Recognize and synthesize speech

Azure AI Speech is a service that provides speech-related functionality, including:

- A *speech-to-text* API that enables you to implement speech recognition (converting audible spoken words into text).
- A *text-to-speech* API that enables you to implement speech synthesis (converting text into audible speech).

In this exercise, you'll use both of these APIs to implement a speaking clock application.

While this exercise is based on Python, you can develop speech applications using multiple language-specific SDKs; including:

- Azure AI Speech SDK for Python
- Azure AI Speech SDK for .NET
- Azure AI Speech SDK for JavaScript

This exercise takes approximately **30** minutes.

> **NOTE** This exercise is designed to be completed in the Azure cloud shell, where direct access to your computer's sound hardware is not supported. The lab will therefore use audio files for speech input and output streams. The code to achieve the same results using a mic and speaker is provided for your reference.

Create an Azure AI Speech resource

Let's start by creating an Azure AI Speech resource.

1. Open the Azure portal at `https://portal.azure.com`, and sign in using the Microsoft account associated with your Azure subscription.
2. In the top search field, search for **Speech service**. Select it from the list, then select **Create**.
3. Provision the resource using the following settings:
    - **Subscription**: *Your Azure subscription*.
    - **Resource group**: *Choose or create a resource group*.
    - **Region**:*Choose any available region*
    - **Name**: *Enter a unique name*.
    - **Pricing tier**: Select **F0** (*free*), or **S** (*standard*) if F is not available.
4. Select **Review + create**, then select **Create** to provision the resource.
5. Wait for deployment to complete, and then go to the deployed resource.
6. View the **Keys and Endpoint** page in the **Resource Management** section. You will need the information on this page later in the exercise.

Prepare and configure the speaking clock app

1. Leaving the **Keys and Endpoint** page open, use the **[>_]** button to the right of the search bar at the top of the page to create a new Cloud Shell in the Azure portal, selecting a ***PowerShell*** environment. The cloud shell provides a command line interface in a pane at the bottom of the Azure portal.

    > **Note**: If you have previously created a cloud shell that uses a *Bash* environment, switch it to ***PowerShell***.

2. In the cloud shell toolbar, in the **Settings** menu, select **Go to Classic version** (this is required to use the code editor).

    Ensure you've switched to the classic version of the cloud shell before continuing.

3. In the PowerShell pane, enter the following commands to clone the GitHub repo for this exercise:

Code Copy

```
rm -r mslearn-ai-language -f
git clone https://github.com/microsoftlearning/mslearn-ai-language
```

> **Tip**: As you enter commands into the cloudshell, the ouput may take up a large amount of the screen buffer. You can clear the screen by entering the `cls` command to make it easier to focus on each task.

4. After the repo has been cloned, navigate to the folder containing the speaking clock application code files:

Code Copy

```
cd mslearn-ai-language/Labfiles/07-speech/Python/speaking-clock
```

5. In the command line pane, run the following command to view the code files in the **speaking-clock** folder:

Code Copy

```
ls -a -l
```

The files include a configuration file (**.env**) and a code file (**speaking-clock.py**). The audio files your application will use are in the **audio** subfolder.

6. Create a Python virtual environment and install the Azure AI Speech SDK package and other required packages by running the following command:

Code Copy

```
python -m venv labenv
./labenv/bin/Activate.ps1
pip install -r requirements.txt azure-cognitiveservices-speech==1.42.0
```

7. Enter the following command to edit the configuration file:

Code Copy

```
code .env
```

The file is opened in a code editor.

8. Update the configuration values to include the **region** and a **key** from the Azure AI Speech resource you created (available on the **Keys and Endpoint** page for your Azure AI Translator resource in the Azure portal).

9. After you've replaced the placeholders, use the **CTRL+S** command to save your changes and then use the **CTRL+Q** command to close the code editor while keeping the cloud shell command line open.

Add code to use the Azure AI Speech SDK

> **Tip**: As you add code, be sure to maintain the correct indentation.

