**목차**

**1. 구글 STT(Speech-To-Text) API 사용**

**2. 구글 TTS(Text-To-Speech) API 사용**

**3. 느낀 점**

**4. 코드**

**1. 구글 STT(Speech-To-Text) API 사용**

① 환경 설정

아나콘다 설정  
- “stt1”가상환경 파일 생성 후 사용(파이썬 3.6.9 버전 사용)  
  
-> 생성한 가상환경(stt1)에 접속

-구글 클라우드 플랫폼 프로젝트 생성 후, STT API 다운로드와 공개 키 발급  
  
->구글 플랫폼 사용 위해, 발급받은 구글 플랫폼 공개 키 등록(구글 크레덴셜 등록/ 터미널 킬때마다 재등록해줘야함 ->환경변수에 추가시 생략 가능)

-구글에서 제공하는 기능 사용 위해, SDK 설치  
->SDK shell 실행 후, 내 프로젝트와 연결

  
->클라이언트 라이브러리 설치하기

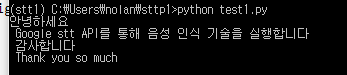
  
->구글 클라우드 스피치 설치하기

  
->서비스 계정 등록하기

  
->마이크사용 위해 파이오디오 설치

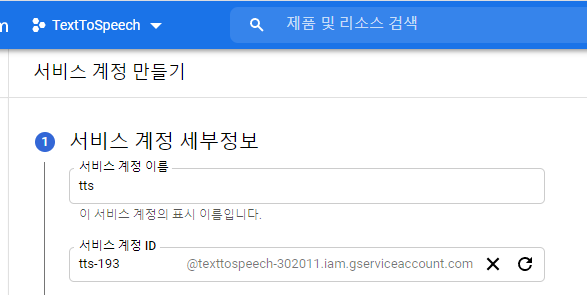
② 음성 -> 텍스트 변환 (test1.py)

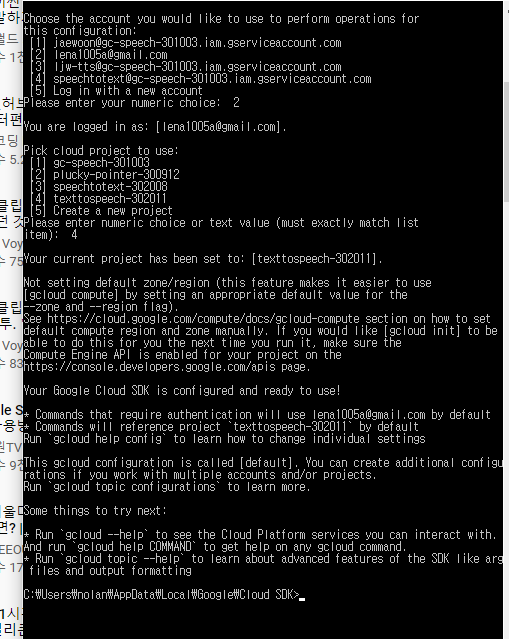
첨부한 test1.py코드 실행



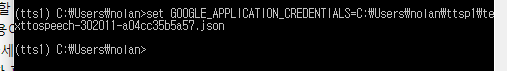
**2. 구글 TTS(Text-To-Speech) API 사용**

실습 앞서 TextToSpeech 이름의 새 프로젝트를 생성  
-> TTS API 설치 (키 발급)



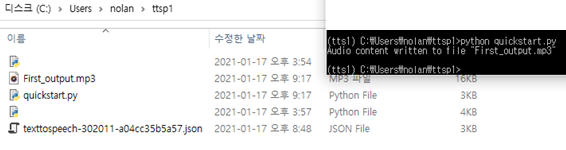
  
-> SDK실행(새로 생성한 프로젝트로 로그인하여 접속)

  
->생성한 가상환경(tts1)에 접속 (파이썬 3.6.9버전)

