3. DATETIME**

DESCRIPTION: THE DATETIME MODULE SUPPLIES CLASSES FOR MANIPULATING DATES AND TIMES. IT PROVIDES A VARIETY OF FUNCTIONS TO WORK WITH DATES, TIMES, AND TIME INTERVALS.

KEY FEATURES:

DATE AND TIME MANIPULATION: YOU CAN EASILY CREATE, FORMAT, AND MANIPULATE DATE AND TIME OBJECTS.

TIME ZONES: SUPPORTS TIME ZONE ADJUSTMENTS AND CONVERSIONS.

FORMATTING: ALLOWS YOU TO FORMAT DATES AND TIMES IN VARIOUS HUMAN-READABLE FORMATS.

USE CASE: ESSENTIAL FOR ANY APPLICATION THAT REQUIRES SCHEDULING, TIMESTAMPING, OR DISPLAYING CURRENT DATES AND TIMES.

**4. WIKIPEDIA**

DESCRIPTION: THE WIKIPEDIA LIBRARY PROVIDES A SIMPLE INTERFACE TO RETRIEVE INFORMATION FROM WIKIPEDIA. IT ALLOWS YOU TO SEARCH AND RETRIEVE SUMMARIES OF WIKIPEDIA ARTICLES PROGRAMMATICALLY.

KEY FEATURES:

EASY ARTICLE ACCESS: YOU CAN EASILY ACCESS ANY WIKIPEDIA ARTICLE BY TITLE.

SUMMARIZATION: RETRIEVES CONCISE SUMMARIES OF ARTICLES, WHICH IS USEFUL FOR QUICK INFORMATION RETRIEVAL.

LANGUAGE SUPPORT: CAN FETCH ARTICLES IN DIFFERENT LANGUAGES BY SPECIFYING THE LANGUAGE CODE.

USE CASE: IDEAL FOR APPLICATIONS THAT NEED TO PROVIDE INFORMATION OR CONTEXT BASED ON USER QUERIES, SUCH AS CHATBOTS OR EDUCATIONAL TOOLS.

**5. WEBBROWSER**

DESCRIPTION: THE WEBBROWSER MODULE PROVIDES A HIGH-LEVEL INTERFACE FOR DISPLAYING WEB-BASED DOCUMENTS TO USERS. IT ALLOWS YOU TO OPEN URLS IN THE DEFAULT WEB BROWSER.

KEY FEATURES:

CROSS-PLATFORM SUPPORT: WORKS ON DIFFERENT OPERATING SYSTEMS WITHOUT ANY MODIFICATIONS.

OPEN IN DEFAULT BROWSER: AUTOMATICALLY OPENS URLS IN THE SYSTEM'S DEFAULT WEB BROWSER.

USE CASE: USEFUL FOR APPLICATIONS THAT REQUIRE ACCESSING WEB RESOURCES, SUCH AS OPENING DOCUMENTATION, WEB-BASED APPLICATIONS, OR SEARCH RESULTS.

**6. OS**

DESCRIPTION: THE OS MODULE PROVIDES A WAY TO INTERACT WITH THE OPERATING SYSTEM. IT ALLOWS YOU TO PERFORM VARIOUS OPERATING SYSTEM-RELATED TASKS, SUCH AS FILE AND DIRECTORY MANIPULATION.

KEY FEATURES:

FILE OPERATIONS: FUNCTIONS FOR CREATING, DELETING, AND MODIFYING FILES AND DIRECTORIES.

ENVIRONMENT VARIABLES: ACCESS AND MANIPULATE ENVIRONMENT VARIABLES.

PATH MANIPULATION: TOOLS FOR WORKING WITH FILE PATHS, ENSURING COMPATIBILITY ACROSS DIFFERENT OPERATING SYSTEMS.

USE CASE: ESSENTIAL FOR ANY APPLICATION THAT NEEDS TO READ FROM OR WRITE TO THE FILE SYSTEM, MANAGE FILE DIRECTORIES, OR PERFORM SYSTEM-LEVEL OPERATIONS.

**SUMMARY**

THESE LIBRARIES AND MODULES FORM THE BACKBONE OF YOUR VOICE ASSISTANT APPLICATION, ENABLING IT TO CONVERT TEXT TO SPEECH, RECOGNIZE SPOKEN COMMANDS, FETCH INFORMATION FROM THE INTERNET, AND INTERACT WITH THE FILE SYSTEM. THEY COLLECTIVELY ENHANCE THE FUNCTIONALITY AND USABILITY OF YOUR APPLICATION, MAKING IT MORE INTERACTIVE AND USER-FRIENDLY.

