Azure OpenAI

Table of content

1. Get started with Azure OpenAI Service:
2. Build natural language solutions with Azure OpenAI Service: 22
3. Apply prompt engineering with Azure OpenAI Service: 38
4. Generate code with Azure OpenAI Service: 63
5. Generate images with Azure OpenAI Service: 89
6. Use your own data with Azure OpenAI Service: 98
7. Responsible Generative AI: 108

1. Get started with Azure OpenAI Service

Introduction

- Azure OpenAI Service brings these generative AI models to the Azure platform, enabling you to develop powerful AI solutions that benefit from the security, scalability, and integration of other services provided by the Azure cloud platform.

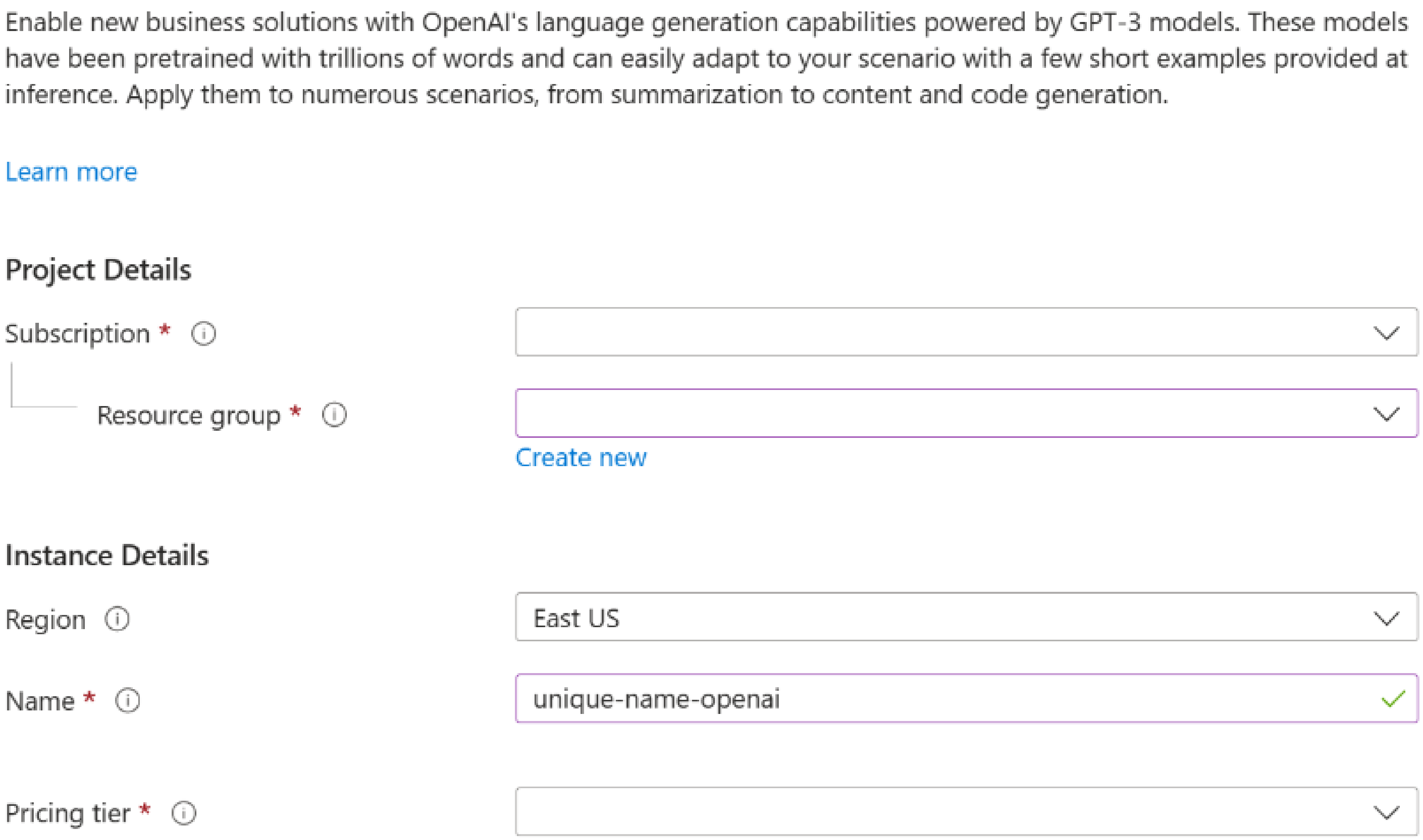
- These models are available for building applications through a REST API, various SDKs, and a Studio interface.

Access Azure OpenAI Service

- The first step in building a generative AI solution with Azure OpenAI is to provision an Azure OpenAI resource in your Azure subscription.

- Azure OpenAI Service is currently in limited access. Users need to apply for service access at https://aka.ms/oai/access.

Create Azure OpenAI



Enable new business solutions with OpenAI's language generation capabilities powered by GPT-3 models. These models have been pretrained with trillions of words and can easily adapt to your scenario with a few short examples provided at inference. Apply them to numerous scenarios, from summarization to content and code generation.

