**Indian Flag Using Python Turtle**

**Introduction:**

In this project, we will show how to use [Turtle](https://docs.python.org/3/library/turtle.html) graphics using Python to make our Indian flag. Turtle is a pre-installed library in Python used to design a virtual canvas. In simple language, it’s a tool that helps you to draw on a board in your program with the help of a few commands. These commands are basically related to directions or the start and stop of drawing. So gear up with your IDE and try this out!

**Explanation:**

As turtle is a library in Python, we will first start by importing it into our program using the import keyword. Next, we import \* from the turtle, this means that all the functions defined in the turtle library will be imported and are free to use in this code.

The second step is to create a screen or your drawing board for the graphics. This can be done by using a turtle.Screen(). This assigns a screen to the variable screen. Now to draw, you will need a medium. On paper, it’s your pencil, and with the virtual board or screen it’s the turtle. Hence, we assign turtle to the variable t using turtle.Turtle() and set its speed to 0. Speed settings can be set between 1 (slowest) to 10 (fastest). At speed 0, all animations are stopped and the turtle can go as fast as possible.

Now, the only thing left to do is to draw.

Start by lifting the pen up which means that it won’t draw whatever direction it goes. Now set the location of the pen(turtle) to the coordinates (-400,250) using method goto(). The coordinates may differ from person to person. Now, initially, the screen is white, so you won’t need to create the white rectangle for the flag but green and orange and also the Ashoka Chakra.

Firstly, we set the color to orange and next use begin\_fill() so that it fills the shape with orange color as the border is created. Next, we set the directions for the turtle to move using methods like right(), left(), and forward(). The trick we used here is after we created the orange rectangle, we extended the vertical left line to reach the co-ordinated where the green rectangle begins. The same steps are to be followed for the green rectangle.

Remember to**penup()** as we move to the center of the white rectangle for the Ashoka Chakra.

The logic here is to make a big circle with the color navy blue. Then draw another circle concentric to this circle with a smaller radius. Concentric circles are circles with the same center with different radii. This will give a circular border effect. Similarly, we draw a small navy blue circle over the white circle in the center. Repeat the steps! =>

t.penup() => t.goto() => t.pendown() => t.color() => t.begin\_fill() => t.circle() => t.end\_fill()

Following the above step next we draw 24 circles on the border and connect to the center by drawing spokes and this is done by using the for loop. And bravo! You have learned to use a turtle and make your flag in it.

**Source Code:**

import turtle

from turtle import\*

#screen for output

screen = turtle.Screen()

# Defining a turtle Instance

t = turtle.Turtle()

speed(0)

# initially penup()

t.penup()

t.goto(-400, 250)

t.pendown()

# Orange Rectangle

#white rectangle

t.color("orange")

t.begin\_fill()

t.forward(800)

t.right(90)

t.forward(167)

t.right(90)

t.forward(800)

t.end\_fill()

t.left(90)

t.forward(167)

# Green Rectangle

t.color("green")

t.begin\_fill()

t.forward(167)

t.left(90)

t.forward(800)

t.left(90)

t.forward(167)

t.end\_fill()

# Big Blue Circle

t.penup()

t.goto(70, 0)

t.pendown()

t.color("navy")

t.begin\_fill()

t.circle(70)

t.end\_fill()

# Big White Circle

t.penup()

t.goto(60, 0)

t.pendown()

t.color("white")

t.begin\_fill()

t.circle(60)

t.end\_fill()

# Mini Blue Circles

t.penup()

t.goto(-57, -8)

t.pendown()

t.color("navy")

# Small Blue Circle

t.penup()

t.goto(20, 0)

t.pendown()

t.begin\_fill()

t.circle(20)

t.end\_fill()

# Spokes

t.penup()

t.goto(0, 0)

t.pendown()

t.pensize(2)

for i in range(24):

t.forward(60)

