# Ici on va chercher les autres librairies que l'on a de besoin pour que ça fonctionne

# C'est comme le code qui a été écrit par d'autres personnes pour nous permettre de le réutiliser

import os

from openai import OpenAI

from playsound import playsound # attention: pip install playsound==1.2.2

from datetime import datetime

import speech\_recognition as sr

import re

import asyncio

# On a ajouté une fonction main() "asynchrone"

# Par convention, c'est comme ça que l'on appelle la fonction qui sera appelé au démarage de l'application.

# La fonction principale, c'est donc la fonction "main".

# Elle est "asynchrone", ça veut dire que l'on va pouvoir dans cette fonction faire plus d'une chose en même temps!

# Par exemple, on va écouter la première phrase de la réponse "en même temps" que l'on va aller chercher la prochaine

# phrase à écouter.  Comme ça, ça nous permet d'avoir la réponse qui commence à parler beaucoup plus vite.

async def main():

    # Ici c'est notre clé top secrète pour appeler ChatGPT.

    # C'est comme si on avait rentré notre nom d'utilisateur et notre mot de passe.

    # N'importe qui qui a cette clé là peut utiliser notre compte et c'est nous qui sommes facturé.

    client = OpenAI(api\_key="sk-")

    # Ça c'est juste pour bien debuger, pour nous permettre de faire un nouveau sous répertoire à chaque expérience

    # On va mettre tous nos fichiers dedans pour les retrouver facilement

    datetime\_folder = datetime.now().strftime("%Y-%m-%d-%H-%M-%S")

    experiment\_folder = os.path.join("experiments", datetime\_folder)

    os.mkdir(experiment\_folder)

    # Ceci, c'est une fonction qui sert à écouter au micro et à tout convertir en texte une fois que

    # l'enfant à dit "stop".  Il peut prendre son temps pour parler et même si il y a des blancs, ça

    # va continuer d'enregistrer jusqu'à ce qu'on entende le mot clé "stop".

    def listen\_and\_transcribe():

        # Pour se faire, on va utiliser un objet pour reconaître la voix.  C'est notre "recognizer"

        recognizer = sr.Recognizer()

        # Puis, on va se garder une variable qui va contenir tout le texte que l'on aura reconnu du micro.

        text = ""

        # Ici on commence à écouter au micro.  On dit donc que l'on veut utiliser le micro comme source

        with sr.Microphone() as source:

            # Avec notre micro, on va commencer par "écouter" le son ambiant pour faire en sorte que lorsque

            # l'on parle, on est capable de reconnaître la voix de tous les bruits de fond.

            print("Ajustement du son ambiant.")

            recognizer.adjust\_for\_ambient\_noise(source)

            # Et maintenant, on commence la vrai saissie pour écouter ce qui se dit et le convertir en texte.

            print("En écoute... Dites 'STOP' pour finir ce que vous avez à dire.")

            # Gardons une variable pour savoir nous en sommes où dans l'écoute de nos segments

            audio\_part\_index = 0

            # On va faire une boucle qui va prendre chaque "segment" de conversation et les reconnaître, tant que

            # nous n'avons pas entendu "STOP".  Un segment, c'est quand quelqu'un fait une pause en parlant...

            # ...et à chaque fois qu'il y a une pause, on en profite pour convertir ce qu'on vient d'entendre en texte

            while True:

                # On se met en mode écoute du microphone

                audio = recognizer.listen(source)

                # On va enregistrer l'audio entendu dans un fichier de notre répertoire d'expériences

                filename = os.path.join(experiment\_folder, str(len(prompts)).zfill(4) + "-kid (" + str(audio\_part\_index).zfill(2) + ").wav")

                with open(filename, "wb") as file:

                    file.write(audio.get\_wav\_data())

                audio\_part\_index += 1

                # Puis on essaie de reconnaître ce qui a été dit.

                try:

                    # C'est notre opération de convertir la voix en texte,

                    # ou comme on appelle en anglais le Speach-To-Text (STT)

                    text = text + " " + recognizer.recognize\_google(audio, language="fr-FR")

                    print(f"Texte reconnu: {text}")

                    # Puis on regarde si le mot clé "STOP" est dans le texte reconnu

                    if "STOP" in text.upper():

                        # Si on l'a trouvé, on va arrêter l'écoute du microphone

                        print("On arrête l'écoute...")