Learn more

Project Details

Subscription *

Resource group *

Create new

Instance Details

Region: East US

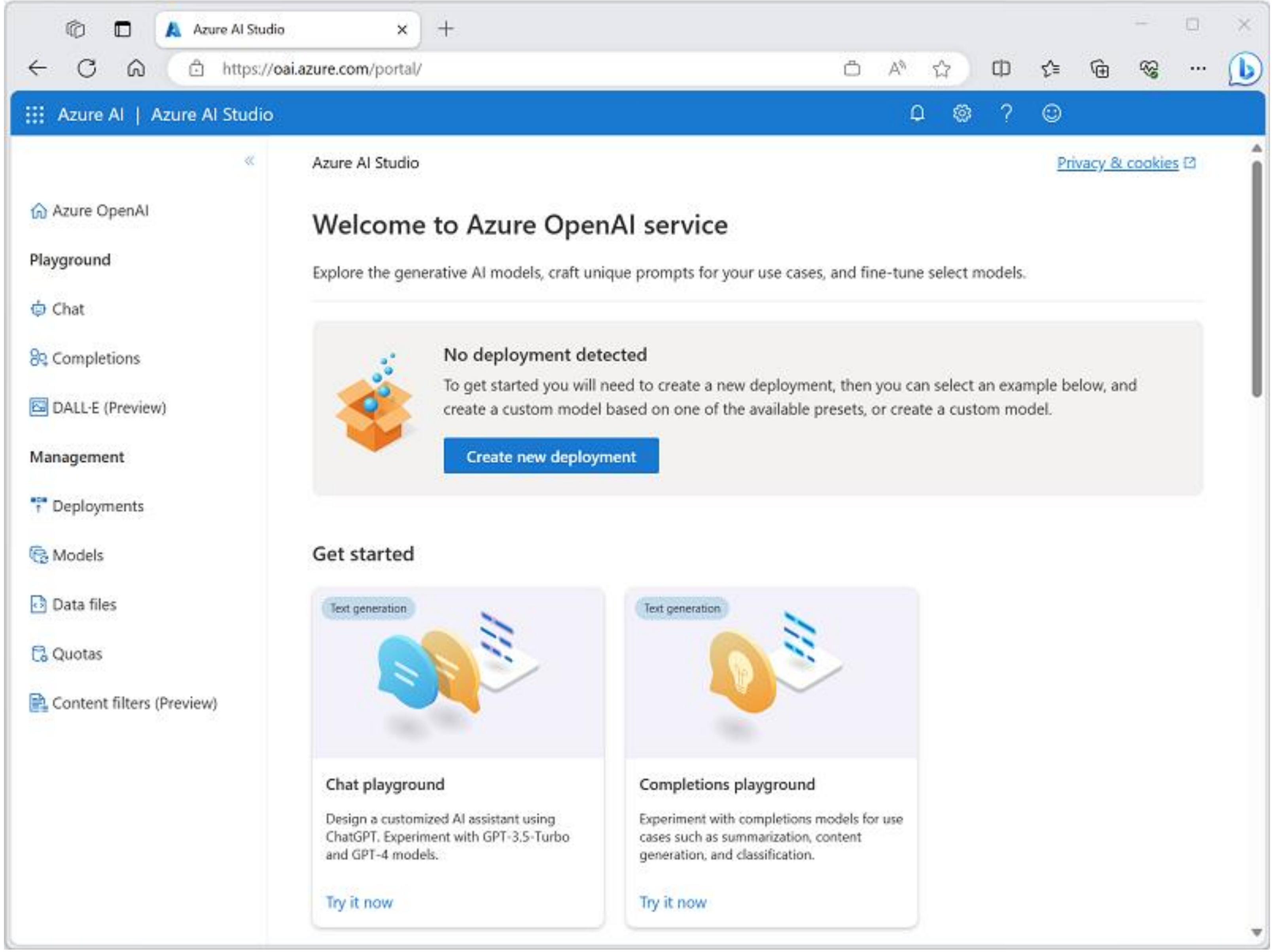
Name *: unique-name-openai ✓

Pricing tier *

Create an Azure OpenAI Service resource in Azure CLI

```
az cognitiveservices account create \
-n MyOpenAIResource \
-g OAIResourceGroup \
-l eastus \
--kind OpenAI \
--sku s0 \
--subscription subscriptionID
```

Use Azure OpenAI Studio



Welcome to Azure OpenAI service

Explore the generative AI models, craft unique prompts for your use cases, and fine-tune select models.

No deployment detected

To get started you will need to create a new deployment, then you can select an example below, and create a custom model based on one of the available presets, or create a custom model.

Get started

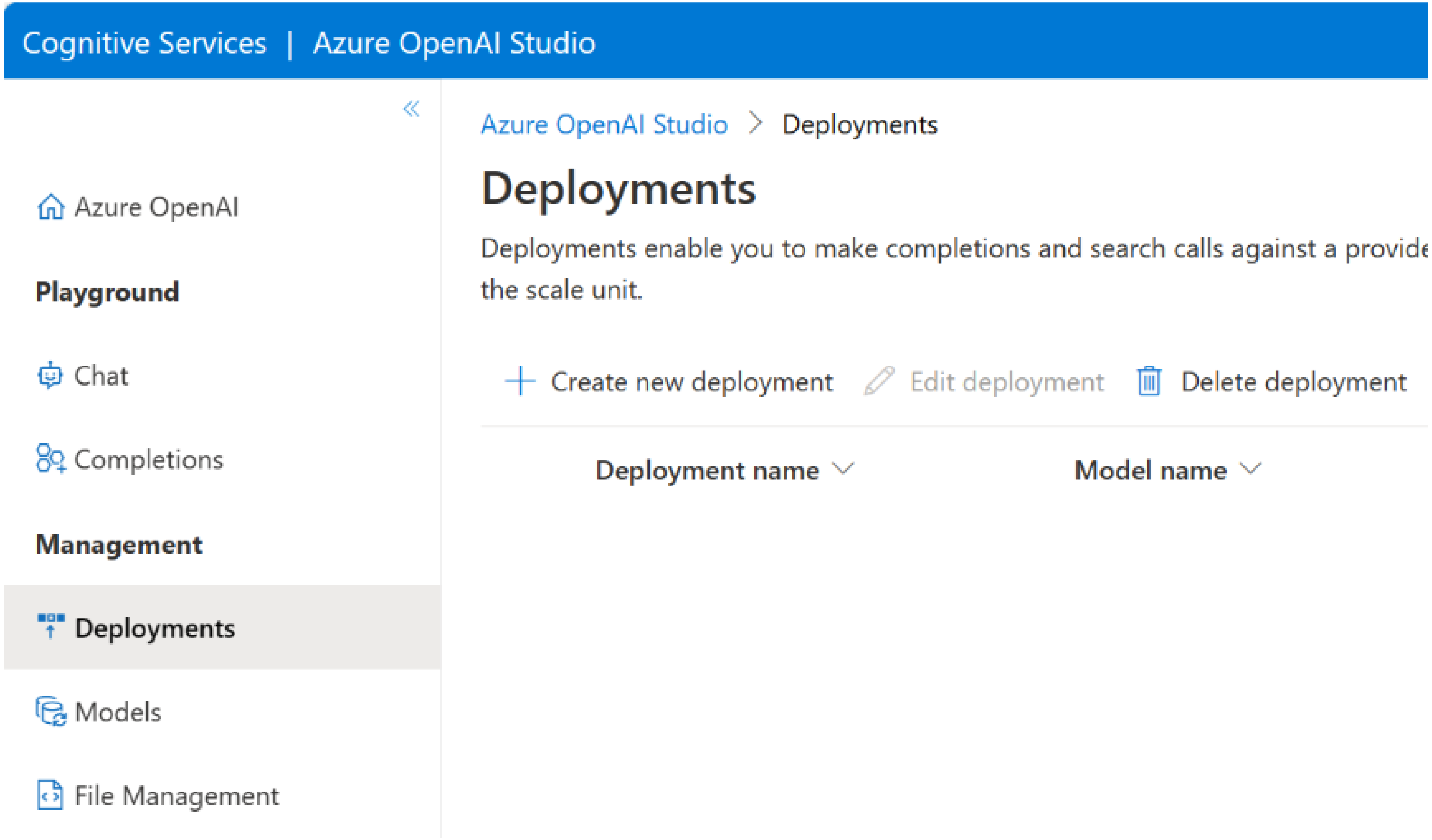
Chat playground

Design a customized AI assistant using ChatGPT. Experiment with GPT-3.5-Turbo and GPT-4 models.

Completions playground

Experiment with completions models for use cases such as summarization, content generation, and classification.

