Utilize prompt engineering in your app

When working with the Azure OpenAI Service, how developers shape their prompt greatly impacts how the generative AI model will respond. Azure OpenAI models are able to tailor and format content, if requested in a clear and concise way. In this exercise, you'll learn how different prompts for similar content help shape the AI model's response to better satisfy your requirements.

In scenario for this exercise, you will perform the role of a software developer working on a wildlife marketing campaign. You are exploring how to use generative AI to improve advertising emails and categorize articles that might apply to your team. The prompt engineering techniques used in the exercise can be applied similarly for a variety of 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.

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.

Explore prompt engineering techniques

Let's start by exploring some prompt engineering techniques in the Chat playground.

1. In **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, select the default system message template to set the context for the chat session. The default system message is *You are an AI assistant that helps people find information*.
4. In the **Chat session**, submit the following query:

```
What kind of article is this?
---
Severe drought likely in California

```

```
9. Millions of California residents are bracing for less water and dry
   lawns as drought threatens to leave a large swath of the region with a
   growing water shortage.
10.
11. In a remarkable indication of drought severity, officials in Southern
   California have declared a first-of-its-kind action limiting outdoor
   water use to one day a week for nearly 8 million residents.
12.
Much remains to be determined about how daily life will change as people
adjust to a drier normal. But officials are warning the situation is dire
and could lead to even more severe limits later in the year.
```

The response provides a description of the article. However, suppose you want a more specific format for article categorization.

13. In the **Setup** section change the system message to `You are a news aggregator that categorizes news articles.`
14. Under the new system message, in the **Examples** section, select the **Add** button. Then add the following example.

User:

```
What kind of article is this?
---
New York Baseballers Wins Big Against Chicago

New York Baseballers mounted a big 5-0 shutout against the Chicago
Cyclones last night, solidifying their win with a 3 run homerun late in
the bottom of the 7th inning.

Pitcher Mario Rogers threw 96 pitches with only two hits for New York,
marking his best performance this year.
```

```
The Chicago Cyclones' two hits came in the 2nd and the 5th innings but were
unable to get the runner home to score.
```

Assistant:

```
Sports
```

15. Add another example with the following text.

User:

```
Categorize this article:
---
Joyous moments at the Oscars

The Oscars this past week where quite something!

Though a certain scandal might have stolen the show, this year's Academy
Awards were full of moments that filled us with joy and even moved us to
tears.
These actors and actresses delivered some truly emotional performances,
along with some great laughs, to get us through the winter.
```

```
From Robin Kline's history-making win to a full performance by none other
than Casey Jensen herself, don't miss tomorrows rerun of all the
festivities.
```

Assistant:

```
Entertainment
```

16. Use the **Apply changes** button at the top of the **Setup** section to update the system message.
17. In the **Chat session** section, resubmit the following prompt:

```
18. What kind of article is this?
19. ---
20. Severe drought likely in California
21.
22. Millions of California residents are bracing for less water and dry
    lawns as drought threatens to leave a large swath of the region with a
    growing water shortage.
23.
24. In a remarkable indication of drought severity, officials in Southern
    California have declared a first-of-its-kind action limiting outdoor
    water use to one day a week for nearly 8 million residents.
25.
Much remains to be determined about how daily life will change as people
adjust to a drier normal. But officials are warning the situation is dire
and could lead to even more severe limits later in the year.
```

The combination of a more specific system message and some examples of expected queries and responses results in a consistent format for the results.

26. In the **Setup** section, change the system message back to the default template, which should be `You are an AI assistant that helps people find information.` with no examples. Then apply the changes.
27. In the **Chat session** section, submit the following prompt:

```
28. # 1. Create a list of animals
29. # 2. Create a list of whimsical names for those animals
# 3. Combine them randomly into a list of 25 animal and name pairs
```

The model will likely respond with an answer to satisfy the prompt, split into a numbered list. This is an appropriate response, but suppose what you actually wanted was for the model to write a Python program that performs the tasks you described?

30. Change the system message to `You are a coding assistant helping write python code.` and apply the changes.
31. Resubmit the following prompt to the model:

```
32. # 1. Create a list of animals
33. # 2. Create a list of whimsical names for those animals
# 3. Combine them randomly into a list of 25 animal and name pairs
```

The model should correctly respond with python code doing what the comments requested.

Prepare to develop an app in Visual Studio Code

Now let's explore the use of prompt engineering in an app that uses the Azure OpenAI service SDK. 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 **mslearn-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-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, and both apps feature the same functionality. First, you'll complete some key parts of the application to enable using your Azure OpenAI resource with asynchronous API calls.

