3.6 - Translate text with Azure AI Translator service

Overview

The Translator service enables you to create intelligent apps and services that can translate text between languages.

Learning objectives

After completing this module, you'll be able to:

- Provision a Translator resource
- Understand language detection, translation, and transliteration
- Specify translation options
- Define custom translations

Introduction

There are many commonly used languages throughout the world, and the ability to exchange information between speakers of different languages is often a critical requirement for global solutions.

The Azure AI Translator provides an API for translating text between 90 supported languages.

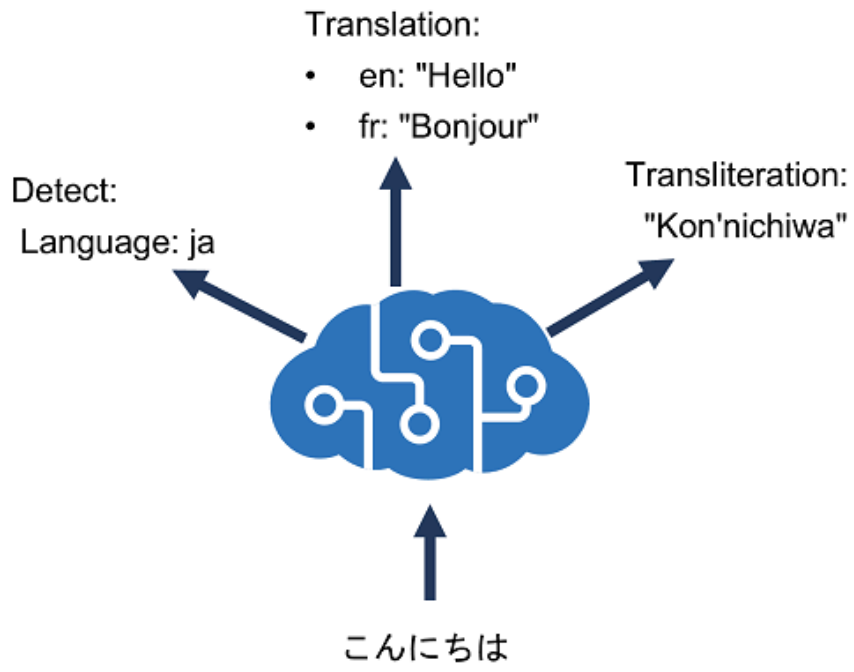
After completing this module, you'll be able to:

- Provision an Azure AI Translator resource
- Understand language detection, translation, and transliteration
- Specify translation options
- Define and run custom translations

Provision an Azure AI Translator resource

Azure AI Translator provides a multilingual text translation API that you can use for:

- Language detection.
- One-to-many translation.
- Script transliteration (converting text from its native script to an alternative script).



Azure resource for Azure AI Translator

To use the Azure AI Translator service, you must provision a resource for it in your Azure subscription. You can provision a single-service Azure AI Translator resource, or you can use the Text Analytics API in a multi-service Azure AI Services resource.

After you have provisioned a suitable resource in your Azure subscription, you can use the **location** where you deployed the resource and one of its **subscription keys** to call the Azure AI Translator APIs from your code. You can call the APIs by submitting requests in JSON format to the REST interface, or by using any of the available programming language-specific SDKs.

> Note: The code examples in the subsequent units in this module show the JSON requests and responses exchanged with the REST interface. When using an SDK, the JSON requests are abstracted by appropriate objects and methods that encapsulate the same data values. You'll get a chance to try the SDK for C# or Python for yourself in the exercise later in the module.

Understand language detection, translation, and transliteration

Let's explore the capabilities of **Azure AI Translator**. These capabilities include:

Language detection

You can use the **Detect** function of the REST API to detect the language in which text is written.

For example, you could submit the following text to the `https://api.cognitive.microsofttranslator.com/detect?api-version=3.0` endpoint using curl.