Use Azure OpenAI Studio



Deployments

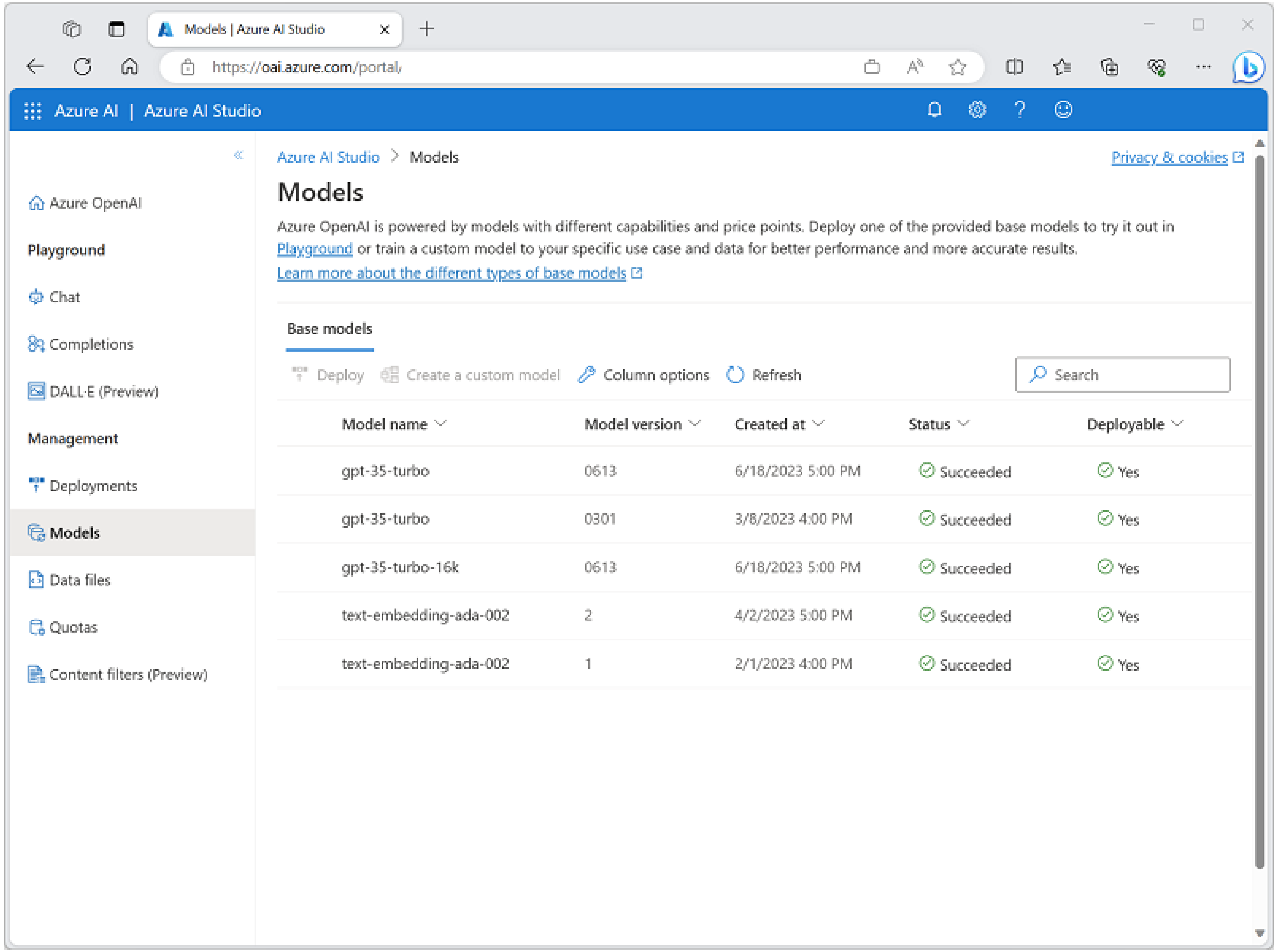
Deployments enable you to make completions and search calls against a provide
the scale unit.

Explore types of generative AI models

Azure OpenAI includes several types of model:

- GPT-4 models
- GPT 3.5 models
- Embeddings models
- DALL-E models

Explore types of generative AI models



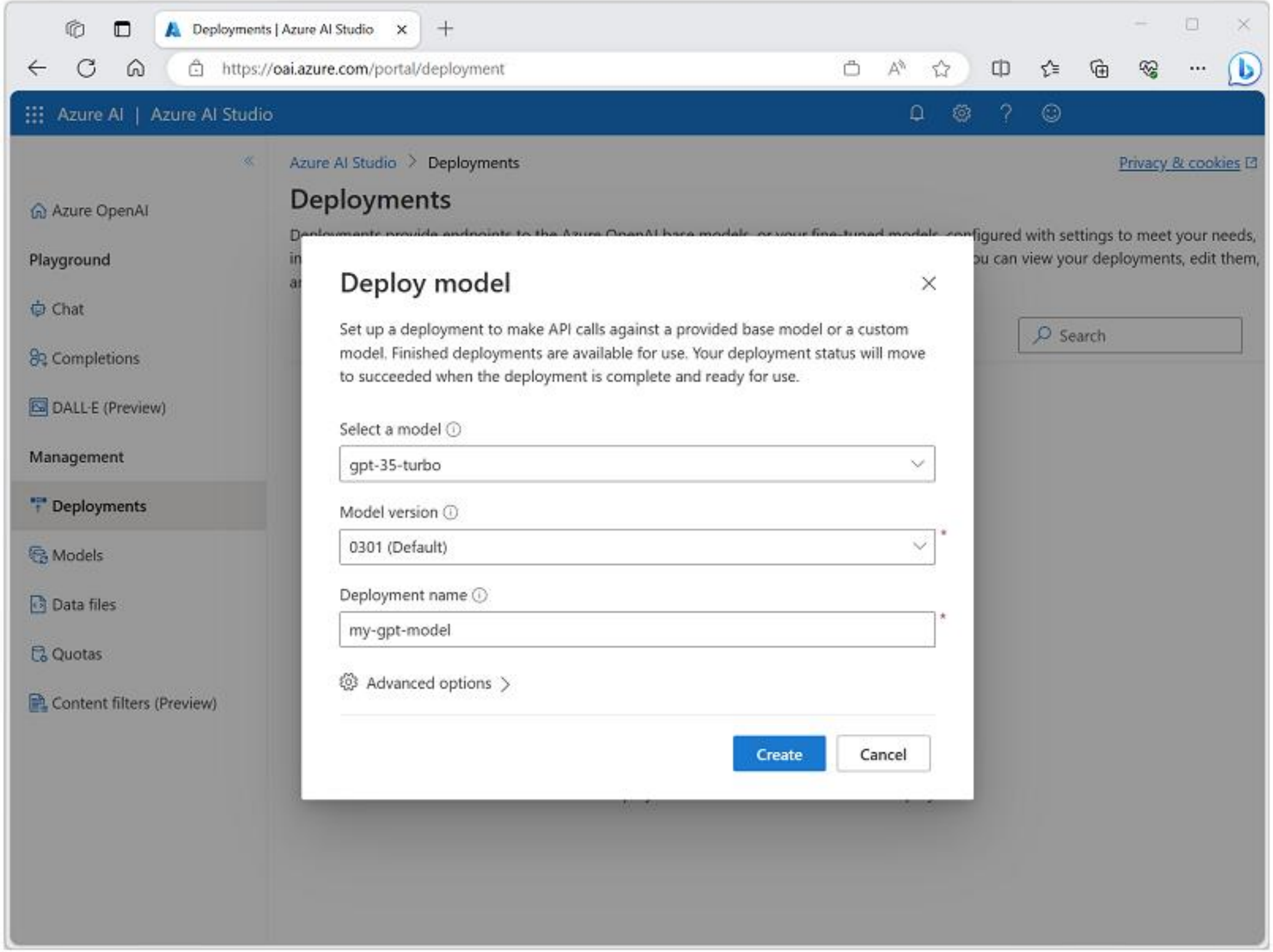
Models

Azure OpenAI is powered by models with different capabilities and price points. Deploy one of the provided base models to try it out in Playground or train a custom model to your specific use case and data for better performance and more accurate results.