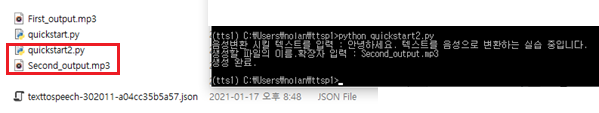
  
->구글 플랫폼 사용 위해, 발급받은 구글 플랫폼 공개 키 등록(구글 크레덴셜 등록/ 터미널 킬때마다 재등록해줘야함 ->환경변수에 추가시 생략 가능)

  
->클라이언트 라이브러리 설치하기

  
->서비스 계정 등록하기

  
-> quickstart.py실행   
->코드 내용에 있는"Hi. I'm Jae Woon, Practicing Artificial Intelligence" 텍스트를  
음성파일 First\_output.mp3로 변환

=====================================  
\*\*한글 변환법 : 40번째 코드   
language\_code="en-US"  
->  language\_code="ko-KR" 로 변경  
===============================================

  
-> quickstart2.py실행  
->음성으로 변환시킬 텍스트와 원하는 파일명으로 생성

**3. 느낀 점**

이번엔 STT와 TTS를 동시에 사용할 수 있는 코드 고안하기

**4. 코드**

① 음성 -> 텍스트 변환 (test1.py)  
#!/usr/bin/env python

# Copyright 2017 Google Inc. All Rights Reserved.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#      http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""Google Cloud Speech API sample application using the streaming API.

NOTE: This module requires the additional dependency `pyaudio`. To install

using pip:

    pip install pyaudio

Example usage:

    python transcribe\_streaming\_mic.py

"""

# [START speech\_transcribe\_streaming\_mic]

from \_\_future\_\_ import division

import re

import sys

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 thread-safe buffer of audio data

        self.\_buff = queue.Queue()

        self.closed = True

    def \_\_enter\_\_(self):

        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/z757pE

            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

        return self

    def \_\_exit\_\_(self, type, value, traceback):

        self.\_audio\_stream.stop\_stream()

        self.\_audio\_stream.close()

        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\_interface.terminate()

    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):

        while not self.closed:

            # 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 listen\_print\_loop(responses):

    """Iterates through server responses and prints them.

    The responses passed is a generator that will block until a response

    is provided by the server.

    Each response may contain multiple results, and each result may contain

    multiple alternatives; for details, see https://goo.gl/tjCPAU.  Here we

    print only the transcription for the top alternative of the top result.

    In this case, responses are provided for interim results as well. If the

    response is an interim one, print a line feed at the end of it, to allow

    the next result to overwrite it, until the response is a final one. For the

    final one, print a newline to preserve the finalized transcription.

    """

    num\_chars\_printed = 0

    for response in responses:

        if not response.results:

            continue

        # The `results` list is consecutive. For streaming, we only care about

        # the first result being considered, since once it's `is\_final`, it

        # moves on to considering the next utterance.

        result = response.results[0]

        if not result.alternatives:

            continue

        # Display the transcription of the top alternative.

        transcript = result.alternatives[0].transcript

        # Display interim results, but with a carriage return at the end of the

        # line, so subsequent lines will overwrite them.

        #

        # If the previous result was longer than this one, we need to print

        # some extra spaces to overwrite the previous result

        overwrite\_chars = ' ' \* (num\_chars\_printed - len(transcript))

        if not result.is\_final:

            sys.stdout.write(transcript + overwrite\_chars + '\r')

            sys.stdout.flush()

            num\_chars\_printed = len(transcript)

        else:

            print(transcript + overwrite\_chars)

            # Exit recognition if any of the transcribed phrases could be

            # one of our keywords.

            if re.search(r'\b(exit|quit)\b', transcript, re.I):

                print('Exiting..')

                break

            num\_chars\_printed = 0

def main():

    # See http://g.co/cloud/speech/docs/languages

    # for a list of supported languages.

    language\_code = 'ko-KR'  # a BCP-47 language tag

    client = speech.SpeechClient()

    config = speech.RecognitionConfig(

        encoding=speech.RecognitionConfig.AudioEncoding.LINEAR16,

        sample\_rate\_hertz=RATE,

        language\_code=language\_code)

    streaming\_config = speech.StreamingRecognitionConfig(

        config=config,

        interim\_results=True)

    with MicrophoneStream(RATE, CHUNK) as stream:

        audio\_generator = stream.generator()

        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.

        listen\_print\_loop(responses)

if \_\_name\_\_ == '\_\_main\_\_':

    main()

# [END speech\_transcribe\_streaming\_mic]

② quickstart.py  
#!/usr/bin/env python

# Copyright 2018 Google Inc. All Rights Reserved.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#      http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""Google Cloud Text-To-Speech API sample application .