                        # On va dire que notre fonction retourne tout ce qu'il y avait comme texte de

                        # détecté "avant" le mot clé "STOP"

                        return text.split("stop")[0]

                        break

                # Mais quand on s'essaie, c'est possible que ça ne fonctionne pas.  Si c'est le cas, on

                # va dire que l'on a eu un cas "d'exception" qui n'est pas ce que l'on attendait.

                except sr.UnknownValueError:

                    print("Exception: Could not understand audio")

                except sr.RequestError as e:

                    print(f"Exception: Could not request results; {e}")

    # Maintenant, premier concept quand on parle à ChatGPT.  On fait des "PROMPTs", c'est à dire des messages.

    # On va se faire une liste de message que l'on s'échange.  En gros ce sont les messages de ChatGPT, les miens, ChatGPT, les miens, ...

    # Dans le langage de ChatGPT, quand ça vient de ChatGPT on va dire que le role=assistant et quand ça vient de l'utilisateur c'est role=user

    prompts = []

    # Mais notre liste, on va toujours la commencer par des messages "SYSTEM" qui expliquent à ChatGPT c'est quoi son rôle, ce qu'il peut faire

    # et ce qu'il ne peut pas faire.  Comment il s'appelle... c'est toute sa personalité et son information de base qui est là dedans.

    prompt\_System = []

    prompt\_System.append("Ton nom est Elis.")

    prompt\_System.append("Tu es une peluche en forme d'ourson.")

    prompt\_System.append("Tu donnes toutes tes réponses en français.")

    prompt\_System.append("Tu t'adresses à un enfant de 5 ans.")

    prompt\_System.append("Tes réponses sont simples, amusantes, appropriées, bienveillantes et engagentes pour un enfant de 5 ans.")

    prompt\_System.append("Tes réponses ont moins de 100 mots au maximum, sans exception.")

    prompt\_System.append("Tu ne fais pas de reproches.")

    prompt\_System.append("Tu veux jouer avec l'enfant.")

    prompt\_System.append("Tu reçois tes questions par la voix et tu les donnes aussi par la voix, donc tu ne fais pas de référence à écrire quelque chose pour communiquer avec toi.")

    prompt\_System.append("Tu habites au Québec et tes référents sont Québecois.")

    prompt\_System.append("Tu ne l'encourages pas à être vulgaire, mais tu comprends que c'est un enfant et tu essaie de ramener la conversation sur un ton positif en comprenant quand même son intention.")

    prompt\_System.append("Tu ne réponds pas que tu es un agent conversationnel, une intellligence artificielle ou une autre variante du même genre.")

    # Et on va commencer av

    answer = "Bonjour mon ami! Je suis là pour toi: que veux-tu faire?"

    # On va se garder une variable pour savoir si on doit continuer notre discussion ou arrêter

    # Pour arrêter, c'est d'écrire "quit" comme réponse de notre part quand c'est à nous d'écrire

    shouldContinue = True

    # Et ici c'est notre boucle... tant que la variable "souldContinue" nous dit de continuer... et bien on va continuer

    while shouldContinue:

        # On va afficher à l'écran ce que dit Elis

        print("ELIS> " + answer)

        # Et l'ajouter à la liste des messages (les PROMPTs) que l'on c'est échangé

        # On se souvient: le role assistant c'est quand c'est la réponse de ChatGPT

        prompts.append({"role": "assistant", "content": answer})

        print()

        # De la réponse que l'on a reçu, on va séparer chaque phrase.

        # split\_answers est une liste de toute les phrases.

        # Ça va nous permettre de les lire une après l'autre et d'avoir la réponse plus rapidement pour l'enfant.

        # Par exemple, si on avait 5 paragraphes de réponses avec un total de 25 phrases, ça pourrait prendre

        # 30 secondes recevoir la réponse de tout l'audio à lire.  Mais si on prend juste la première phrase,

        # on a peut-être la réponse en 2 secondes et pendant qu'on va la faire jouer, en même temps on va aller chercher

        # la prochaine phrase à lire pour qu'elle soit disponible quand on va avoir finit d'écouter la première.

        # C'est ce qu'on pourrait appeler un truc ou une astuce pour améliorer notre expérience.

        # On peut aussi appeler ça dans le jargon, une heuristique.

        # Le fait de pouvoir aller chercher le contenu à lire en petit morceau plutôt que tout d'un coup, on appelle

        # ça du "streaming".  On peut donc dire que l'on a codé une heuristique de streaming !

        split\_answers = re.findall(r'[^.!?]\*[.!?]', answer)