Learn more about the different types of base models

Model name	Model version	Created at	Status	Deployable
gpt-35-turbo	0613	6/18/2023 5:00 PM	✓ Succeeded	✓ Yes
gpt-35-turbo	0301	3/8/2023 4:00 PM	✓ Succeeded	✓ Yes
gpt-35-turbo-16k	0613	6/18/2023 5:00 PM	✓ Succeeded	✓ Yes
text-embedding-ada-002	2	4/2/2023 5:00 PM	✓ Succeeded	✓ Yes
text-embedding-ada-002	1	2/1/2023 4:00 PM	✓ Succeeded	✓ Yes

Deploy using Azure OpenAI Studio



Deploy model

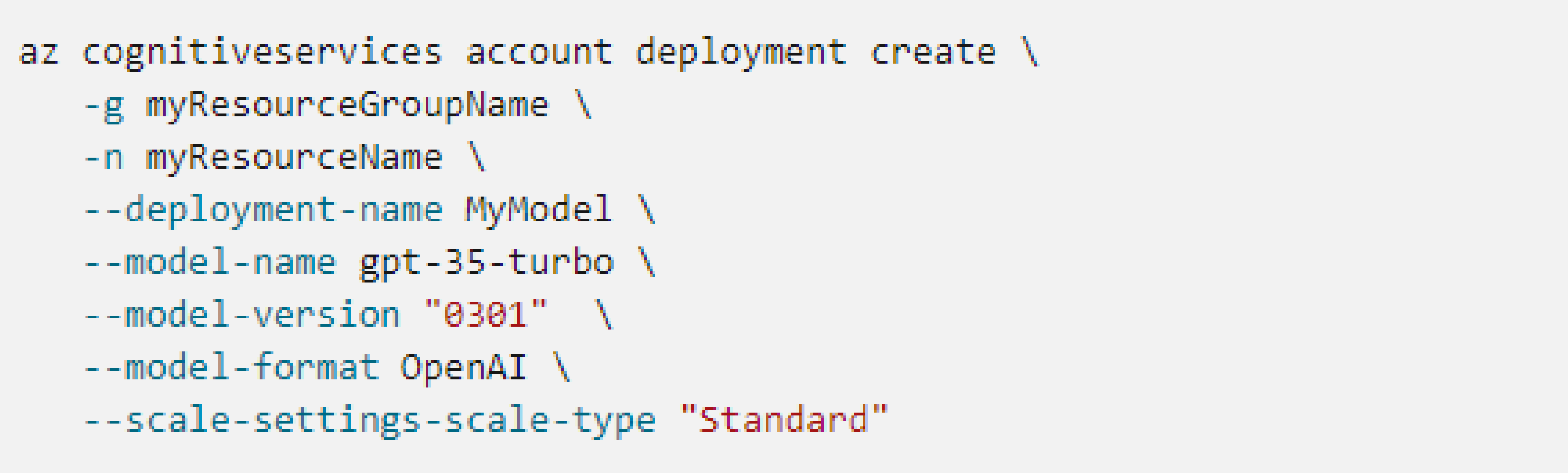
Set up a deployment to make API calls against a provided base model or a custom model. Finished deployments are available for use. Your deployment status will move to succeeded when the deployment is complete and ready for use.

Select a model: gpt-35-turbo

Model version: 0301 (Default)

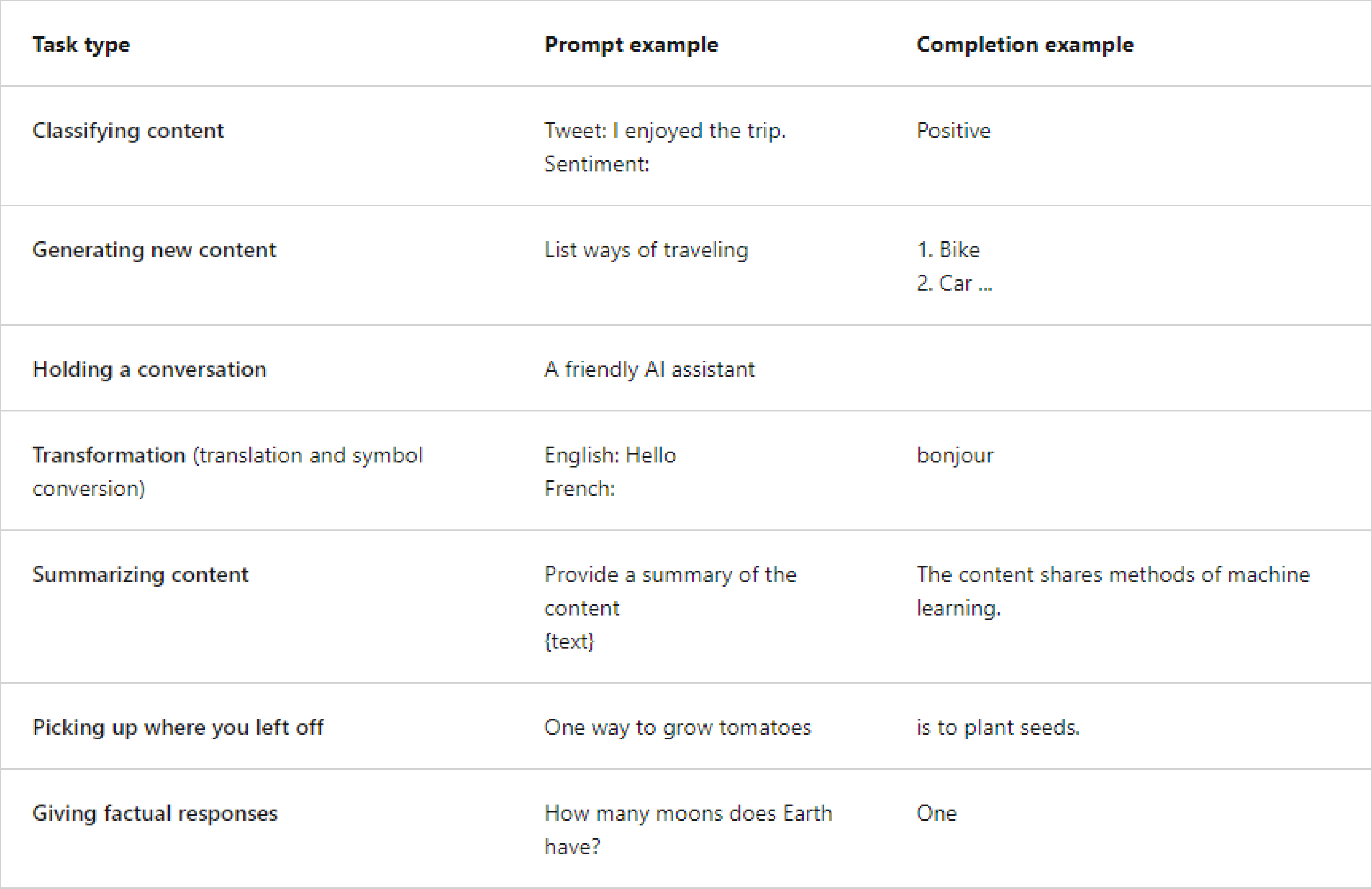
Deployment name: my-gpt-model

Deploy using Azure CLI



```
az cognitiveservices account deployment create \
   -g myResourceGroupName \
   -n myResourceName \
   --deployment-name MyModel \
   --model-name gpt-35-turbo \
   --model-version "0301"  \
   --model-format OpenAI \
   --scale-settings-scale-type "Standard"
```