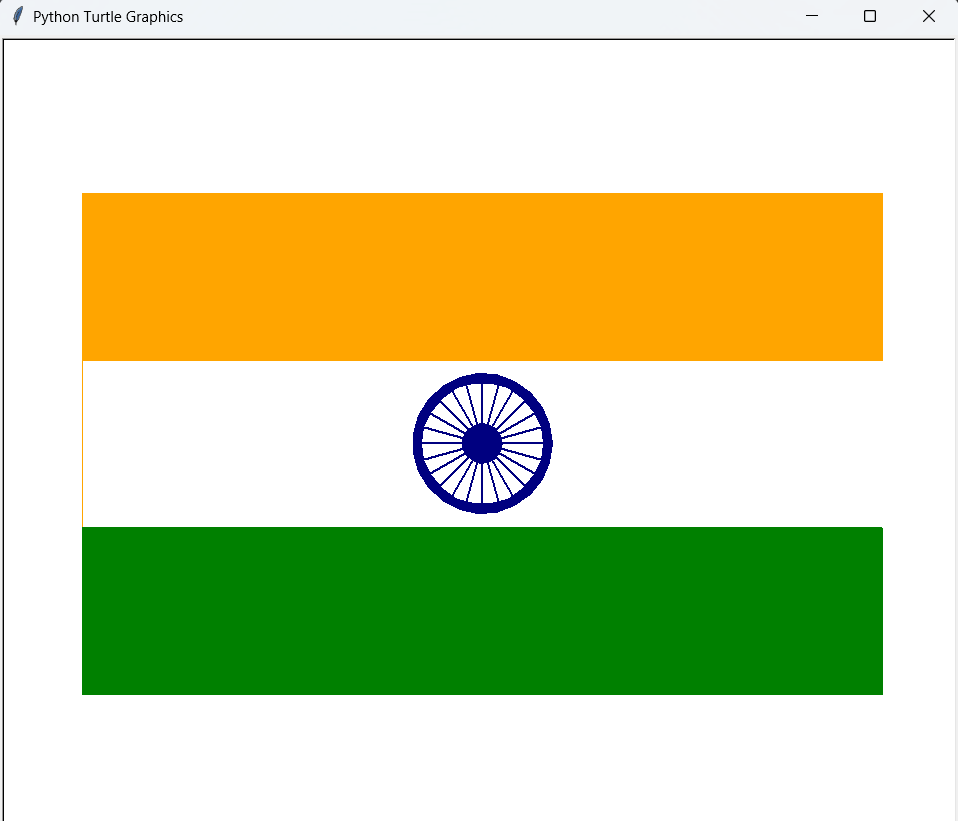
t.backward(60)

t.left(15)

#to hold the

#output window

turtle.done()Output



Real-Time Face Detection

**Introduction:**

The provided Python code demonstrates a real-time face detection application using the OpenCV (Open Source Computer Vision) library. Face detection is a fundamental task in computer vision, and it involves locating and identifying human faces within digital images or video streams. In this script, a pre-trained Haar cascade classifier for frontal face detection is utilized to detect faces from a live video feed.

This code is a Python script that uses the OpenCV library to perform real-time face detection from a live video stream. Let's break down the code step-by-step:

1. Importing necessary libraries:
   * **cv2**: The OpenCV library used for computer vision tasks, including image and video processing.
2. Loading the face detection cascade classifier:
   * **cv2.CascadeClassifier**: A class in OpenCV used to load pre-trained Haar cascades for object detection, in this case, a frontal face detection cascade.
   * The **haarcascade\_frontalface\_default.xml** file contains the pre-trained model for detecting frontal faces. The full path to the XML file is provided.
3. Initializing the video capture:
   * **cv2.VideoCapture**: A class in OpenCV used to capture video from various sources, such as a webcam or video file.
   * **videoCap = cv2.VideoCapture(0)**: The code initializes the video capture object to capture video from the default camera (index 0). You can replace **0** with the index of a different camera if you have multiple cameras connected.
4. Main loop for real-time face detection:
   * The loop runs indefinitely until the user presses the "a" key.
   * **videoCap.read()**: It reads a frame from the video capture object. The return value **ret** is a boolean indicating whether the frame was read successfully, and **video** is the captured frame as a NumPy array (BGR format).
   * **cv2.cvtColor(video, cv2.COLOR\_BGR2GRAY)**: Converts the captured frame (in BGR color) to grayscale. Grayscale images are used for better performance in face detection.
   * **face\_cap.detectMultiScale(...)**: It detects faces in the grayscale frame using the pre-trained Haar cascade classifier. The detected faces are returned as a list of rectangles, each representing a face's position and size.
   * **for (x, y, w, h) in faces:**: Loop through the detected faces and draw rectangles around them on the original (color) frame using **cv2.rectangle(...)**.
5. Displaying the video stream with detected faces:
   * **cv2.imshow("video.live", video)**: It displays the current frame (**video**) with the detected faces on a window named "video.live."
6. Breaking the loop and cleaning up:
   * **cv2.waitKey(10) == ord("a")**: The loop waits for a key press for 10 milliseconds. If the key "a" is pressed, the loop breaks, and the program moves on to the release and cleanup stage.
   * **videoCap.release()**: It releases the video capture object, allowing the camera to be used by other applications.
   * **cv2.destroyAllWindows()**: Closes all the OpenCV windows that were created.

Overall, this script captures video from the default camera, detects faces in real-time using the Haar cascade classifier, and displays the video stream with rectangles around the detected faces. The loop continues until the user presses the "a" key, at which point the video capture is released, and all OpenCV windows are closed, terminating the program.

**Algorithm:**

1. Import the necessary libraries:
   * Import the OpenCV library (**cv2**) to access its computer vision functions and classes.
2. Load the pre-trained face detection cascade classifier:
   * Initialize the face detection cascade classifier (**cv2.CascadeClassifier**) with the path to the Haar cascade file for frontal face detection (**haarcascade\_frontalface\_default.xml**).
3. Initialize video capture:
   * Create a video capture object (**cv2.VideoCapture**) to access the video stream from the default camera (index 0).
4. Real-time face detection loop:
   * Enter an infinite loop to continuously process video frames from the camera.
   * For each iteration:
     + Read a frame from the video capture object (**videoCap.read()**).
     + Convert the captured frame to grayscale (**cv2.cvtColor(video, cv2.COLOR\_BGR2GRAY)**). Grayscale images are used for better performance in face detection.
     + Use the face detection cascade classifier to detect faces in the grayscale frame (**face\_cap.detectMultiScale(...)**). The classifier returns a list of rectangles, each representing the position and size of a detected face.
     + Iterate through the detected faces and draw rectangles around them on the original (color) frame using **cv2.rectangle(...)**.
     + Display the video frame with the detected faces on a window named "video.live" (**cv2.imshow("video.live", video)**).
     + Wait for a key press for 10 milliseconds. If the key "a" is pressed, break the loop and proceed to the cleanup stage.
5. Release resources and clean up:
   * Release the video capture object (**videoCap.release()**) to free the camera for other applications.
   * Close all OpenCV windows (**cv2.destroyAllWindows()**) that were created during the execution of the program.

**Summary:**

The Python script utilizes OpenCV's Haar cascade classifier for frontal face detection to create a real-time face detection application. The script continuously captures video frames from the default camera, processes them to detect faces, and displays the live video stream with rectangles drawn around the detected faces. The application runs in an infinite loop until the user presses the "a" key, at which point it releases the video capture object and closes all OpenCV windows, terminating the program. This script provides a simple yet effective example of real-time face detection using OpenCV and demonstrates the power of computer vision in real-world applications.