1. Enter the following command to edit the code file that has been provided:

Code Copy

```
code speaking-clock.py
```

2. At the top of the code file, under the existing namespace references, find the comment **Import namespaces**. Then, under this comment, add the following language-specific code to import the namespaces you will need to use the Azure AI Speech SDK:

Code Copy

```
# Import namespaces
from azure.core.credentials import AzureKeyCredential
import azure.cognitiveservices.speech as speech_sdk
```

3. In the **main** function, under the comment **Get config settings**, note that the code loads the key and region you defined in the configuration file.

4. Find the comment **Configure speech service**, and add the following code to use the AI Services key and your region to configure your connection to the Azure AI Services Speech endpoint:

Code Copy

```
# Configure speech service
speech_config = speech_sdk.SpeechConfig(speech_key, speech_region)
print('Ready to use speech service in:', speech_config.region)
```

5. Save your changes (*CTRL+S*), but leave the code editor open.

Run the app

So far, the app doesn't do anything other than connect to your Azure AI Speech service, but it's useful to run it and check that it works before adding speech functionality.

1. In the command line, enter the following command to run the speaking clock app:

Code Copy

```
python speaking-clock.py
```

The code should display the region of the speech service resource the application will use. A successful run indicates that the app has connected to your Azure AI Speech resource.

Add code to recognize speech

Now that you have a **SpeechConfig** for the speech service in your project's Azure AI Services resource, you can use the **Speech-to-text** API to recognize speech and transcribe it to text.

In this procedure, the speech input is captured from an audio file, which you can play here:

1. In the code file, note that the code uses the **TranscribeCommand** function to accept spoken input. Then in the **TranscribeCommand** function, find the comment **Configure speech recognition** and add the appropriate code below to create a **SpeechRecognizer** client that can be used to recognize and transcribe speech from an audio file:

Code Copy

```
# Configure speech recognition
current_dir = os.getcwd()
audioFile = current_dir + '/time.wav'
audio_config = speech_sdk.AudioConfig(filename=audioFile)
speech_recognizer = speech_sdk.SpeechRecognizer(speech_config, audio_config)
```

2. In the **TranscribeCommand** function, under the comment **Process speech input**, add the following code to listen for spoken input, being careful not to replace the code at the end of the function that returns the command:

Code Copy

```
# Process speech input
print("Listening...")
speech = speech_recognizer.recognize_once_async().get()
if speech.reason == speech_sdk.ResultReason.RecognizedSpeech:
    command = speech.text
    print(command)
else:
    print(speech.reason)
    if speech.reason == speech_sdk.ResultReason.Canceled:
        cancellation = speech.cancellation_details
        print(cancellation.reason)
        print(cancellation.error_details)
```

3. Save your changes (*CTRL+S*), and then in the command line below the code editor, re-run the program:
4. Review the output, which should successfully "hear" the speech in the audio file and return an appropriate response (note that your Azure cloud shell may be running on a server that is in a different time-zone to yours!)

> **Tip**: If the SpeechRecognizer encounters an error, it produces a result of "Cancelled". The code in the application will then display the error message. The most likely cause is an incorrect region value in the configuration file.

Synthesize speech

Your speaking clock application accepts spoken input, but it doesn't actually speak! Let's fix that by adding code to synthesize speech.

Once again, due to the hardware limitations of the cloud shell we'll direct the synthesized speech output to a file.

1. In the code file, note that the code uses the **TellTime** function to tell the user the current time.
2. In the **TellTime** function, under the comment **Configure speech synthesis**, add the following code to create a **SpeechSynthesizer** client that can be used to generate spoken output:

Code Copy

```
# Configure speech synthesis
output_file = "output.wav"
speech_config.speech_synthesis_voice_name = "en-GB-RyanNeural"
audio_config = speech_sdk.audio.AudioConfig(filename=output_file)
speech_synthesizer = speech_sdk.SpeechSynthesizer(speech_config, audio_config,)
```

3. In the **TellTime** function, under the comment **Synthesize spoken output**, add the following code to generate spoken output, being careful not to replace the code at the end of the function that prints the response:

```
# Synthesize spoken output
speak = speech_synthesizer.speak_text_async(response_text).get()
if speak.reason != speech_sdk.ResultReason.SynthesizingAudioCompleted:
    print(speak.reason)
else:
    print("Spoken output saved in " + output_file)
```

4. Save your changes (*CTRL+S*) and re-run the program, which should indicate that the spoken output was saved in a file.