Use prompts to get completions from models



Task type	**Prompt example**	**Completion example**
Classifying content	Tweet: I enjoyed the trip. Sentiment:	Positive
Generating new content	List ways of traveling	1. Bike 2. Car ...
Holding a conversation	A friendly AI assistant	
Transformation (translation and symbol conversion)	English: Hello French:	bonjour
Summarizing content	Provide a summary of the content {text}	The content shares methods of machine learning.
Picking up where you left off	One way to grow tomatoes	is to plant seeds.
Giving factual responses	How many moons does Earth have?	One

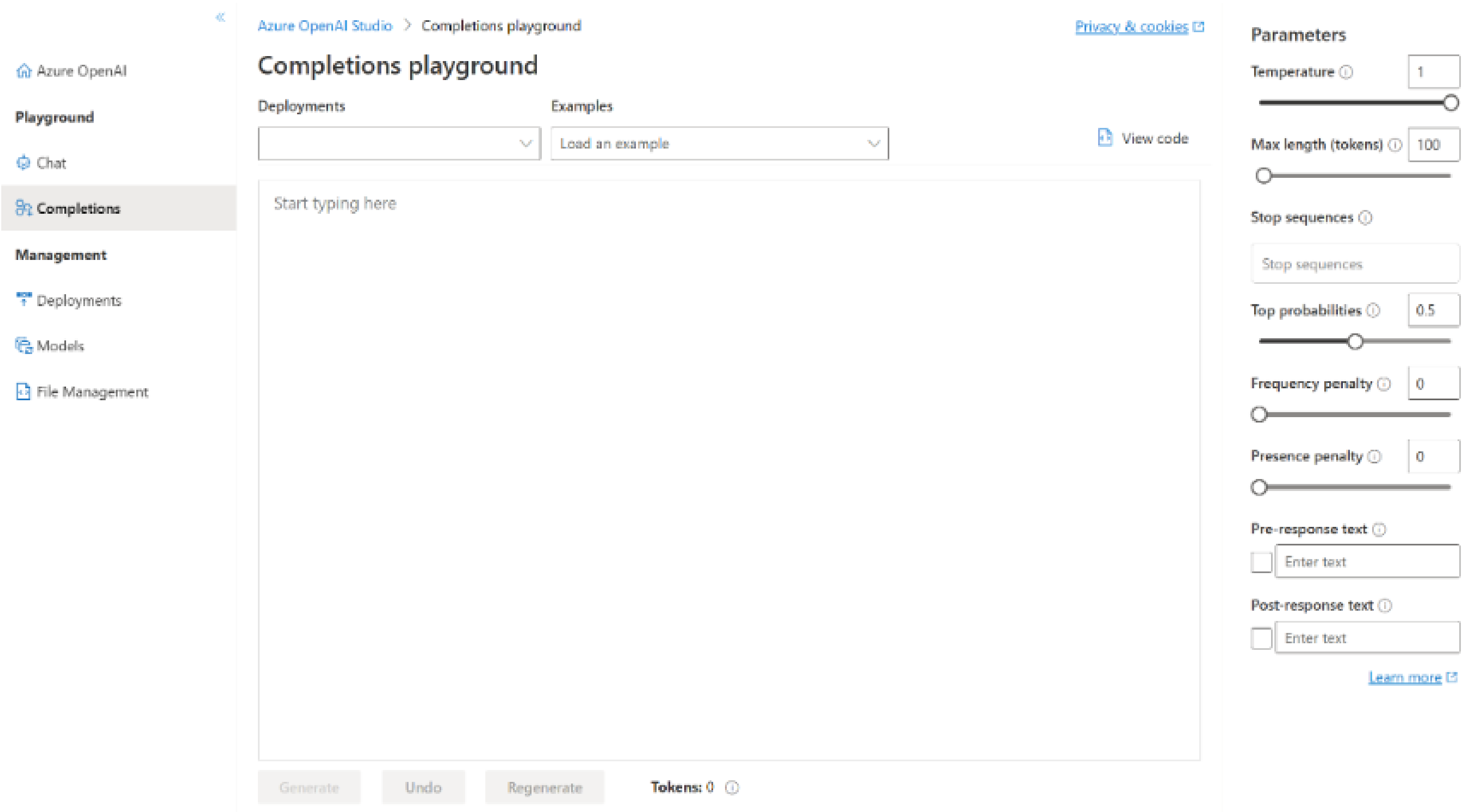
Completion quality

Several factors affect the quality of completions you'll get from a generative AI solution.

- The way a prompt is engineered.
- The model parameters (covered next)
- The data the model is trained on, which can be adapted through model fine-tuning with customization.

