Generate and improve code with Azure OpenAI Service

The Azure OpenAI Service models can generate code for you using natural language prompts, fixing bugs in completed code, and providing code comments. These models can also explain and simplify existing code to help you understand what it does and how to improve it.

In scenario for this exercise, you will perform the role of a software developer exploring how to use generative AI to make coding tasks easier and more efficient. The techniques used in the exercise can be applied to other code files, programming languages, and use cases.

Provision an Azure OpenAI resource

If you don't already have one, provision an Azure OpenAI resource in your Azure subscription.

1. Sign into the **Azure portal** at `https://portal.azure.com`.
2. Create an **Azure OpenAI** resource with the following settings:
    - **Subscription**: *Select an Azure subscription that has been approved for access to the Azure OpenAI service*
    - **Resource group**: *Choose or create a resource group*
    - **Region**: *Make a **random** choice from any of the following regions**
        - Australia East
        - Canada East
        - East US
        - East US 2
        - France Central
        - Japan East
        - North Central US
        - Sweden Central
        - Switzerland North
        - UK South
    - **Name**: *A unique name of your choice*
    - **Pricing tier**: Standard S0

    * Azure OpenAI resources are constrained by regional quotas. The listed regions include default quota for the model type(s) used in this exercise. Randomly choosing a region reduces the risk of a single region reaching its quota limit in scenarios where you are sharing a subscription with other users. In the event of a quota limit being reached later in the exercise, there's a possibility you may need to create another resource in a different region.need to create another resource in a different region.

3. Wait for deployment to complete. Then go to the deployed Azure OpenAI resource in the Azure portal.

Deploy a model

Azure OpenAI provides a web-based portal named **Azure OpenAI Studio**, that you can use to deploy, manage, and explore models. You'll start your exploration of Azure OpenAI by using Azure OpenAI Studio to deploy a model.

1. On the **Overview** page for your Azure OpenAI resource, use the **Go to Azure OpenAI Studio** button to open Azure OpenAI Studio in a new browser tab.
2. In Azure OpenAI Studio, on the **Deployments** page, view your existing model deployments. If you don't already have one, create a new deployment of the **gpt-35-turbo-16k** model with the following settings:
    - **Model**: gpt-35-turbo-16k *(if the 16k model isn't available, choose gpt-35-turbo)*
    - **Model version**: Auto-update to default
    - **Deployment name**: *A unique name of your choice. You'll use this name later in the lab.*
    - **Advanced options**
        - **Content filter**: Default
        - **Deployment type**: Standard
        - **Tokens per minute rate limit**: 5K*
        - **Enable dynamic quota**: Enabled

    * A rate limit of 5,000 tokens per minute is more than adequate to complete this exercise while leaving capacity for other people using the same subscription.

Generate code in chat playground

Before using in your app, examine how Azure OpenAI can generate and explain code in the chat playground.

1. In the **Azure OpenAI Studio** at `https://oai.azure.com`, in the **Playground** section, select the **Chat** page. The **Chat** playground page consists of three main sections:
    - **Setup** - used to set the context for the model's responses.
    - **Chat session** - used to submit chat messages and view responses.
    - **Configuration** - used to configure settings for the model deployment.
2. In the **Configuration** section, ensure that your model deployment is selected.
3. In the **Setup** area, set the system message to `You are a programming assistant helping write code` and apply the changes.
4. In the **Chat session**, submit the following query:

    ```
    Write a function in python that takes a character and a string as input, and returns how many times the character appears in the string
    ```

    The model will likely respond with a function, with some explanation of what the function does and how to call it.

5. Next, send the prompt `Do the same thing, but this time write it in C#`.

   The model likely responded very similarly as the first time, but this time coding in C#. You can ask it again for a different language of your choice, or a function to complete a different task such as reversing the input string.

6. Next, let's explore using AI to understand code. Submit the following prompt as the user message.

```
What does the following function do?
---
def multiply(a, b):
    result = 0
    negative = False
    if a < 0 and b > 0:
        a = -a
        negative = True
    elif a > 0 and b < 0:
        b = -b
        negative = True
    elif a < 0 and b < 0:
        a = -a
        b = -b
    while b > 0:
        result += a
        b -= 1
    if negative:
        return -result
    else:
     return result
```

   The model should describe what the function does, which is to multiply two numbers together by using a loop.

27. Submit the prompt `Can you simplify the function?`.

    The model should write a simpler version of the function.

28. Submit the prompt: `Add some comments to the function.`

    The model adds comments to the code.