**Source code\_\_\_\_\_**

import cv2

face\_cap = cv2.CascadeClassifier("C:/Program Files/Python311/Lib/site-packages/opencv-4.x/data/haarcascades/haarcascade\_frontalface\_default.xml")

videoCap = cv2.VideoCapture(0)

while True:

ret, video = videoCap.read()

col = cv2.cvtColor(video, cv2.COLOR\_BGR2GRAY)

faces = face\_cap.detectMultiScale(col, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30), flags=cv2.CASCADE\_SCALE\_IMAGE)

for (x, y, w, h) in faces:

cv2.rectangle(video, (x, y), (x + w, y + h), (0, 255, 0), 2)

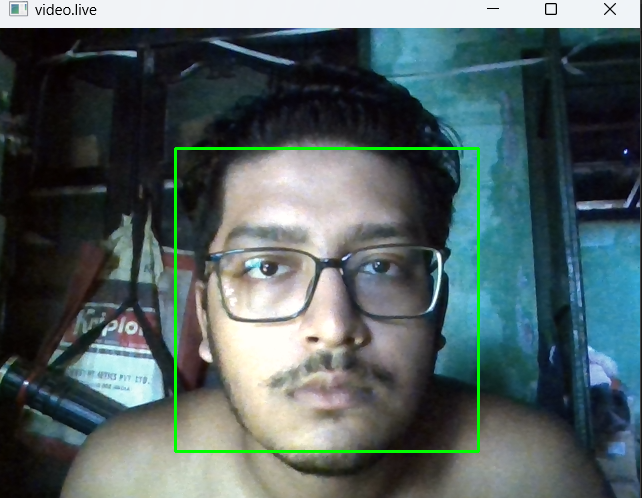
cv2.imshow("video.live", video)

if cv2.waitKey(10) == ord("a"):

break

videoCap.release()

cv2.destroyAllWindows()



**Phone number get details**

**Introduction:**

The provided Python script is an application that utilizes the **phonenumbers** library to gather information about a phone number with a country code. The script takes a phone number as input, validates it, and then provides information such as the timezone, service provider (carrier), and country using the **phonenumbers** library functions. Additionally, it checks whether the phone number is valid and if it is possible to be a valid number.

**Algorithm:**

1. Import the necessary libraries:
   * Import the **phonenumbers** library to work with phone numbers and its related data.
   * Import specific modules from **phonenumbers** to access functions like timezone, geocoder, and carrier data.
2. Enter an infinite loop:
   * The script enters an infinite loop (**while True:**) to allow the user to input multiple phone numbers and get their information.
3. Read the input phone number:
   * Prompt the user to enter a phone number with a country code (**number=input("Enter a number with country code:\t")**).
4. Parse and process the phone number:
   * Use **phonenumbers.parse(number)** to parse the phone number and obtain a **PhoneNumber** object (**phone**) containing the parsed information.
5. Get additional information about the phone number:
   * Use **timezone.time\_zones\_for\_geographical\_number(phone)** to get the timezone information for the provided phone number.
   * Use **carrier.name\_for\_number(phone, "en")** to get the service provider (carrier) information for the phone number in English.
   * Use **geocoder.description\_for\_number(phone, "en")** to get the country information for the phone number in English.
6. Validate and check the possibility of the phone number:
   * Use **phonenumbers.is\_valid\_number(phone)** to check if the phone number is valid.
   * Use **phonenumbers.is\_possible\_number(phone)** to check if the phone number is possible (e.g., it meets the general requirements but might not be assigned to a subscriber).
7. Print the gathered information:
   * Print the phone number, validity, possibility, timezone, service provider (carrier), and country.
8. Continue to the next iteration of the loop:
   * The script will continue to the next iteration of the loop to prompt the user for another phone number until the user chooses to terminate the program externally.

**Summary:**

The Python script uses the **phonenumbers** library to collect and present information about a phone number with a country code. It parses the input phone number and then retrieves the timezone, carrier, and country information using appropriate library functions. The script also checks whether the phone number is valid and possible. The loop ensures the user can input multiple phone numbers to gather information until the user decides to stop the execution externally. This script serves as a useful tool to analyze and validate phone numbers along with their associated details.

