

# الحقيقي الوقت في الكلام على التعرف

تطبيقات برمجة واجهة باستخدام الحقيقي الوقت في الكلام على التعرف بتنفيذ بايثون بلغة الكود هذا يقوم  
برمجة واجهة إلى ويثه الميكروفون، من الصوت بالتقاط الكود يقوم . مكتبة و .  
main الدالة وتقوم الصوت، إدخال مع MicrophoneStream فئة تتعامل المُنسخ. النص ويطبّع، تطبيقات  
الصوت. تدفق ومعالجة الكلام على التعرف عميل بإعداد

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

from google.cloud import speech

import pyaudio
from six.moves import queue

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

class MicrophoneStream(object):
    """
    """
    def __init__(self, rate, chunk):
        self._rate = rate
        self._chunk = chunk

        # PyAudio
        self._audio_interface = pyaudio.PyAudio()
        self._audio_stream = self._audio_interface.open(
            format=pyaudio.paInt16,
            # ( )
            # https://goo.gl/z726ff
            channels=1, rate=self._rate,
            input=True, frames_per_buffer=self._chunk,
            #
            #
```

```

        #
        stream_callback=self._fill_buffer,
    )
    self.closed = False
    self._buff = queue.Queue()

def _fill_buffer(self, in_data, frame_count, time_info, status_flags):
    """
    """
    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:
        #
        #
        #
        chunk = self._buff.get()
        if chunk is None:
            return
        data = [chunk]

        #
        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
    #
    #
    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'):
    # http://g.co/cloud/speech/docs/languages
    #
    # language_code = 'en-US' # 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)

    #
    transcript = ""
    for response in responses:
        print(response)
    #
    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= " ". (
    parser.add_argument('--duration', type=int, default=10, help= " ". (
    parser.add_argument('--language_code', type=str, default='en-US', help= " ". (
    args = parser.parse_args()
    print( " "... (
    main(record_seconds=args.duration, language_code=args.language_code)

```