        # On va se garder un index que quelle phrase (quelle partie de la réponse) nous sommes entrain de lire.

        answer\_part\_index = 0

        # Et on va se garder une variable qui sert à contrôler notre lecture.  Ce sera task\_player

        task\_player = asyncio.sleep(0)

        # On se fait une boucle pour passer chaque phrase une à la suite de l'autre.

        for answer\_part in split\_answers:

            # Ici c'est le code pour transformer le texte en son.  On appelle ça du TTS (text-to-speech)

            # On va prendre la dernière chose que nous a dit ChatGPT et on va demander à un site web de OpenAI

            # de nous le convertir dans un fichier audio, comme un fichier mp3

            response = client.audio.speech.create(

                model="tts-1",

                # la liste des voix est disponible au https://platform.openai.com/docs/guides/text-to-speech

                # Présentement, on peut choisir entre: alloy, echo, fable, onyx, nova, ou shimmer

                voice="echo",

                response\_format="mp3",

                input=answer\_part

                )

            # Puis on va se faire un fichier où tout enregistrer ce son là dans notre répertoire spéciale fait au début

            filename = os.path.join(experiment\_folder, str(len(prompts)).zfill(4) + "-Elis (" + str(answer\_part\_index).zfill(2) + ").mp3")

            response.stream\_to\_file(filename)

            # Maintenant que l'on a reçu notre réponse de l'audio de la prochaine phrase, on va attendre "await"

            # que la phrase précédente qui est entrain d'être lu soit fini d'être dite.

            await task\_player

            # Et on va maintenant faire jouer la nouvelle phrase pour l'entendre dans le haut parleur

            task\_player = asyncio.create\_task(asyncio.to\_thread(playsound, filename))

            await asyncio.sleep(0) # cette ligne est seulement là pour forcer à commencer la lecture tout de suite

            # et on ajoute 1 à notre variable qui compte on a lu combien de phrase.

            answer\_part\_index += 1

        # En sortant de notre boucle qui lit chaque phrase, on va s'assurer que l'on a bien lu la dernière phrase au complet.

        await task\_player

        # Maintenant, c'est le temps de demander à l'enfant ce qu'il veut dire

        # Choix 1: Code pour écrire la réponse de l'enfant.

        # question = input("enfant> ")

        # Choix 2: Code pour écouter le micro plutôt que le clavier.

        question = listen\_and\_transcribe();

        print("enfant> " + question)

        # Que l'on va ajouter à notre liste de "PROMPTs" avec le role user.

        prompts.append({"role": "user", "content": question })

        print()

        # C'est ici qu'on regarde si la réponse que l'on a eu c'est "quit" pour mettre fin à notre discussion

        shouldContinue = question != "quit"

        # Et justement, si on n'est pas entrain d'arrêter, alors on continu !

        if shouldContinue:

            # En commençant par tout mettre nos PROMPTs ensemble pour les envoyer à ChatGPT et lui demander sa réponse

            gpt\_messages = []

            # On va commencer par mettre nos messages "system" au début de nos PROMPTs

            for system\_instruction in prompt\_System:

                gpt\_messages.append({ "role": "system", "content": system\_instruction })

            # Puis on va ajouter les messages de assistant et de user dans la liste des PROMPTs.

            for prompt in prompts:

                gpt\_messages.append(prompt)

            # ...et c'est ici que l'on va aller demander à ChatGPT de prendre tous ces PROMPTs de notre conversations

            # ...et nous dire c'est quoi le prochain bon message !

            gpt\_client = client.chat.completions.create(

                #model="gpt-3.5-turbo",

                model="gpt-4-1106-preview",

                messages=gpt\_messages

            )

            # On met la réponse de ChatGPR dans la variable "answer", et on va ensuite recommencer notre boucle !

            answer = gpt\_client.choices[0].message.content

# Ici, c'est la première ligne de code qui sera exécuter dans notre application.

# C'est elle qui va aller exécuter la fonction principale, c'est à dire "main()" que l'on a défini au début de notre code.

asyncio.run(main())