Real-time Speech Recognition

This Python code implements real-time speech recognition using Google Cloud Speech-to-Text API and the PyAudio library. It captures audio from the microphone, streams it to the Speech-to-Text API, and prints the transcribed text. The `MicrophoneStream` class handles audio input, and the `main` function sets up the speech recognition client and processes the audio stream.

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

from google.cloud import speech

import pyaudio
from six.moves import queue


# Audio recording parameters
RATE = 16000
CHUNK = int(RATE / 10)  # 100ms


class MicrophoneStream(object):
    """Opens a recording stream as a generator yielding the audio chunks."""
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # Create a audio interface using PyAudio
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # The API currently only supports 1-channel (mono) audio
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            # Run the audio stream asynchronously to fill the buffer object.
            # This is necessary so that the input device's buffer doesn't
```

```
        # overflow while the calling thread makes network requests, 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):
    """Continuously collect data from the audio stream, into the 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:
        # Use a blocking get() to ensure there's at least one chunk of
        # data, and stop iteration if the chunk is None, indicating the
        # end of the audio stream.
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        # Now consume whatever other data's still buffered.
        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
    # Signal the generator to terminate so that the client's
    # streaming recognize method will not block the process termination.
    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='en-US'):
    # See http://g.co/cloud/speech/docs/languages
    # for a list of supported languages.
    # language_code = 'en-US'  # a BCP-47 language tag

    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)


        # Now, put the transcription responses to use.
        transcript = ""
        for response in responses:
            print(response)
            # Once the transcription is done, print the result.
            for result in response.results:
```

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


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Real-time speech recognition with adjustable duration.")
    parser.add_argument('--duration', type=int, default=10, help="Duration of recording in seconds.")
    parser.add_argument('--language_code', type=str, default='en-US', help="Language code for transcription.")
    args = parser.parse_args()
    print("Please speak...")
    main(record_seconds=args.duration, language_code=args.language_code)
```