**Source code\_\_\_**

import phonenumbers

from phonenumbers import timezone,geocoder,carrier,carrierdata

while True:

number=input("Entar a number with country code:\t")

phone=phonenumbers.parse(number)

time=timezone.time\_zones\_for\_geographical\_number(phone)

cari=carrier.name\_for\_number(phone,"en")

reg=geocoder.description\_for\_number(phone,"en")

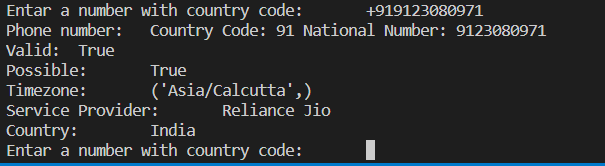
# Validating a phone number

valid = phonenumbers.is\_valid\_number(phone)

# Checking possibility of a number

possible = phonenumbers.is\_possible\_number(phone)

print(f"Phone number:\t{phone}\nValid:\t{valid}\nPossible:\t{possible}\nTimezone:\t{time}\nService Provider:\t{cari}\nCountry:\t{reg}")



**QR maker**

**Introduction:**

The provided Python script utilizes the **qrcode** library to generate a QR code for a specific URL and saves it as an image file. QR codes are two-dimensional barcodes that can store various types of information, such as URLs, text, or contact details. In this script, we are generating a QR code for the URL "skf.edu.in" and saving it as "skf.png."

**Algorithm:**

1. Import the necessary libraries:
   * Import the **qrcode** library to work with QR code generation.
   * Import **Image** from the **PIL** (Python Imaging Library) to work with images.
2. Initialize the QR code object:
   * Create a new **QRCode** object and configure its properties:
     + Set the **version** to 1, indicating the QR code version. Higher versions support more data capacity.
     + Set the **error\_correction** to **qrcode.constants.ERROR\_CORRECT\_H**, indicating the error correction level. Higher levels provide better error correction capability.
     + Set the **box\_size** to 10, defining the size of each QR code module (box).
     + Set the **border** to 4, defining the number of boxes forming the white border around the QR code.
3. Add data to the QR code:
   * Use the **qr.add\_data(...)** method to add the data to be encoded in the QR code. In this case, we add the URL "skf.edu.in."
4. Make the QR code:
   * Use the **qr.make(fit=True)** method to generate the QR code based on the provided data. The **fit=True** parameter ensures the QR code automatically adjusts its size to accommodate the data.
5. Create an image representation of the QR code:
   * Use the **qr.make\_image(...)** method to convert the QR code into an image. We set the **fill\_color** to "black" and **back\_color** to "white," indicating the color of the QR code modules and background, respectively.
6. Save the QR code image:
   * Use the **img.save(...)** method to save the generated QR code image as "skf.png" in the current working directory.

**Summary:**

The Python script uses the **qrcode** library to generate a QR code for the URL "skf.edu.in" with specified properties such as version, error correction level, box size, and border. The generated QR code image is saved as "skf.png" in the current working directory. QR codes are widely used for sharing information, URLs, or contact details in a convenient and machine-readable format. This script serves as a simple example of how to generate QR codes programmatically using Python.

**Source code----**

import qrcode

from PIL import Image

qr=qrcode.QRCode(version=1,error\_correction=qrcode.constants.ERROR\_CORRECT\_H,box\_size=10,border=4)

qr.add\_data("skf.edu.in")

qr.make(fit=True)

img=qr.make\_image(fill\_color="black",back\_color="white")

img.save("skf.png")



**Jarvis voice assistant**

**Introduction:**

The provided Python script is an implementation of a basic virtual assistant named "Jarvis." It uses the **pyttsx3**, **datetime**, **wikipedia**, **speech\_recognition**, **webbrowser**, and **os** libraries to interact with the user through speech, perform tasks based on user commands, and access various functionalities like searching Wikipedia, opening webpages, playing music, telling the time, and opening applications.