Example usage:

    python quickstart.py

"""

def run\_quickstart():

    # [START tts\_quickstart]

    """Synthesizes speech from the input string of text or ssml.

    Note: ssml must be well-formed according to:

        https://www.w3.org/TR/speech-synthesis/

    """

    from google.cloud import texttospeech

    # Instantiates a client

    client = texttospeech.TextToSpeechClient()

    # Set the text input to be synthesized

    synthesis\_input = texttospeech.SynthesisInput(text="Hi. I'm Jae Woon, Practicing Artificial Intelligence")

    # Build the voice request, select the language code ("en-US") and the ssml

    # voice gender ("neutral")

    voice = texttospeech.VoiceSelectionParams(

        language\_code="en-US", ssml\_gender=texttospeech.SsmlVoiceGender.NEUTRAL

    )

    # Select the type of audio file you want returned

    audio\_config = texttospeech.AudioConfig(

        audio\_encoding=texttospeech.AudioEncoding.MP3

    )

    # Perform the text-to-speech request on the text input with the selected

    # voice parameters and audio file type

    response = client.synthesize\_speech(

        input=synthesis\_input, voice=voice, audio\_config=audio\_config

    )

    # The response's audio\_content is binary.

    with open("First\_output.mp3", "wb") as out:

        # Write the response to the First\_output file.

        out.write(response.audio\_content)

        print('Audio content written to file "First\_output.mp3"')

    # [END tts\_quickstart]

if \_\_name\_\_ == "\_\_main\_\_":

    run\_quickstart()

③ quickstart2.py  
#!/usr/bin/env python

# Copyright 2018 Google Inc. All Rights Reserved.

#

# Licensed under the Apache License, Version 2.0 (the "License");

# you may not use this file except in compliance with the License.

# You may obtain a copy of the License at

#

#      http://www.apache.org/licenses/LICENSE-2.0

#

# Unless required by applicable law or agreed to in writing, software

# distributed under the License is distributed on an "AS IS" BASIS,

# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

# See the License for the specific language governing permissions and

# limitations under the License.

"""Google Cloud Text-To-Speech API sample application .

Example usage:

    python quickstart.py

"""

def run\_quickstart():

    # [START tts\_quickstart]

    """Synthesizes speech from the input string of text or ssml.

    Note: ssml must be well-formed according to:

        https://www.w3.org/TR/speech-synthesis/

    """

    from google.cloud import texttospeech

    # Instantiates a client

    client = texttospeech.TextToSpeechClient()

    sentence = input("음성변환 시킬 텍스트를 입력 : ")

    # Set the text input to be synthesized

    synthesis\_input = texttospeech.SynthesisInput(text=sentence)

    # Build the voice request, select the language code ("en-US") and the ssml

    # voice gender ("neutral")

    voice = texttospeech.VoiceSelectionParams(

        language\_code="ko-KR", ssml\_gender=texttospeech.SsmlVoiceGender.NEUTRAL

    )

    # Select the type of audio file you want returned

    audio\_config = texttospeech.AudioConfig(

        audio\_encoding=texttospeech.AudioEncoding.MP3

    )

    # Perform the text-to-speech request on the text input with the selected

    # voice parameters and audio file type

    response = client.synthesize\_speech(

        input=synthesis\_input, voice=voice, audio\_config=audio\_config

    )

    OUTPUT = input("생성할 파일의 이름.확장자 입력 : ")

    # The response's audio\_content is binary.

    with open(OUTPUT, "wb") as out:

        # Write the response to the First\_output file.

        out.write(response.audio\_content)

        print('생성 완료.')

    # [END tts\_quickstart]

if \_\_name\_\_ == "\_\_main\_\_":

    run\_quickstart()