Here's the text we want to translate:

```
{ 'Text' : 'こんにちは' }
```

Here's a call using curl to the endpoint to detect the language of our text:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/detect?api-version=3.0" -H
"Ocp-Apim-Subscription-Region: <your-service-region>" -H "Ocp-Apim-Subscription-Key:
<your-key>" -H "Content-Type: application/json" -d "[{ 'Text' : 'こんにちは' }]
```

The response to this request looks as follows, indicating that the text is written in Japanese:

```
[
  {
    "language": "ja",
    "score": 1.0,
    "isTranslationSupported": true,
    "isTransliterationSupported": true


   }
]
```

Translation

To translate text from one language to another, use the **Translate** function; specifying a single **from** parameter to indicate the source language, and one or more **to** parameters to specify the

languages into which you want the text translated.

For example, you could submit the same JSON we previously used to detect the language, specifying a **from** parameter of **ja** (Japanese) and two **to** parameters with the values **en** (English) and **fr** (French). To do this, you'd call:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=ja&to=fr&to=en" -H "Ocp-Apim-Subscription-Key: <your-key>" -H "Ocp-Apim-Subscription-Region: <your-service-region>" -H "Content-Type: application/json; charset=UTF-8" -d "[{ 'Text' : 'こんにちは' }]"
```

This would produce the following result:

```
[
  {"translations":
    [
      {"text": "Hello", "to": "en"},
      {"text": "Bonjour", "to": "fr"}
    ]
  }
]
```

Transliteration

Our Japanese text is written using Hiragana script, so rather than translate it to a different language, you may want to transliterate it to a different script - for example to render the text in Latin script (as used by English language text).

To accomplish this, we can submit the Japanese text to the **Transliterate** function with a **fromScript** parameter of **Jpan** and a **toScript** parameter of **Latn**:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/transliterate?api-version=3.0&fromScript=Jpan&toScript=Latn" -H "Ocp-Apim-Subscription-Key: <your-key>" -H "Ocp-Apim-Subscription-Region: <your-service-region>" -H "Content-Type: application/json" -d "[{ 'Text' : 'こんにちは' }]"
```

The response would give you the following result:

```
[
    {
        "script": "Latn",
        "text": "Kon'nichiwa"
    }
]
```

Specify translation options

The **Translate** function of the API supports numerous parameters that affect the output.

Word alignment

In **written English (using Latin script)**, spaces are used to separate words. However, in some other languages (and more specifically, scripts) this is not always the case.

For example, translating "Smart Services" from **en** (English) to **zh** (Simplified Chinese) produces the result "智能服务", and it's difficult to understand the relationship between the characters in the source text and the corresponding characters in the translation. To resolve this problem, you can specify the **includeAlignment** parameter with a value of **true** in your call to produce the following result:

```
[
   {
      "translations":[
         {
            "text":"智能服务",
            "to":"zh-Hans",
            "alignment":{
               "proj":"0:4-0:1 6:13-2:3"
            }
         }
      ]
   }
]
```

These results tell us that **characters 0 to 4 in the source correspond to characters 0 to 1 in the translation**, while characters 6 to 13 in the source correspond to characters 2 to 3 in the translation.

Sentence length

Sometimes it might be useful to **know the length of a translation** (i.e., number of characters), for example to determine how best to display it in a user interface. You can get this information by setting the **includeSentenceLength** parameter to **true**.

For example, specifying this parameter when translating the English (**en**) text "Hello world" to French (**fr**) produces the following results:

```
[
   {
      "translations":[
         {
            "text":"Salut tout le monde",
            "to":"fr",
            "sentLen":{"srcSentLen":[12],"transSentLen":[20]}
         }
      ]
   }
]
```

Profanity filtering

Sometimes text contains profanities, which you might want to obscure or omit altogether in a translation. You can handle profanities by specifying the **profanityAction** parameter, which can have one of the following values:

- **NoAction**: Profanities are translated along with the rest of the text.
- **Deleted**: Profanities are omitted in the translation.
- **Marked**: Profanities are indicated using the technique indicated in the **profanityMarker** parameter (if supplied). The default value for this parameter is **Asterisk**, which replaces characters in profanities with "*". As an alternative, you can specify a **profanityMarker** value of **Tag**, which causes profanities to be enclosed in XML tags.

For example, translating the English (**en**) text "JSON is ██████ great!" (where the blocked out word is a profanity) to German (**de**) with a **profanityAction** of **Marked** and a **profanityMarker** of **Asterisk** produces the following result:

```
[
   {
      "translations":[
         {
            "text":"JSON ist *** erstaunlich.",
            "to":"de"
         }
      ]
   }
]
```

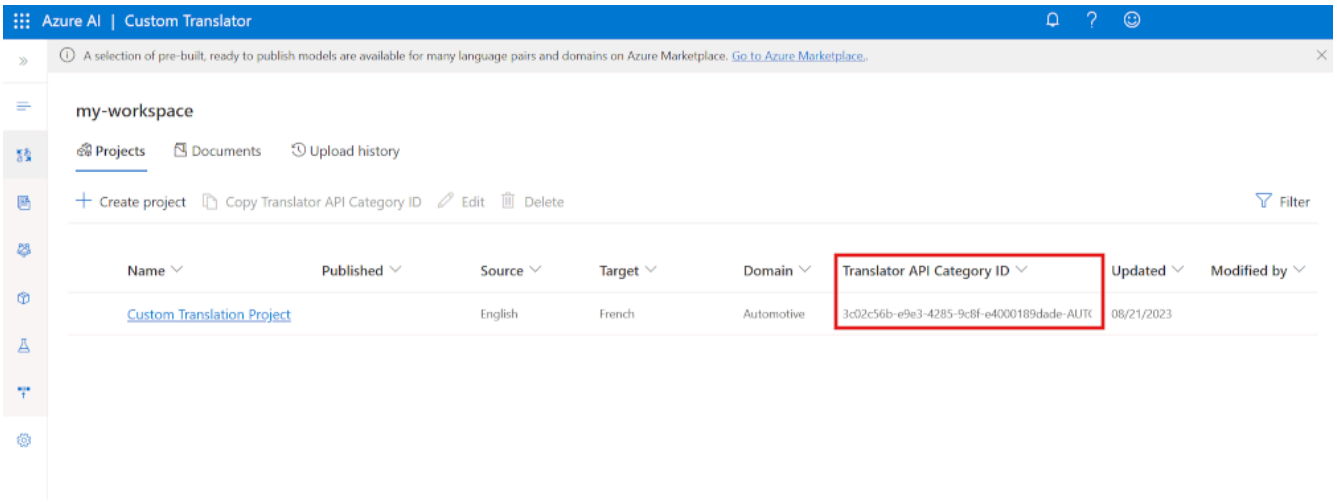
> Note: To learn more about the translation options, including some not described here, see the Azure AI Translator API documentation.

Define custom translations

While the default translation model used by Azure AI Translator is effective for general translation, you may need to develop a translation solution for businesses or industries in that have specific vocabularies of terms that require custom translation.

To solve this problem, you can create a custom model that maps your own sets of source and target terms for translation. To create a custom model, use the Custom Translator portal to:

1. Create a workspace linked to your Azure AI Translator resource.
2. Create a project.
3. Upload training data files and train a model.
4. Test your model and publish your model.
5. Make translation calls to the API.



Your custom model is assigned a unique **category Id** (highlighted in the screenshot), which you can specify in **translate** calls to your Azure AI Translator resource by using the **category** parameter, causing translation to be performed by your custom model instead of the default model.

How to call the API

To initiate a translation, you send a **POST** request to the following request URL:

```
https://api.cognitive.microsofttranslator.com/translate?api-version=3.0
```

Your request needs to include a couple of parameters:

- `api-version` : The required version of the API.
- `to` : The target language to translate to. For example: `to=fr` for French.
- `category` : Your **category Id**.

Your request must also include a number of required headers:

- `Ocp-Apim-Subscription-Key` . Header for your client key. For example: `Ocp-Apim-Subscription-Key=<your-client-key>` .
- `Content-Type` . The content type of the payload. The required format is: `Content-Type: application/json; charset=UTF-8` .

The request body should contain an array that includes a **JSON object with a `Text` property** that specifies the text that you want to translate:

```
[
    {"Text":"Where can I find my employee details?"}
]
```

There are different ways you can send your request to the API, including using the C#, Python, and curl. For instance, to make a quick call, you can send a POST request using curl:

```
curl -X POST "https://api.cognitive.microsofttranslator.com/translate?api-version=3.0&from=en&to=nl&category=<category-id>" -H "Ocp-Apim-Subscription-Key: <your-
```

```
key" -H "Content-Type: application/json; charset=UTF-8" -d "[{'Text':'Where can I find
my employee details?'}]"
```

The request above makes a call to translate a sentence from English to Dutch.

Response returned

The response returns a response code of `200` if the request was successful. It also returns a response body that contains the translated text, like this:

```
[
    {
        "translations":[
            {"text":"Waar vind ik mijn personeelsgegevens?","to":"nl"}
        ]
    }
]
```

If the request wasn't successful, then a number of different status codes may be returned depending on the error type, such as `400` (missing or invalid query parameters). See Response status codes for a full list of codes and their explanation.

> Note: For more information about custom translation, see Quickstart: Build, publish, and translate with custom models.

Exercise - Translate text with the Azure AI Translator service

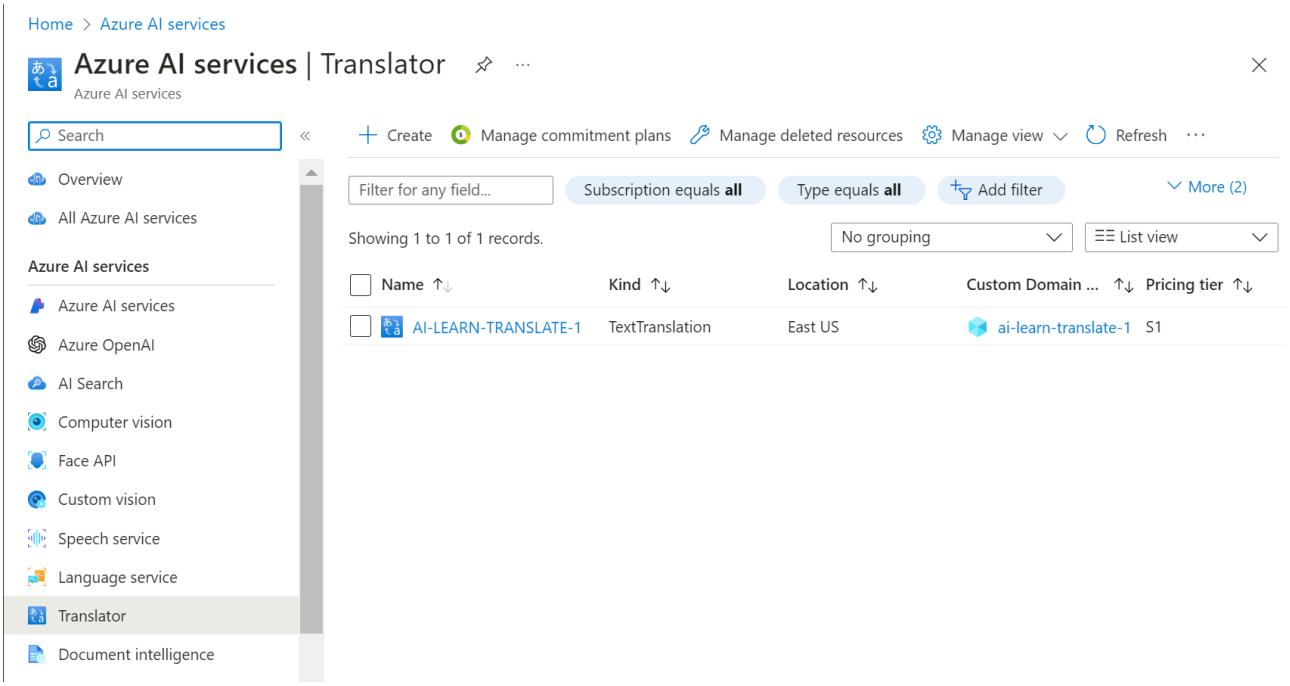
Azure AI Translator is a service that enables you to translate text between languages. In this exercise, you'll use it to create a simple app that translates input in any supported language to the target language of your choice.

Provision an *Azure AI Translator* resource

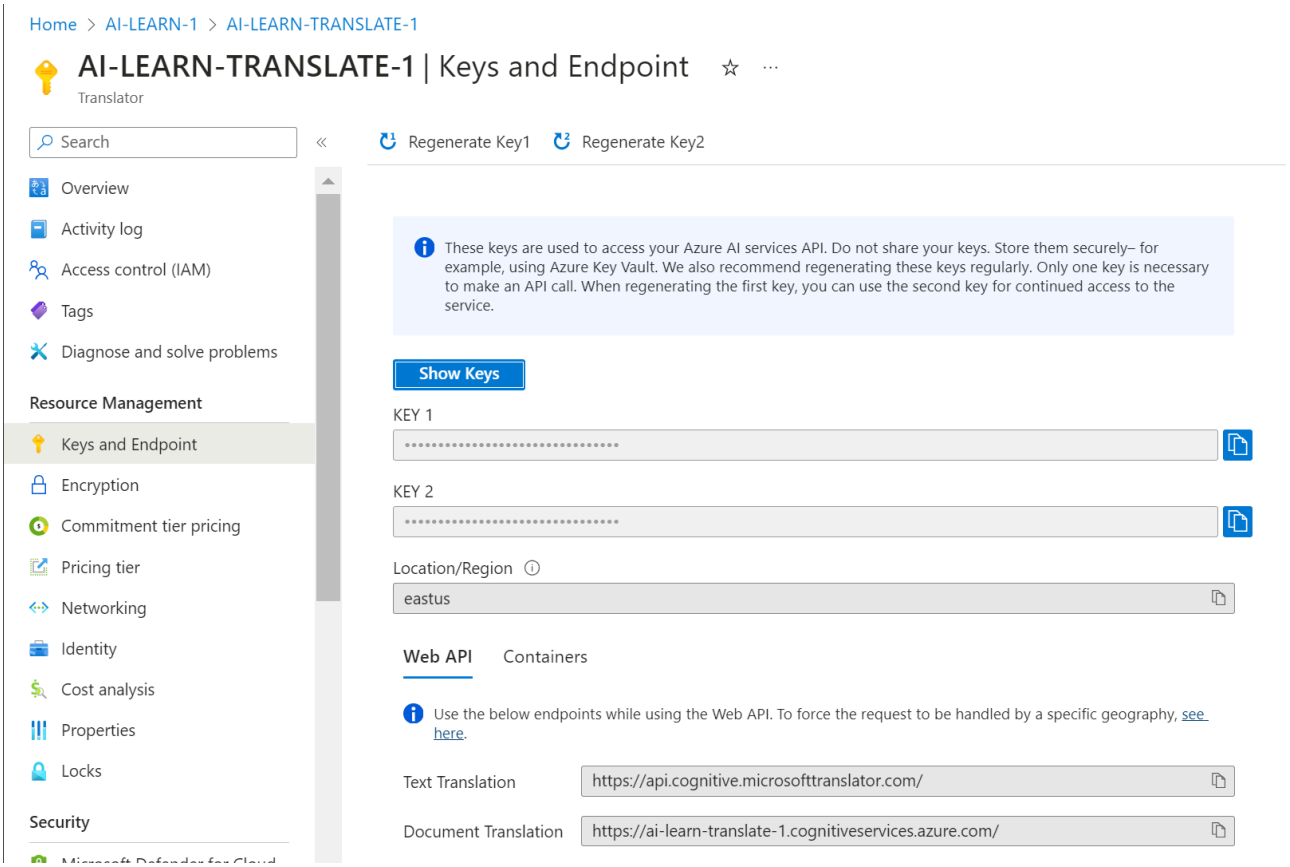
If you don't already have one in your subscription, you'll need to provision an **Azure AI Translator** resource.

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 search field at the top, search for **Azure AI services** and press **Enter**, then select **Create** under **Translator** in the results.
3. Create a resource with 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.
   - **Responsible AI Notice**: Agree.
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. You will need the information on this page later in the exercise.



Prepare to develop an app in Visual Studio Code

You'll develop your text translation app using Visual Studio Code. The code files for your app have been provided in a GitHub repo.

> **Tip**: If you have already cloned the **mslearn-ai-language** repo, open it in Visual Studio code. Otherwise, follow these steps to clone it to your development environment.

1. Start Visual Studio Code.

2. Open the palette (SHIFT+CTRL+P) and run a **Git: Clone** command to clone the `https://github.com/MicrosoftLearning/mslearn-ai-language` repository to a local folder (it doesn't matter which folder).
3. When the repository has been cloned, open the folder in Visual Studio Code.