**Algorithm:**

1. Import the necessary libraries:
   * Import **pyttsx3** for text-to-speech functionality.
   * Import **datetime** to work with date and time.
   * Import **wikipedia** to access Wikipedia for searching information.
   * Import **speech\_recognition** as **sr** for speech recognition functionality.
   * Import **webbrowser** to open webpages.
   * Import **os** for interacting with the operating system.
2. Initialize the text-to-speech engine:
   * Use **pyttsx3.init('sapi5')** to initialize the text-to-speech engine with the 'sapi5' voice API.
   * Set the voice property to one of the available voices from the system (e.g., **voices[0].id**).
3. Define functions for speaking and wishing:
   * **speak(audio)** takes an input string **audio** and uses the text-to-speech engine to speak it out.
   * **wishMe()** greets the user based on the current time and introduces the virtual assistant as "Jarvis."
4. Implement the function to take user commands through speech:
   * **takeCommand()** listens to the user's voice input using the microphone (**sr.Microphone**).
   * Recognizes the audio using Google's speech recognition (**r.recognize\_google**) and returns the recognized text as **query**.
5. Main program:
   * The **\_\_name\_\_=="\_\_main\_\_"** condition ensures that the following code executes only if the script is run as the main program.
   * Call **wishMe()** to greet the user and introduce the virtual assistant.
   * Enter an infinite loop to continuously listen to user commands using **takeCommand()**.
   * Based on the recognized **query**, perform different tasks using conditional statements (**if-elif-else**).
   * Supported tasks include searching Wikipedia, opening YouTube, Google, Stack Overflow, Chat GPT from OpenAI, playing music, telling the time, and opening Visual Studio Code.
   * The virtual assistant responds with speech for relevant tasks and executes the corresponding actions.

**Summary:**

The Python script implements a basic virtual assistant "Jarvis" that interacts with the user through speech. It recognizes user commands, performs various tasks like searching Wikipedia, opening webpages, playing music, telling the time, and opening applications based on the user's voice input. The script demonstrates the integration of text-to-speech and speech recognition functionalities, making it a simple yet effective virtual assistant that responds to user commands and queries.

**Source code----**

import pyttsx3

import datetime

import wikipedia

import speech\_recognition as sr

import webbrowser

import os

engine = pyttsx3.init('sapi5')

voices = engine.getProperty('voices')

# print(voices[1].id)

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

def speak(audio):

engine.say(audio)

engine.runAndWait()

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 SIR me how can I help You")

def takeCommand():

r = sr.Recognizer()

with sr.Microphone() as source:

print("Listening...")

r.pause\_threshold=1

audio = r.listen(source)

try:

print("Recocnizing...")

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

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

except Exception:

# print(e)

print("Say That again...")

return "None"

return query

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

wishMe()

while True:

query=takeCommand().lower()

# logic for executingn task based on query

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)

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")

elif 'open chat gpt' in query:

webbrowser.open("chat.openai.com")

elif 'play music' in query:

music\_dir = "C:\\Users\\rajde\\OneDrive\\Desktop\\JARVIS\\musics"

songs = os.listdir(music\_dir)

print(songs)

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

elif 'the time' in query:

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

speak(f"The time is{strTime}")

print(f"The time is{strTime}")