Prepare to develop an app in Visual Studio Code

Now let's explore how you could build a custom app that uses Azure OpenAI service to generate code. You'll develop your 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 azure-**openai** 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/parveenkrraina/azure-openai` 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, as well as a sample text file you'll use to test the summarization. Both apps feature the same functionality. First, you'll complete some key parts of the application to enable using your Azure OpenAI resource.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/04-code-generation** folder and expand the **CSharp** or **Python** folder depending on your language preference. Each folder contains the language-specific files for an app into which you're you're going to integrate Azure OpenAI functionality.
2. Right-click the **CSharp** or **Python** folder containing your code files and open an integrated terminal. Then install the Azure OpenAI SDK package by running the appropriate command for your language preference:

   C#:

   ```
   dotnet add package Azure.AI.OpenAI --version 1.0.0-beta.14
   ```

   Python:

   ```
   pip install openai==1.13.3
   ```

3. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the configuration file for your preferred language
   - **C#**: appsettings.json
   - **Python**: .env
4. Update the configuration values to include:
   - The **endpoint** and a **key** from the Azure OpenAI resource you created (available on the **Keys and Endpoint** page for your Azure OpenAI resource in the Azure portal)
   - The **deployment name** you specified for your model deployment (available in the **Deployments** page in Azure OpenAI Studio).
5. Save the configuration file.

Add code to use your Azure OpenAI service model

Now you're ready to use the Azure OpenAI SDK to consume your deployed model.

1. In the **Explorer** pane, in the **CSharp** or **Python** folder, open the code file for your preferred language. In the function that calls the Azure OpenAI model, under the comment ***Format and send the request to the model***, add the code to format and send the request to the model.

   C#: Program.cs

   ```
   csharp
   // Format and send the request to the model
   var chatCompletionsOptions = new ChatCompletionsOptions()
   {
       Messages =
       {
           new ChatRequestSystemMessage(systemPrompt),
           new ChatRequestUserMessage(userPrompt)
       },
       Temperature = 0.7f,
       MaxTokens = 1000,
       DeploymentName = oaiDeploymentName
   };

   // Get response from Azure OpenAI
   Response<ChatCompletions> response = await
   client.GetChatCompletionsAsync(chatCompletionsOptions);

   ChatCompletions completions = response.Value;
string completion = completions.Choices[0].Message.Content;
   ```

   Python: code-generation.py

   ```
   python
   # Format and send the request to the model
   messages =[
       {"role": "system", "content": system_message},
       {"role": "user", "content": user_message},
   ]

   # Call the Azure OpenAI model
   response = client.chat.completions.create(
       model=model,
       messages=messages,
       temperature=0.7,
       max_tokens=1000
)
   ```

2. Save the changes to the code file.

Run the application

Now that your app has been configured, run it to try generating code for each use case. The use case is numbered in the app, and can be run in any order.

Note: Some users may experience rate limiting if calling the model too frequently. If you hit an error about a token rate limit, wait for a minute then try again.

1. In the **Explorer** pane, expand the **Labfiles/04-code-generation/sample-code** folder and review the function and the *go-fish* app for your language. These files will be used for the tasks in the app.
2. In the interactive terminal pane, ensure the folder context is the folder for your preferred language. Then enter the following command to run the application.
    - **C#**: `dotnet run`
    - **Python**: `python code-generation.py`

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

3. Choose option **1** to add comments to your code and enter the following prompt. Note, the response might take a few seconds for each of these tasks.

    ```
    prompt
    Add comments to the following function. Return only the commented code.\n---\n
    ```

    The results will be put into **result/app.txt**. Open that file up, and compare it to the function file in **sample-code**.

4. Next, choose option **2** to write unit tests for that same function and enter the following prompt.

    ```
    prompt
    Write four unit tests for the following function.\n---\n
    ```

    The results will replace what was in **result/app.txt**, and details four unit tests for that function.

5. Next, choose option **3** to fix bugs in an app for playing Go Fish. Enter the following prompt.

    ```
    prompt
    Fix the code below for an app to play Go Fish with the user. Return only the corrected code.\n---\n
    ```

    The results will replace what was in **result/app.txt**, and should have very similar code with a few things corrected.

    - **C#**: Fixes are made on line 30 and 59
    - **Python**: Fixes are made on line 18 and 31

The app for Go Fish in **sample-code** can be run if you replace the lines that contain bugs with the response from Azure OpenAI. If you run it without the fixes, it will not work correctly.

Note: It's important to note that even though the code for this Go Fish app was corrected for some syntax, it's not a strictly accurate representation of the game. If you look closely, there are issues with not checking if the deck is empty when drawing cards, not removing pairs from the players hand when they get a pair, and a few other bugs that require understanding of card games to realize. This is a great example of how useful generative AI models can be to assist with code generation, but can't be trusted as correct and need to be verified by the developer.

If you would like to see the full response from Azure OpenAI, you can set the **printFullResponse** variable to `True`, and rerun the app.