> **Note**: If Visual Studio Code shows you a pop-up message to prompt you to trust the code you are opening, click on **Yes, I trust the authors** option in the pop-up.

4. Wait while additional files are installed to support the C# code projects in the repo.

> **Note**: If you are prompted to add required assets to build and debug, select **Not Now**.

Configure your application

Applications for both C# and Python have been provided. Both apps feature the same functionality. First, you'll complete some key parts of the application to enable it to use your Azure AI Translator resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/06b-translator-sdk** folder and expand the **CSharp** or **Python** folder depending on your language preference and the **translate-text** folder it contains. Each folder contains the language-specific code files for an app into which you're you're going to integrate Azure AI Translator functionality.
2. Right-click the **translate-text** folder containing your code files and open an integrated terminal. Then install the Azure AI Translator SDK package by running the appropriate command for your language preference:

   C#:

   ```
   dotnet add package Azure.AI.Translation.Text --version 1.0.0-beta.1
   ```

   Python:

   ```
   pip install azure-ai-translation-text==1.0.0b1
   ```

3. In the **Explorer** pane, in the **translate-text** folder, open the configuration file for your preferred language
   - **C#**: appsettings.json
   - **Python**: .env
4. Update the configuration values to include the **region** and a **key** from the Azure AI Translator resource you created (available on the **Keys and Endpoint** page for your Azure AI Translator resource in the Azure portal).

> **NOTE**: Be sure to add the *region* for your resource, not the endpoint!

5. Save the configuration file.

Add code to translate text

Now you're ready to use Azure AI Translator to translate text.

1. Note that the **translate-text** folder contains a code file for the client application:

- **C#**: Program.cs
- **Python**: translate.py (View here: translate.py)

Open the code file and at the top, 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 Text Analytics SDK:

C#: Programs.cs