Test models in Azure OpenAI Studio's playgrounds

Completions playground



Temperature: 1

Max length (tokens): 100

Top probabilities: 0.5

Frequency penalty: 0

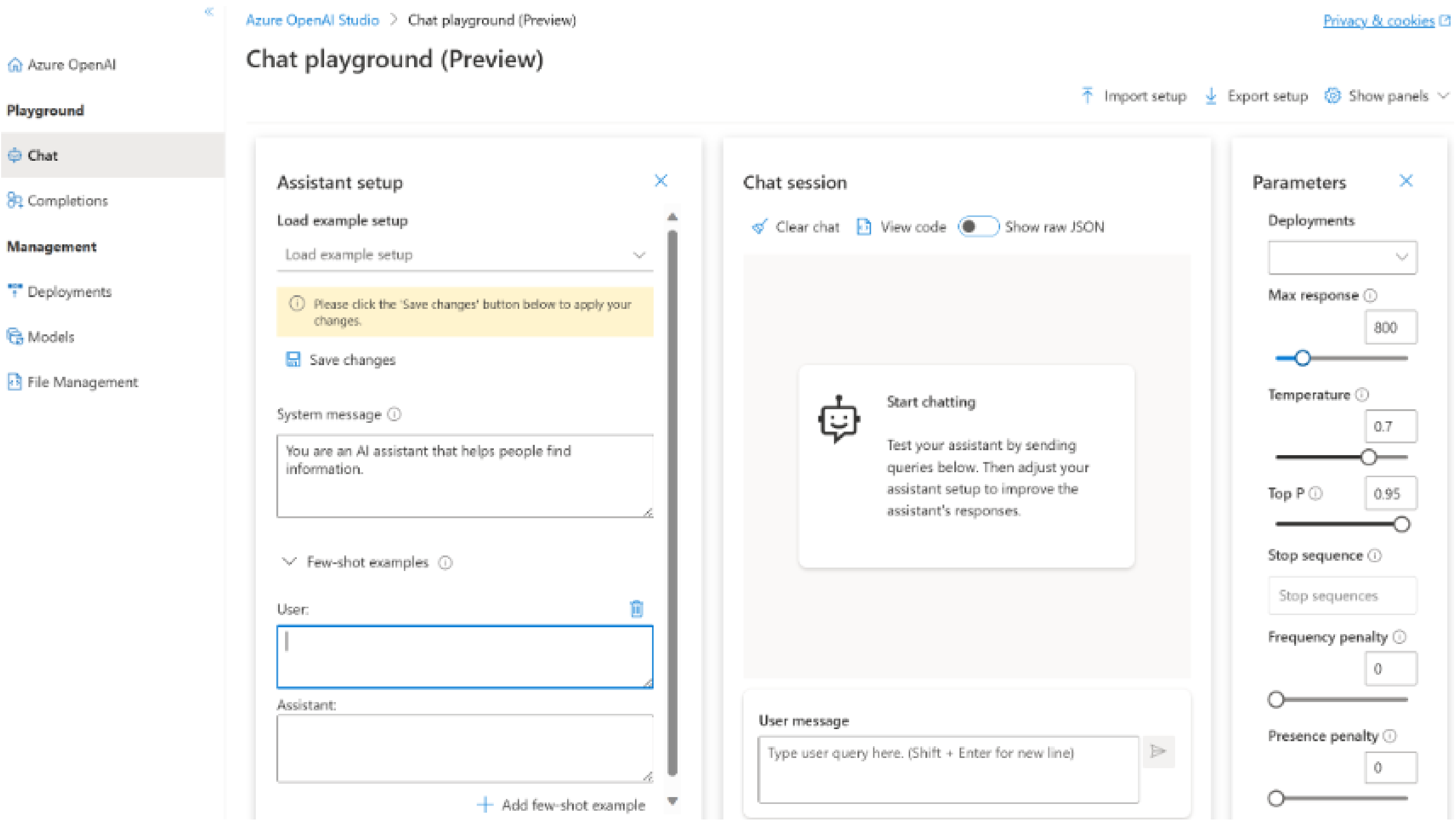
Presence penalty: 0

Completions Playground parameters

There are many parameters that you can adjust to change the performance of your model:

- Temperature
- Max length (tokens)
- Stop sequences
- Top probabilities (Top P)
- Frequency penalty
- Presence penalty
- Pre-response text
- Post-response text

Chat playground



Please click the 'Save changes' button below to apply your changes.

System message

You are an AI assistant that helps people find information.

Start chatting

Test your assistant by sending queries below. Then adjust your assistant setup to improve the assistant's responses.

Max response: 800

Temperature: 0.7

Top P: 0.95

Stop sequence

Frequency penalty: 0

Presence penalty: 0

Chat playground parameters

The Chat playground, like the Completions playground, also includes the Temperature parameter. The Chat playground also supports other parameters *not* available in the Completions playground. These include:

- Max response
- Top P
- Past messages included

Summary

You learned how to:

- Create an Azure OpenAI Service resource and understand types of Azure OpenAI base models.
- Use the Azure OpenAI Studio, Azure CLI, and REST API to deploy a base model.
- Generate completions to prompts.
- Test models in the Studio's playgrounds and begin to manage model parameters.

2. Build natural language solutions with Azure OpenAI Service

Introduction

- Azure OpenAI provides a platform for developers to add artificial intelligence functionality to their applications with the help of both Python and C# SDKs and REST APIs.

- The platform has various AI models available, each specializing in different tasks, which can be deployed through the Azure OpenAI Service.

Create an Azure OpenAI resource

1. Navigate to the Azure portal
2. Search for **Azure OpenAI**, select it, and click **Create**
3. Enter the appropriate values for the empty fields, and create the resource.

Choose and deploy a model

Each model family excels at different tasks, and there are different capabilities of the models within each family. Model families break down into three main families:

- Text or Generative Pre-trained Transformer (GPT)
- Code
- Embeddings

Prompt engineering

Classify the following news headline into 1 of the following categories: Business, Tech, Politics, Sport, Entertainment

Headline 1: Donna Steffensen Is Cooking Up a New Kind of Perfection. The Internet's most beloved cooking guru has a buzzy new book and a fresh new perspective
Category: Entertainment

Headline 2: Major Retailer Announces Plans to Close Over 100 Stores
Category:

Available endpoints

AOAI can be accessed via a REST API or an SDK currently available for Python and C#. The endpoints available for interacting with a deployed model are used differently, and certain endpoints can only use certain models. The available endpoints are:

- Completion
- ChatCompletion
- Embeddings

Available endpoints

- For example, the input for Completion might be a prompt like "What is Azure OpenAI", or it might include some role tags or other prompt engineering elements.

```
<|im_start|>user
What is Azure OpenAI?
<|im_end|>
```

Available endpoints

- In contrast, the input for ChatCompletion is a conversation with clearly defined roles for each message:

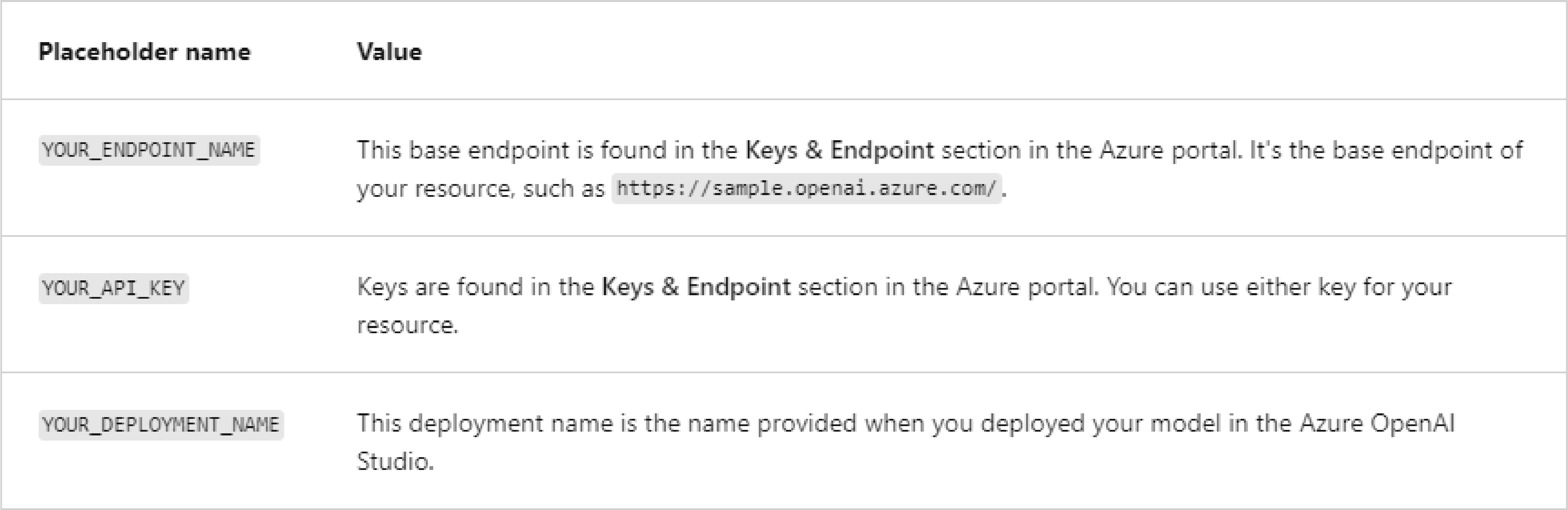
```
{"role": "system", "content": "You are a helpful assistant, teaching people about AI."},
{"role": "user", "content": "Does Azure OpenAI support multiple languages?"},
{"role": "assistant", "content": "Yes, Azure OpenAI supports several languages, and can translate between them."},
{"role": "user", "content": "Do other Azure AI Services support translation too?"}
```

Available endpoints

- For example, if you want to generate a job description, provide ChatCompletion with something like the following conversation input.

```
{"role": "system", "content": "You are an assistant designed to write intriguing job descriptions. "}, {"role":
"user", "content": "Write a job description for the following job title: 'Business Intelligence Analyst'. It should
include responsibilities, required qualifications, and highlight benefits like time off and flexible hours."}
```

Use Azure OpenAI SDK



Placeholder name	**Value**
`YOUR_ENDPOINT_NAME`	This base endpoint is found in the **Keys & Endpoint** section in the Azure portal. It's the base endpoint of your resource, such as `https://sample.openai.azure.com/`.
`YOUR_API_KEY`	Keys are found in the **Keys & Endpoint** section in the Azure portal. You can use either key for your resource.
`YOUR_DEPLOYMENT_NAME`	This deployment name is the name provided when you deployed your model in the Azure OpenAI Studio.

