Reconnaissance vocale en temps réel

Ce code Python implémente une reconnaissance vocale en temps réel utilisant l'API Google Cloud Speech-to-Text et la bibliothèque PyAudio. Il capture l'audio depuis le microphone, le transmet en continu à l'API Speech-to-Text et affiche le texte transcrit. La classe `MicrophoneStream` gère l'entrée audio, et la fonction `main` configure le client de reconnaissance vocale et traite le flux audio.

```
import os
import argparse
import io
import sys
import time

from google.cloud import speech

import pyaudio
from six.moves import queue


# Paramètres d'enregistrement audio
RATE = 16000
CHUNK = int(RATE / 10)  # 100ms


class MicrophoneStream(object):
    """Ouvre un flux d'enregistrement en tant que générateur produisant les blocs audio."""
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # Créer une interface audio utilisant PyAudio
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # L'API ne supporte actuellement que l'audio 1 canal (mono)
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            # Exécuter le flux audio de manière asynchrone pour remplir l'objet buffer.
            # Ceci est nécessaire pour que le buffer du périphérique d'entrée ne
```

```
        # déborde pas pendant que le thread appelant effectue des requêtes réseau, etc.
        stream_callback=self._fill_buffer,
    )
    self.closed = False
    self._buff = queue.Queue()

def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    """Collecte continuellement les données du flux audio, dans le buffer."""
    self._buff.put(in_data)
    return None, pyaudio.paContinue

def generator(self, record_seconds):
    start_time = time.time()
    while not self.closed and time.time() - start_time < record_seconds:
        # Utiliser un get() bloquant pour s'assurer qu'il y a au moins un bloc de
        # données, et arrêter l'itération si le bloc est None, indiquant la
        # fin du flux audio.
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        # Maintenant, consommer toutes les autres données encore en buffer.
        while True:
            try:
                chunk = self._buff.get(block=False)
                if chunk is None:
                    return
                data.append(chunk)
            except queue.Empty:
                break

        yield b''.join(data)

def close(self):
    self.closed = True
    # Signaler au générateur de terminer afin que la méthode
    # streaming recognize du client ne bloque pas la terminaison du processus.
    self._buff.put(None)
    self._audio_stream.close()
```

```
        self._audio_interface.terminate()

    def __enter__(self):
        return self

    def __exit__(self, type, value, traceback):
        self.close()


def main(record_seconds=10, language_code='fr-FR'):
    # Voir http://g.co/cloud/speech/docs/languages
    # pour une liste des langues prises en charge.
    # language_code = 'fr-FR'  # une balise de langue BCP-47

    client = speech.SpeechClient()
    config = speech.RecognitionConfig(
        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,
        sample_rate_hertz=RATE,
        language_code=language_code,
        model="latest_long",
    )

    streaming_config = speech.StreamingRecognitionConfig(
        config=config,
        interim_results=True)

    with MicrophoneStream(RATE, CHUNK) as stream:
        audio_generator = stream.generator(record_seconds)
        requests = (speech.StreamingRecognizeRequest(audio_content=content)
                    for content in audio_generator)

        responses = client.streaming_recognize(streaming_config, requests)

        # Maintenant, utiliser les réponses de transcription.
        transcript = ""
        for response in responses:
            print(response)
            # Une fois la transcription terminée, afficher le résultat.
            for result in response.results:
                if result.is_final:
```

```
                alternative = result.alternatives[0]
                transcript += alternative.transcript + " "
        print(u'Transcription : {}'.format(transcript))


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Reconnaissance vocale en temps réel avec durée ajustable.")
    parser.add_argument('--duration', type=int, default=10, help="Durée de l'enregistrement en secondes.")
    parser.add_argument('--language_code', type=str, default='fr-FR', help="Code de langue pour la transcripti
    args = parser.parse_args()
    print("Veuillez parler...")
    main(record_seconds=args.duration, language_code=args.language_code)
```