```
// import namespaces using Azure; using Azure.AI.Translation.Text;
```

Python: translate.py (View here: translate.py)

```
# import namespaces from azure.ai.translation.text import * from
azure.ai.translation.text.models import InputTextItem
```

2. In the **Main** function, note that the existing code reads the configuration settings.
3. Find the comment **Create client using endpoint and key** and add the following code:

C#: Programs.cs

```
// Create client using endpoint and key AzureKeyCredential credential =
new(translatorKey); TextTranslationClient client = new(credential, translatorRegion);
```

Python: translate.py (View here: translate.py)

```
# Create client using endpoint and key credential = TranslatorCredential(translatorKey,
translatorRegion) client = TextTranslationClient(credential)
```

4. Find the comment **Choose target language** and add the following code, which uses the Text Translator service to return list of supported languages for translation, and prompts the user to select a language code for the target language.

C#: Programs.cs

```
// Choose target language Response<GetLanguagesResult> languagesResponse = await
client.GetLanguagesAsync(scope:"translation").ConfigureAwait(false); GetLanguagesResult
languages = languagesResponse.Value; Console.WriteLine($"{languages.Translation.Count}
languages available.\n(See https://learn.microsoft.com/azure/ai-
services/translator/language-support#translation)"); Console.WriteLine("Enter a target
language code for translation (for example, 'en'):"); string targetLanguage = "xx"; bool
languageSupported = false; while (!languageSupported) { targetLanguage =
Console.ReadLine(); if (languages.Translation.ContainsKey(targetLanguage)) {
languageSupported = true; } else { Console.WriteLine($"{targetLanguage} is not a
supported language."); } }
```

Python: translate.py (View here: translate.py)

```
# Choose target language languagesResponse = client.get_languages(scope="translation")
print("{} languages supported.".format(len(languagesResponse.translation))) print("(See
https://learn.microsoft.com/azure/ai-services/translator/language-support#translation)")
print("Enter a target language code for translation (for example, 'en'):")
targetLanguage = "xx" supportedLanguage = False while supportedLanguage == False:
targetLanguage = input() if targetLanguage in languagesResponse.translation.keys():
supportedLanguage = True else: print("{} is not a supported
language.".format(targetLanguage))
```

5. Find the comment **Translate text** and add the following code, which repeatedly prompts the user for text to be translated, uses the Azure AI Translator service to translate it to the target language (detecting the source language automatically), and displays the results until the user enters *quit*.

C#: Programs.cs

```
// Translate text string inputText = ""; while (inputText.ToLower() != "quit") { Console.WriteLine("Enter text to translate ('quit' to exit)"); inputText = Console.ReadLine(); if (inputText.ToLower() != "quit") { Response<IReadOnlyList<TranslatedTextItem>> translationResponse = await client.TranslateAsync(targetLanguage, inputText).ConfigureAwait(false); IReadOnlyList<TranslatedTextItem> translations = translationResponse.Value; TranslatedTextItem translation = translations[0]; string sourceLanguage = translation?.DetectedLanguage?.Language; Console.WriteLine($"'{inputText}' translated from {sourceLanguage} to {translation?.Translations[0].To} as '{translation?.Translations?[0]?.Text}'."); } }
```

Python: translate.py (View here: translate.py)

