# Spracherkennung in Echtzeit

Dieser Python-Code implementiert eine Echtzeit-Spracherkennung mit der Google Cloud Speech-to-Text API und der PyAudio-Bibliothek. Er nimmt Audio von einem Mikrofon auf, streamt es an die Speech-to-Text API und gibt den transkribierten Text aus. Die Klasse `MicrophoneStream` verarbeitet die Audioeingabe, und die Funktion `main` richtet den Spracherkennungs-Client ein und verarbeitet den Audiostream.

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

from google.cloud import speech

import pyaudio
from six.moves import queue


# Audioaufnahmeparameter
RATE = 16000
CHUNK = int(RATE / 10)  # 100ms


class MicrophoneStream(object):
    """Öffnet einen Aufnahmestream als Generator, der die Audio-Chunks liefert."""
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # Erstellt eine Audio-Schnittstelle mit PyAudio
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # Die API unterstützt derzeit nur 1-Kanal (Mono)-Audio
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            # Führt den Audiostream asynchron aus, um das Pufferobjekt zu füllen.
            # Dies ist notwendig, damit der Puffer des Eingabegeräts nicht
```

```
        # überläuft, während der aufrufende Thread Netzwerkaufforderungen usw. ausführt.
        stream_callback=self._fill_buffer,
    )
    self.closed = False
    self._buff = queue.Queue()

def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    """Sammelt kontinuierlich Daten aus dem Audiostream in den Puffer."""
    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:
        # Verwendet ein blockierendes get(), um sicherzustellen, dass mindestens ein Chunk
        # an Daten vorhanden ist, und stoppt die Iteration, wenn der Chunk None ist, was das
        # Ende des Audiostreams anzeigt.
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        # Verbraucht nun alle anderen Daten, die noch gepuffert sind.
        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
    # Signalisiert dem Generator, die Beendigung, damit die
    # Streaming-Erkennungsmethode des Clients die Prozessbeendigung nicht blockiert.
    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'):
    # Siehe http://g.co/cloud/speech/docs/languages
    # für eine Liste der unterstützten Sprachen.
    # language_code = 'en-US'  # ein BCP-47 Sprachtag

    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)

        # Nun, die Transkriptionsantworten verwenden.
        transcript = ""
        for response in responses:
            print(response)
            # Sobald die Transkription abgeschlossen ist, wird das Ergebnis ausgegeben.
            for result in response.results:
                if result.is_final:
```

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


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description="Echtzeit-Spracherkennung mit einstellbarer Dauer.")
    parser.add_argument('--duration', type=int, default=10, help="Dauer der Aufnahme in Sekunden.")
    parser.add_argument('--language_code', type=str, default='en-US', help="Sprachcode für die Transkription.'
    args = parser.parse_args()
    print("Bitte sprechen...")
    main(record_seconds=args.duration, language_code=args.language_code)
```