Detailed Explanation of the Code

1. Importing Libraries

python

Copy code

import pyttsx3

import speech\_recognition as sr

import datetime

import wikipedia

import webbrowser

import os

import pyttsx3: This library is used for converting text to speech. It allows your program to produce spoken audio from text strings, enabling the assistant to talk back to the user.

import speech\_recognition as sr: This library is essential for converting spoken language into text. It utilizes various speech recognition APIs, including Google’s Web Speech API, to recognize audio input.

import datetime: This module provides functions to work with dates and times. It allows the program to obtain the current time and format it for display or use in greetings.

import wikipedia: This library lets you search and retrieve information directly from Wikipedia. It can provide summaries, specific articles, and more based on the user's request.

import webbrowser: This module helps in opening URLs in the default web browser of the user's system. It makes it easy to navigate to specific websites through voice commands.

import os: The os module provides functions to interact with the operating system, like accessing file paths, listing directories, and executing system commands.

2. Initializing the Speech Engine

python

Copy code

engine = pyttsx3.init('sapi5')

voices = engine.getProperty('voices')

engine.setProperty("voice", voices[0].id)

engine = pyttsx3.init('sapi5'): Initializes the text-to-speech engine using SAPI5, which is a Microsoft speech API. This enables the application to use the built-in voices on Windows.

voices = engine.getProperty('voices'): Retrieves a list of available voices on the system. Each voice corresponds to a specific language and gender.

engine.setProperty("voice", voices[0].id): Sets the voice for the speech engine. In this case, it selects the first available voice. You can change the index (e.g., voices[1]) to use a different voice if available.

3. Defining the speak Function

python

Copy code

def speak(audio):

engine.say(audio)

engine.runAndWait()

def speak(audio):: This function takes a string argument called audio, which contains the text you want to be spoken.

engine.say(audio): This method adds the text to the speech queue.

engine.runAndWait(): This method processes the speech queue. It blocks the program until all the speech commands in the queue have been spoken. This ensures that the assistant speaks one command completely before taking any further action.

4. Defining the wishMe Function

python

Copy code

def wishMe():

hour = int(datetime.datetime.now().hour)

if hour >= 0 and hour < 12:

speak("Good Morning!")

elif hour >= 12 and hour < 18:

speak("Good Afternoon!")

else:

speak("Good Evening!")

speak("I am Jarvis, Please tell me how may I help you")

def wishMe():: This function greets the user based on the current time.

hour = int(datetime.datetime.now().hour): Obtains the current hour using the datetime module. The now() function returns the current date and time, and hour extracts just the hour in 24-hour format.

Conditional Statements:

Morning Greeting: If the hour is between 0 and 11, it speaks "Good Morning!"

Afternoon Greeting: If the hour is between 12 and 17, it speaks "Good Afternoon!"

Evening Greeting: For hours 18 and above, it speaks "Good Evening!"

Final Greeting: After greeting based on the time, it introduces itself as "Jarvis" and prompts the user for input, indicating readiness to assist.

5. Defining the takeCommand Function

python

Copy code

def takeCommand():

r = sr.Recognizer()

with sr.Microphone() as source:

print("Listening ... ")

r.pause\_threshold = 1

audio = r.listen(source)

try:

print("Recognizing ... ")

query = r.recognize\_google(audio, language='en-in')

print(f"User said: {query}\n")

except Exception as e:

print("Say that again please ... ")

return "None"

return query

def takeCommand():: This function listens for user input and converts the spoken words into text.

r = sr.Recognizer(): Creates an instance of the Recognizer class from the speech recognition library, which is responsible for processing audio.

with sr.Microphone() as source:: Uses the microphone as the input source for audio. The with statement ensures that the microphone resource is properly managed (opened and closed).

print("Listening ... "): Informs the user that the assistant is ready to listen.

r.pause\_threshold = 1: Sets a threshold for silence (1 second) after which the recognizer will stop listening. This helps avoid capturing too long of silence.

audio = r.listen(source): Listens for audio input from the user until the pause threshold is reached and stores it in the audio variable.

Recognition Process:

query = r.recognize\_google(audio, language='en-in'): Uses Google’s speech recognition service to convert the audio to text. The language='en-in' parameter specifies that the recognition is for English (India).

If recognition is successful, it prints what the user said.

Error Handling:

If an exception occurs (e.g., if the user doesn't speak clearly), it prompts the user to repeat and returns "None". This ensures the program doesn't crash and can continue running.