Install libraries

- First, install the client library for your preferred language.
- The C# SDK is a .NET adaptation of the REST APIs and built specifically for Azure OpenAI, however it can be used to connect to Azure OpenAI resources or non-Azure OpenAI endpoints.
- The Python SDK is built and maintained by OpenAI.

```
dotnet add package Azure.AI.OpenAI --prerelease
```

Configure app to access Azure OpenAI resource

```
// Add OpenAI library
using Azure.AI.OpenAI;
// Define parameters and initialize the client
string endpoint = "<YOUR_ENDPOINT_NAME>";
string key = "<YOUR_API_KEY>";
string deploymentName = "<YOUR_DEPLOYMENT_NAME>"; //SDK calls this "engine", but naming
                                                // it "deploymentName" for clarity

OpenAIClient client = new OpenAIClient(new Uri(endpoint), new AzureKeyCredential(key));
```

Call Azure OpenAI resource

```
string prompt = "What is Azure OpenAI?";

Response<Completions> completionsResponse = client.GetCompletions(deploymentName, prompt); string
completion = completionsResponse.Value.Choices[0].Text;
Console.WriteLine($"Chatbot: {completion}");
```

Call Azure OpenAI resource

```
// Build completion options object
ChatCompletionsOptions chatCompletionsOptions = new ChatCompletionsOptions()
 {
        Messages =
        {
                 new ChatMessage(ChatRole.System, "You are a helpful AI bot."),
                 new ChatMessage(ChatRole.User, "What is Azure OpenAI?")
         }
};

 // Send request to Azure OpenAI model
ChatCompletions chatCompletionsResponse = client.GetChatCompletions(
        deploymentName,
        chatCompletionsOptions);
 ChatMessage completion = chatCompletionsResponse.Choices[0].Message; Console.WriteLine($"Chatbot:
{completion.Content}");
```

Call Azure OpenAI resource

"Azure OpenAI is a cloud-based artificial intelligence (AI) service that offers a range of tools and services for developing and deploying AI applications. Azure OpenAI provides a variety of services for training and deploying machine learning models, including a managed service for training and deploying deep learning models, a managed service for deploying machine learning models, and a managed service for managing and deploying machine learning models."

Summary

In this module, you learned how to:

- Integrate Azure OpenAI into your application
- Differentiate between different endpoints available to your application
- Generate completions to prompts using the REST API and language specific SDKs

3. Apply prompt engineering with Azure OpenAI Service

Introduction

- In this module, you'll learn about prompt engineering and how it can be used to optimize the performance of Azure OpenAI models.

- Prompt engineering involves designing and optimizing prompts to better utilize Azure OpenAI models.

What is prompt engineering

- Prompt engineering is the process of designing and optimizing prompts to better utilize LLMs.
- Designing effective prompts is critical to the success of prompt engineering, and it can significantly improve the AI model's performance on specific tasks.
- Providing relevant, specific, unambiguous, and well structured prompts can help the model better understand the context and generate more accurate responses.

Considerations for API endpoints

- Functionally, ChatCompletion has the option of defining a system message for the AI model, in addition to built-in structure to provide previous messages in the prompt.
- If using Completion, this functionality can be achieved with what's called a meta prompt.
- In terms of model availability, both endpoints can utilize similar models, including gpt-35-turbo, but only ChatCompletion can be used with gpt-4 generation models.

Adjusting model parameters

- Higher values produce more creative and random responses, but will likely be less consistent or focused.

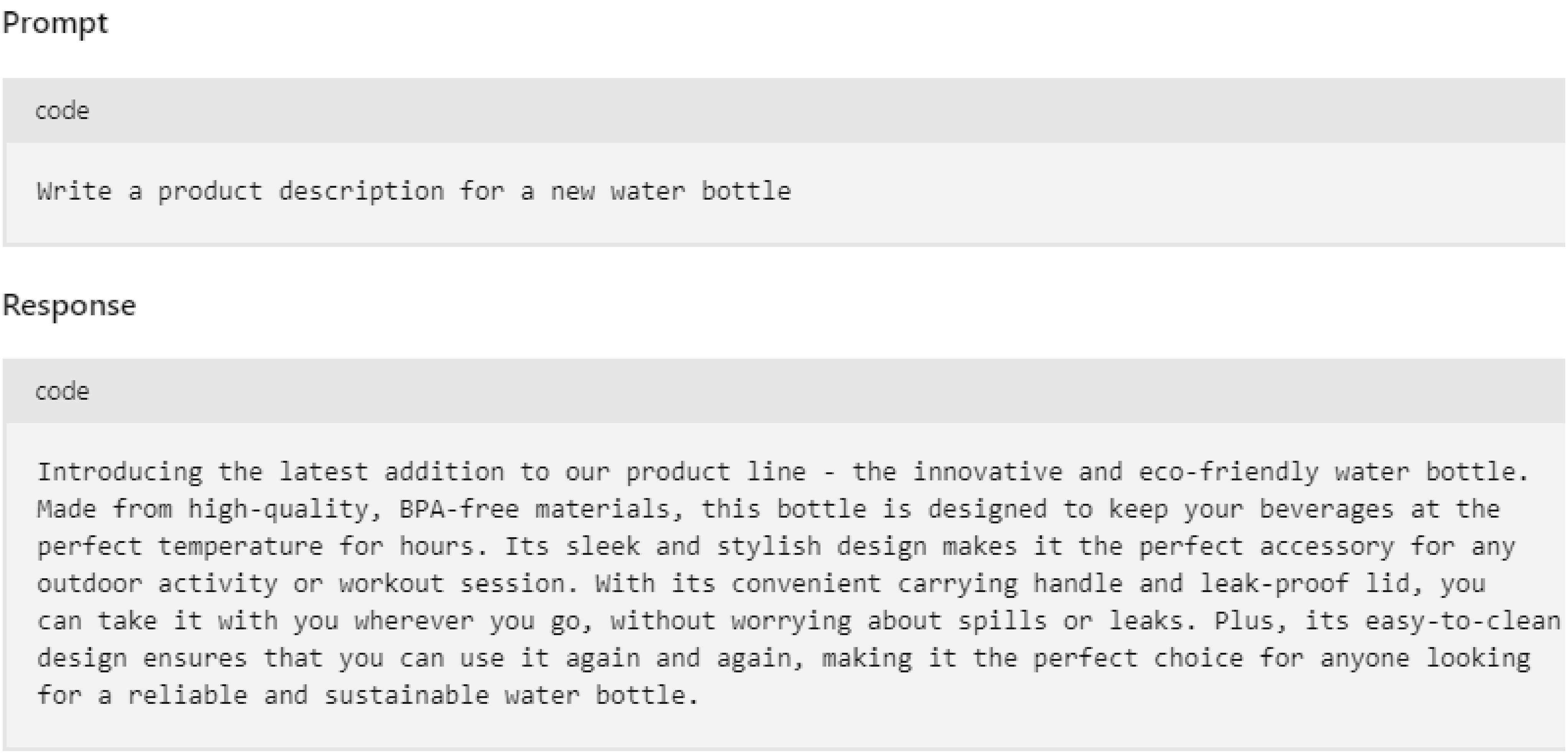
- Responses expected to be fictional or unique benefit from higher values for these parameters, whereas content desired to be more consistent and concrete should use lower values.