5. If you have a media player capable of playing .wav audio files, download the file that was generated by entering the following command:

```
download ./output.wav
```

The download command creates a popup link at the bottom right of your browser, which you can select to download and open the file.

The file should sound similar to this:

Use Speech Synthesis Markup Language

Speech Synthesis Markup Language (SSML) enables you to customize the way your speech is synthesized using an XML-based format.

1. In the **TellTime** function, replace all of the current code under the comment **Synthesize spoken output** with the following code (leave the code under the comment **Print the response**):

```
# Synthesize spoken output
responseSsml = " \
   <speak version='1.0' xmlns='http://www.w3.org/2001/10/synthesis' xml:lang='en-US'> \
       <voice name='en-GB-LibbyNeural'> \
           {} \
           <break strength='weak'/> \
           Time to end this lab! \
       </voice> \
   </speak>".format(response_text)
speak = speech_synthesizer.speak_ssml_async(responseSsml).get()
if speak.reason != speech_sdk.ResultReason.SynthesizingAudioCompleted:
   print(speak.reason)
else:
   print("Spoken output saved in " + output_file)
```

2. Save your changes and re-run the program, which should once again indicate that the spoken output was saved in a file.
3. Download and play the generated file, which should sound similar to this:

Clean up

If you've finished exploring Azure AI Speech, you should delete the resources you have created in this exercise to avoid incurring unnecessary Azure costs.

1. Close the Azure cloud shell pane
2. In the Azure portal, browse to the Azure AI Speech resource you created in this lab.
3. On the resource page, select **Delete** and follow the instructions to delete the resource.

What if you have a mic and speaker?

In this exercise, the Azure Cloud Shell environment we used doesn't support audio hardware, so you used audio files for the speech input and output. Let's see how the code can be modified to use audio hardware if you have it available.

Using speech recognition with a microphone

If you have a mic, you can use the following code to capture spoken input for speech recognition:

Code Copy

```
# Configure speech recognition
audio_config = speech_sdk.AudioConfig(use_default_microphone=True)
speech_recognizer = speech_sdk.SpeechRecognizer(speech_config, audio_config)
print('Speak now...')

# Process speech input
speech = speech_recognizer.recognize_once_async().get()
if speech.reason == speech_sdk.ResultReason.RecognizedSpeech:
    command = speech.text
    print(command)
else:
    print(speech.reason)
    if speech.reason == speech_sdk.ResultReason.Canceled:
        cancellation = speech.cancellation_details
        print(cancellation.reason)
        print(cancellation.error_details)
```

> **Note**: The system default microphone is the default audio input, so you could also just omit the AudioConfig altogether!

Using speech synthesis with a speaker

If you have a speaker, you can use the following code to synthesize speech.

Code Copy

```
response_text = 'The time is {}:{:02d}'.format(now.hour,now.minute)

# Configure speech synthesis
speech_config.speech_synthesis_voice_name = "en-GB-RyanNeural"
audio_config = speech_sdk.audio.AudioOutputConfig(use_default_speaker=True)
speech_synthesizer = speech_sdk.SpeechSynthesizer(speech_config, audio_config)

# Synthesize spoken output
speak = speech_synthesizer.speak_text_async(response_text).get()
if speak.reason != speech_sdk.ResultReason.SynthesizingAudioCompleted:
    print(speak.reason)
```

> **Note**: The system default speaker is the default audio output, so you could also just omit the AudioConfig altogether!

More information

For more information about using the **Speech-to-text** and **Text-to-speech** APIs, see the Speech-to-text documentation and Text-to-speech documentation.