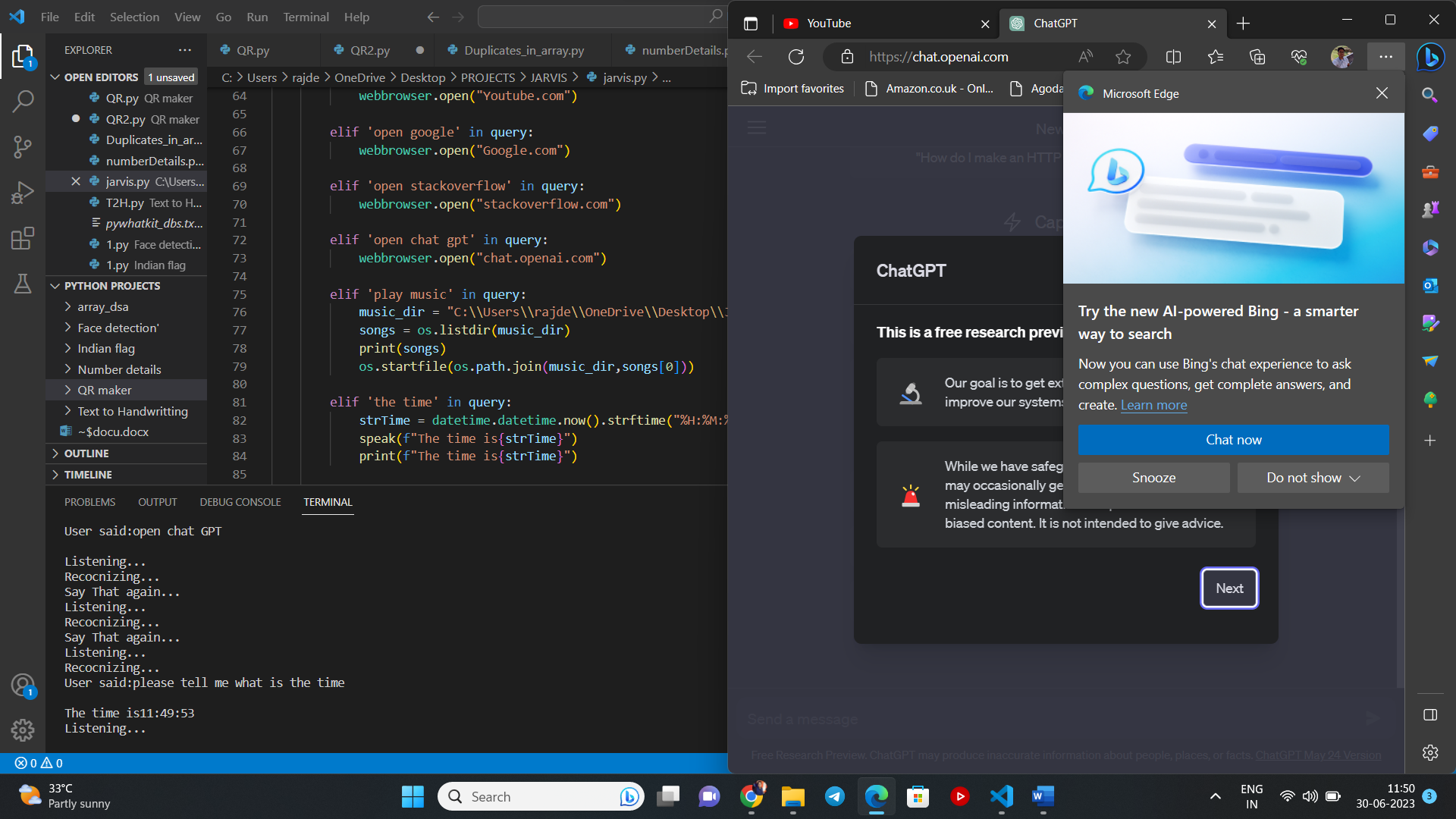
elif 'open code' in query:

codepath = "C:\\Users\\rajde\\AppData\\Local\\Programs\\Microsoft VS Code\\Code.exe"

os.startfile(codepath)

elif 'very good' in query:

speak("Thank You sir")

****

**SpeakBot**

This code defines a simple program called SpeakerBot v1.1 using the **pyttsx3** library, which allows the bot to speak the text entered by the user. Here's an explanation of the code:

1. Import **pyttsx3**: The code begins by importing the **pyttsx3** library, which is a text-to-speech conversion library in Python.
2. Define the **speak** function: The **speak** function takes a single parameter **text** and uses **pyttsx3.init()** to initialize the text-to-speech engine. It then uses **engine.say(text)** to pass the **text** to the engine for speech synthesis. Finally, **engine.runAndWait()** is used to wait until the speech is complete before the program proceeds.
3. **\_\_name\_\_** check and welcome message: The script then checks whether it is being run as the main program using **if \_\_name\_\_ == '\_\_main\_\_':**. If true, it prints the "Welcome to SpeakerBot v1.1" message.
4. Main loop: The script enters a **while True:** loop, which runs indefinitely until the user decides to quit the program.
5. User input and speaking functionality: Inside the loop, the script takes input from the user using **x = input("Enter what you want me to speak: ")**. The user can type any text, and the **speak** function is called with the entered text, causing the bot to speak the input using the system's default text-to-speech engine.

Note that this script solely focuses on the speaking functionality and does not include any options to write a report or perform other tasks. It provides a simple way for users to make the bot read out any text they input.

To run the program, you can execute it in a Python environment and interact with it by entering text that you want the bot to speak. For example:

cssCopy code

Welcome to SpeakerBot v1.1 Enter what you want me to speak: Hello, I am SpeakerBot v1.1

The bot will then speak the text "Hello, I am SpeakerBot v1.1."

Top of Form

Regenerate response

**Code==**

import pyttsx3

def speak(text):

engine = pyttsx3.init()

engine.say(text)

engine.runAndWait()

if \_\_name\_\_ == '\_\_main\_\_':

print("Welcome to SpeakerBot v1.1")

while True:

x = input("Enter what you want me to speak: ")

speak(x)

**Whether Bot**

This code is a simple Python script that retrieves weather data for a given city using the WeatherAPI and then uses the `pyttsx3` library to speak out some of the weather information. Let me explain it step by step:

1. Import libraries: The script starts by importing the necessary libraries, including `requests`, `json`, and `pyttsx3`.

2. Define the `speak` function: Just like in the previous code, the `speak` function is used to convert text to speech using the `pyttsx3` library.

3. User input: The script prompts the user to input the name of the city they want to get weather information for, and the input is stored in the `city` variable.

4. API call and data retrieval: The script constructs the API URL using the provided city and makes a GET request to the WeatherAPI using `requests.get(url)`. The response is stored in the variable `r`.

5. Parsing JSON data: The API response contains weather data in JSON format. The script uses `json.loads(r.text)` to convert the JSON data into a Python dictionary named `wDic`.

6. Speaking the weather information: The script uses the `speak` function to read out some key weather information to the user. Specifically, it speaks the current temperature in Celsius (`wDic["current"]["temp\_c"]`), humidity (`wDic["current"]["humidity"]`), and the "feels like" temperature (`wDic["current"]["feelslike\_c"]`).

7. Explanation: The `explain` function seems to be missing, and the provided code doesn't have any further explanation. If you intended to explain the code further, you can add comments to clarify different parts of the code.

To run the program, execute it in a Python environment. When prompted, enter the name of the city for which you want to check the weather. The script will retrieve the weather data and then read out the temperature, humidity, and "feels like" temperature using the text-to-speech functionality provided by `pyttsx3`.

Note: The script will work as expected as long as the WeatherAPI URL is correct and the API key (`key=3f5effc010924b49881110231230107`) is valid. Also, ensure that you have the `requests`, `json`, and `pyttsx3` libraries installed in your Python environment.

**Code==**

import requests

import json

import pyttsx3

def speak(text):

engine = pyttsx3.init()

engine.say(text)

engine.runAndWait()

city=input("Enter the name of the city\n ")

url= f"https://api.weatherapi.com/v1/current.json?key=3f5effc010924b49881110231230107&q={city}"

r= requests.get(url)

# print(r.text)

wDic=json.loads(r.text)

speak("Temperature is")

speak(wDic["current"]["temp\_c"])

speak("and the humidity is")

speak(wDic["current"]["humidity"])

speak("and it feels like")

speak(wDic["current"]["feelslike\_c"])

Bottom of Form