1. In Visual Studio Code, in the **Explorer** pane, browse to the **Labfiles/03-prompt-engineering** 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 the Azure OpenAI service

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, and replace the comment ***Add Azure OpenAI package*** with code to add the Azure OpenAI SDK library:

   C#: Program.cs

   ```
   csharp
   // Add Azure OpenAI package
   using Azure.AI.OpenAI;
   ```

   Python: prompt-engineering.py

   ```
   python
   # Add Azure OpenAI package
   from openai import AsyncAzureOpenAI
   ```

2. In the code file, find the comment ***Configure the Azure OpenAI client***, and add code to configure the Azure OpenAI client:

   C#: Program.cs

   ```
   csharp
   // Configure the Azure OpenAI client
   OpenAIClient client = new OpenAIClient(new Uri(oaiEndpoint), new
   AzureKeyCredential(oaiKey));
   ```

   Python: prompt-engineering.py

   ```
   python
   # Configure the Azure OpenAI client
   client = AsyncAzureOpenAI(
       azure_endpoint = azure_oai_endpoint,
       api_key=azure_oai_key,
       api_version="2024-02-15-preview"
       )
   ```

3. 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(systemMessage),
        new ChatRequestUserMessage(userMessage)
    },
    Temperature = 0.7f,
    MaxTokens = 800,
    DeploymentName = oaiDeploymentName
};

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

Python: prompt-engineering.py

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

print("\nSending request to Azure OpenAI model...\n")

# Call the Azure OpenAI model
response = await client.chat.completions.create(
    model=model,
    messages=messages,
    temperature=0.7,
    max_tokens=800
)
```

4. Save the changes to the code file.

Run your application

Now that your app has been configured, run it to send your request to your model and observe the response. You'll notice the only difference between the different options is the content of the prompt, all other parameters (such as token count and temperature) remain the same for each request.

1. In the folder of your preferred language, open `system.txt` in Visual Studio Code. For each of the interations, you'll enter the **System message** in this file and save it. Each iteration will pause first for you to change the system message.
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 prompt-engineering.py`

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

3. For the first iteration, enter the following prompts:

   System message

```
    prompt
You are an AI assistant
```

   User message:

```
    prompt
Write an intro for a new wildlife Rescue
```

4. Observe the output. The AI model will likely produce a good generic introduction to a wildlife rescue.
5. Next, enter the following prompts which specify a format for the response:

   System message

```
    prompt
You are an AI assistant helping to write emails
```

   User message:

```
    prompt
    Write a promotional email for a new wildlife rescue, including the
    following:
    - Rescue name is Contoso
    - It specializes in elephants
- Call for donations to be given at our website
```

6. Observe the output. This time, you'll likely see the format of an email with the specific animals included, as well as the call for donations.
7. Next, enter the following prompts that additionally specify the content:

   System message

```
    prompt
You are an AI assistant helping to write emails
```

   User message:

```
    prompt
    Write a promotional email for a new wildlife rescue, including the
    following:
    - Rescue name is Contoso
    - It specializes in elephants, as well as zebras and giraffes
    - Call for donations to be given at our website
\n Include a list of the current animals we have at our rescue after the
signature, in the form of a table. These animals include elephants, zebras,
gorillas, lizards, and jackrabbits.
```

8. Observe the output, and see how the email has changed based on your clear instructions.
9. Next, enter the following prompts where we add details about tone to the system message:

   System message

   ```
   prompt
   You are an AI assistant that helps write promotional emails to generate interest in a new business. Your tone is light, chit-chat oriented and you always include at least two jokes.
   ```

   User message:

   ```
   prompt
   Write a promotional email for a new wildlife rescue, including the following:
   - Rescue name is Contoso
   - It specializes in elephants, as well as zebras and giraffes
   - Call for donations to be given at our website
   \n Include a list of the current animals we have at our rescue after the signature, in the form of a table. These animals include elephants, zebras, gorillas, lizards, and jackrabbits.
   ```

10. Observe the output. This time you'll likely see the email in a similar format, but with a much more informal tone. You'll likely even see jokes included!
11. For the final iteration, we're deviating from email generation and exploring *grounding context*. Here you provide a simple system message, and change the app to provide the grounding context as the beginning of the user prompt. The app will then append the user input, and extract information from the grounding context to answer our user prompt.
12. Open the file `grounding.txt` and briefly read the grounding context you'll be inserting.
13. In your app immediately after the comment ***Format and send the request to the model*** and before any existing code, add the following code snippet to read text in from `grounding.txt` to augment the user prompt with the grounding context.

    C#: Program.cs

    ```
    csharp
    // Format and send the request to the model
    Console.WriteLine("\nAdding grounding context from grounding.txt");
    string groundingText = System.IO.File.ReadAllText("grounding.txt");
    userMessage = groundingText + userMessage;
    ```

    Python: prompt-engineering.py

    ```
    python
    # Format and send the request to the model
    print("\nAdding grounding context from grounding.txt")
    grounding_text = open(file="grounding.txt", encoding="utf8").read().strip()
    user_message = grounding_text + user_message
    ```

14. Save the file and rerun your app.

15. Enter the following prompts (with the **system message** still being entered and saved in `system.txt`).

    System message

    ```prompt
    You're an AI assistant who helps people find information. You'll provide answers from the text provided in the prompt, and respond concisely.
    ```

    User message:

    ```prompt
    What animal is the favorite of children at Contoso?
    ```

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

Clean up

When you're done with your Azure OpenAI resource, remember to delete the deployment or the entire resource in the **Azure portal** at `https://portal.azure.com`.