```
# Translate text inputText = "" while inputText.lower() != "quit": inputText = input("Enter text to translate ('quit' to exit):") if inputText != "quit": input_text_elements = [InputTextItem(text=inputText)] translationResponse = client.translate(content=input_text_elements, to=[targetLanguage]) translation = translationResponse[0] if translationResponse else None if translation: sourceLanguage = translation.detected_language for translated_text in translation.translations: print(f"'{inputText}' was translated from {sourceLanguage.language} to {translated_text.to} as '{translated_text.text}'.")
```

6. Save the changes to your code file.

Test your application

Now your application is ready to test.

1. In the integrated terminal for the **Translate text** folder, and enter the following command to run the program:
   - **C#**: `dotnet run`
   - **Python**: `python translate.py`

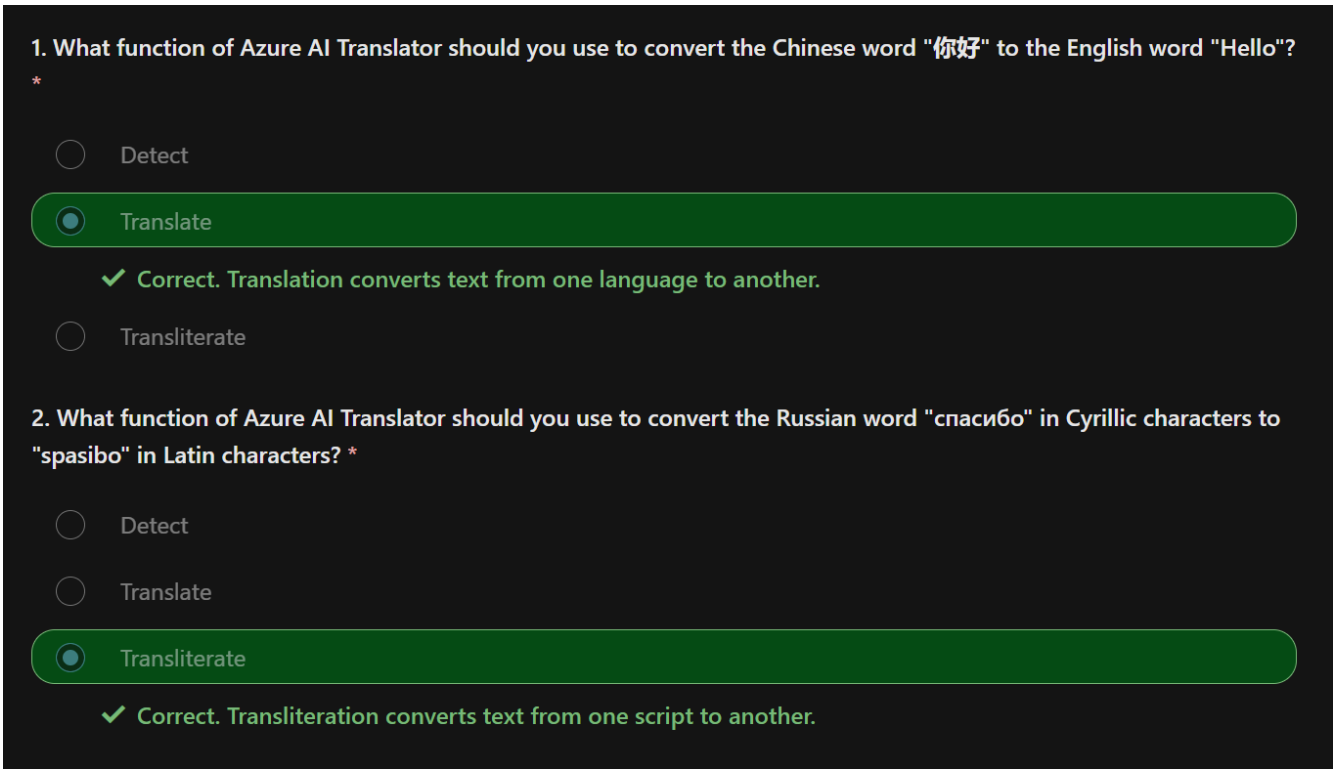
   > **Tip**: You can use the **Maximize panel size** (**^**) icon in the terminal toolbar to see more of the console text.

2. When prompted, enter a valid target language from the list displayed.
3. Enter a phrase to be translated (for example `This is a test` or `C'est un test`) and view the results, which should detect the source language and translate the text to the target language.
4. When you're done, enter `quit`. You can run the application again and choose a different target language.

Clean up

When you don't need your project anymore, you can delete the Azure AI Translator resource in the Azure portal.

Knowledge Check



Summary

In this module, you learned how to:

- Provision an Azure AI Translator resource
- Understand language detection, translation, and transliteration
- Specify translation options
- Define custom translations

To learn more about Azure AI Translator, see the Azure AI Translator documentation.

✍️ Compiled by Kenneth Leung (2025)