6. Main Program Logic

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

wishMe()

while True:

query = takeCommand().lower()

if 'wikipedia' in query:

speak('Searching Wikipedia ... ')

query = query.replace("wikipedia", "")

results = wikipedia.summary(query, sentences=2)

speak("According to Wikipedia")

print(results)

speak(results)

if 'wikipedia' in query:: This checks if the word "wikipedia" is part of the user’s spoken command. This is a common way to determine if the user is asking for information from Wikipedia.

speak('Searching Wikipedia ... '): It speaks out loud that it is searching Wikipedia. This provides feedback to the user, letting them know that their command is being processed.

query = query.replace("wikipedia", ""): This line removes the word "wikipedia" from the query string. This makes the subsequent search cleaner, focusing solely on the actual topic the user wants information about.

results = wikipedia.summary(query, sentences=2): Uses the wikipedia library to fetch a summary of the topic specified in the query. The sentences=2 argument indicates that the summary should consist of two sentences.

speak("According to Wikipedia"): It lets the user know that the following information is sourced from Wikipedia.

print(results): This prints the results to the console for the user to see.

speak(results): It reads the fetched summary aloud to the user.

Opening Websites

python

Copy code

elif 'open youtube' in query:

webbrowser.open("youtube.com")

elif 'open google' in query:

webbrowser.open("google.com")

elif "open stackoverflow" in query:

webbrowser.open("stackoverflow.com")

Each of these conditions checks if a specific phrase is included in the user’s command.

For example, if the user says "open YouTube," it will execute webbrowser.open("youtube.com"), which opens YouTube in the default web browser.

This logic is straightforward: the command just needs to match part of the string in the user's input to trigger the appropriate action.

Playing Music

python

Copy code

elif 'play music' in query:

music\_dir = "D:\\songs\\Favorite"

songs = os.listdir(music\_dir)

print(songs)

os.startfile(os.path.join(music\_dir, songs[0]))

elif 'play music' in query:: This checks if the command includes "play music."

music\_dir = "D:\\songs\\Favorite": Specifies the directory where the music files are stored. You should ensure this path is correct and contains valid audio files.

songs = os.listdir(music\_dir): Retrieves a list of all files in the specified music directory. This is a directory listing, which gives you access to the files in that location.

print(songs): Outputs the list of music files to the console, allowing you to see what songs are available.

os.startfile(os.path.join(music\_dir, songs[0])): This line constructs the full path to the first song in the list and plays it. os.startfile() is a Windows-specific function that opens the file with its associated application (e.g., a media player).

Time Inquiry

python

Copy code

elif "the time" in query:

strTime = datetime.datetime.now().strftime("%H:%M:%S")

speak(f"Sir, the time is {strTime}")

elif "the time" in query:: Checks if the user wants to know the current time.

strTime = datetime.datetime.now().strftime("%H:%M:%S"): Gets the current time using datetime.datetime.now() and formats it as HH:MM

using strftime. This format is both concise and clear.

speak(f"Sir, the time is {strTime}"): Speaks the formatted time back to the user, providing the information in a natural conversational way.

Exit Command

python

Copy code

elif "exit" in query or "stop" in query:

speak("Goodbye! Have a nice day!")

break

elif "exit" in query or "stop" in query:: This checks if the user has issued a command to exit the application.

speak("Goodbye! Have a nice day!"): This gives a farewell message to the user before the program terminates. It's a polite way to end the interaction.

break: This statement exits the while True: loop, effectively terminating the program.

Entry Point and Main Loop

The program structure is designed to allow continuous interaction with the user. Here's how it works:

Entry Point:

python

Copy code

if \_\_name\_\_ == "\_\_main\_\_":

wishMe()

The if \_\_name\_\_ == "\_\_main\_\_": line ensures that the following code runs only when the script is executed directly (not when imported as a module in another script).

wishMe(): Calls the wishMe function to greet the user, which makes the interaction feel more personalized and engaging from the start.

Main Loop:

python

Copy code

while True:

query = takeCommand().lower()

while True:: This starts an infinite loop, allowing the assistant to keep listening for commands until explicitly told to exit.

query = takeCommand().lower(): This captures the user’s spoken input through the takeCommand() function. The .lower() method is applied to make the input case-insensitive, simplifying the comparison checks later.

Summary

Overall, your Jarvis voice assistant is structured to perform various tasks based on simple voice commands. It:

Utilizes text-to-speech and speech recognition to create a conversational interface.

Integrates with external libraries like Wikipedia and webbrowser to provide useful functionalities.

Handles user commands in a loop, making it interactive and responsive.

The program is designed to be user-friendly, and each function is clearly defined, making it easier to maintain and extend. You could add more features, improve error handling, or customize responses to enhance user interaction further.