- Try adjusting these parameters with the same prompt to see how they impact the response.

- It's recommended to change either temperature or top_p at a time, but not both.

Write more effective prompts

Provide clear instructions



Prompt

code

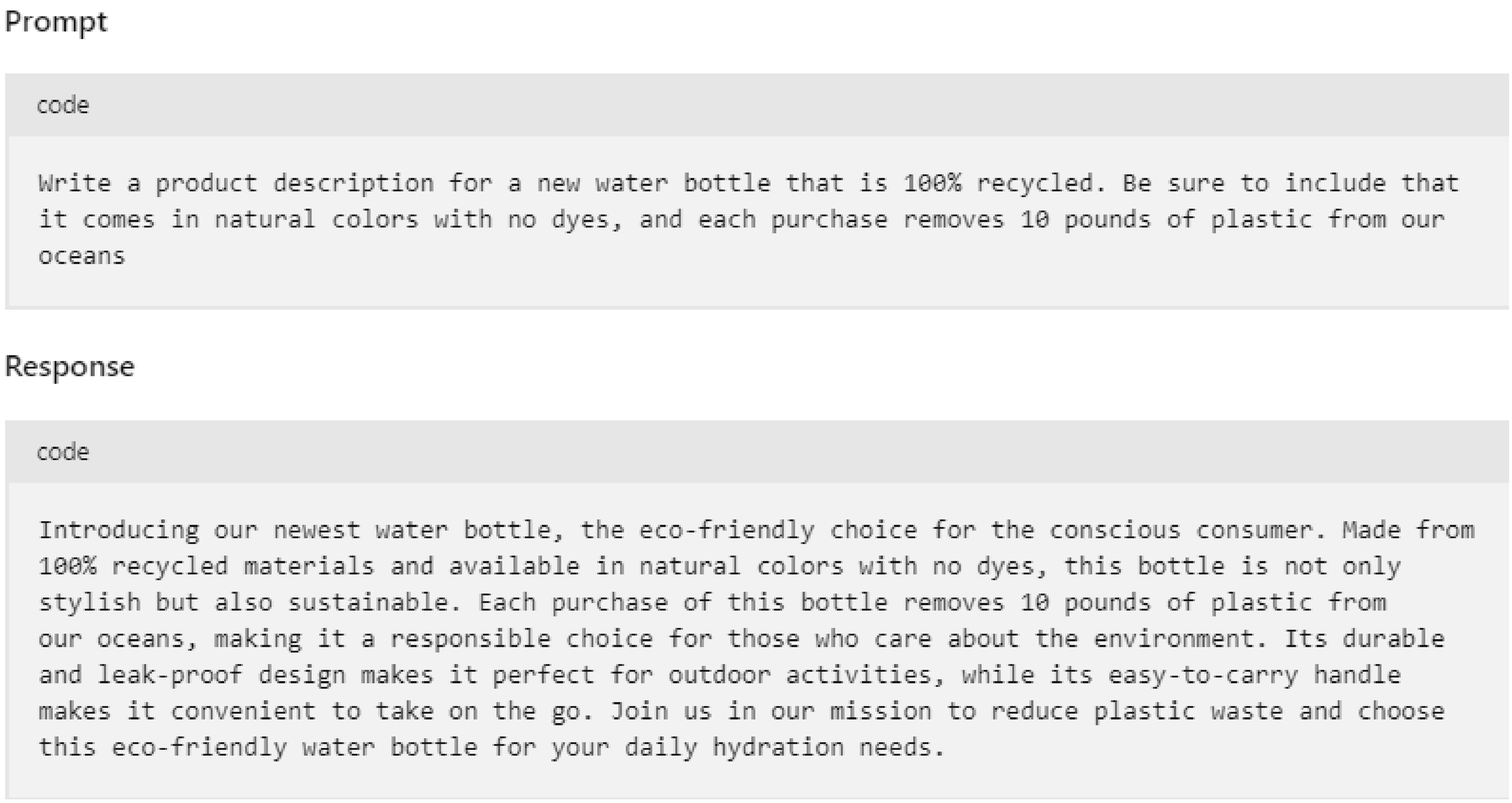
```
Write a product description for a new water bottle
```

Response

code

```
Introducing the latest addition to our product line - the innovative and eco-friendly water bottle.
Made from high-quality, BPA-free materials, this bottle is designed to keep your beverages at the
perfect temperature for hours. Its sleek and stylish design makes it the perfect accessory for any
outdoor activity or workout session. With its convenient carrying handle and leak-proof lid, you
can take it with you wherever you go, without worrying about spills or leaks. Plus, its easy-to-clean
design ensures that you can use it again and again, making it the perfect choice for anyone looking
for a reliable and sustainable water bottle.
```

Write more effective prompts



Prompt

code

```
Write a product description for a new water bottle that is 100% recycled. Be sure to include that
it comes in natural colors with no dyes, and each purchase removes 10 pounds of plastic from our
oceans
```

Response

code

```
Introducing our newest water bottle, the eco-friendly choice for the conscious consumer. Made from
100% recycled materials and available in natural colors with no dyes, this bottle is not only
stylish but also sustainable. Each purchase of this bottle removes 10 pounds of plastic from
our oceans, making it a responsible choice for those who care about the environment. Its durable
and leak-proof design makes it perfect for outdoor activities, while its easy-to-carry handle
makes it convenient to take on the go. Join us in our mission to reduce plastic waste and choose
this eco-friendly water bottle for your daily hydration needs.
```

Format of instructions

- How instructions are formatted can impact how the model interprets the prompt.
- Recency bias can affect models, where information located towards the end of the prompt can have more influence on the output than information at the beginning.
- You may get better responses by repeating the instructions at the end of the prompt and assessing how that affects the generated response.

Use section markers

- A specific technique for formatting instructions is to split the instructions at the beginning or end of the prompt, and have the user content contained within --- or ### blocks.
- These tags allow the model to more clearly differentiate between instructions and content.

For example:

```
Translate the text into French

---
What's the weather going to be like today?
---
```

Primary, supporting, and grounding content

Including content for the model to use to respond with allows it to answer with greater accuracy. This content can be thought of in following ways:

- Primary content
- Supporting content
- Grounding content

Primary content

For example, say we have a long article that we want to summarize. We could put it in a --- block in the prompt, then end with the instruction.

```
---
<insert full article here, as primary content>
---

Summarize this article and identify three takeaways in a bulleted list
```

Supporting content

```
---
<insert full email here, as primary content>
 ---
<the next line is the supporting content>
Topics I'm very interested in: AI, webinar dates, submission deadlines

Extract the key points from the above email, and put them in a bulleted list:
```

Grounding content

- For example, when provided an unpublished research paper on the history of AI, it can then answer questions using that grounding content.

<insert unpublished paper on the history of AI here, as grounding content>

Where and when did the field of AI start?

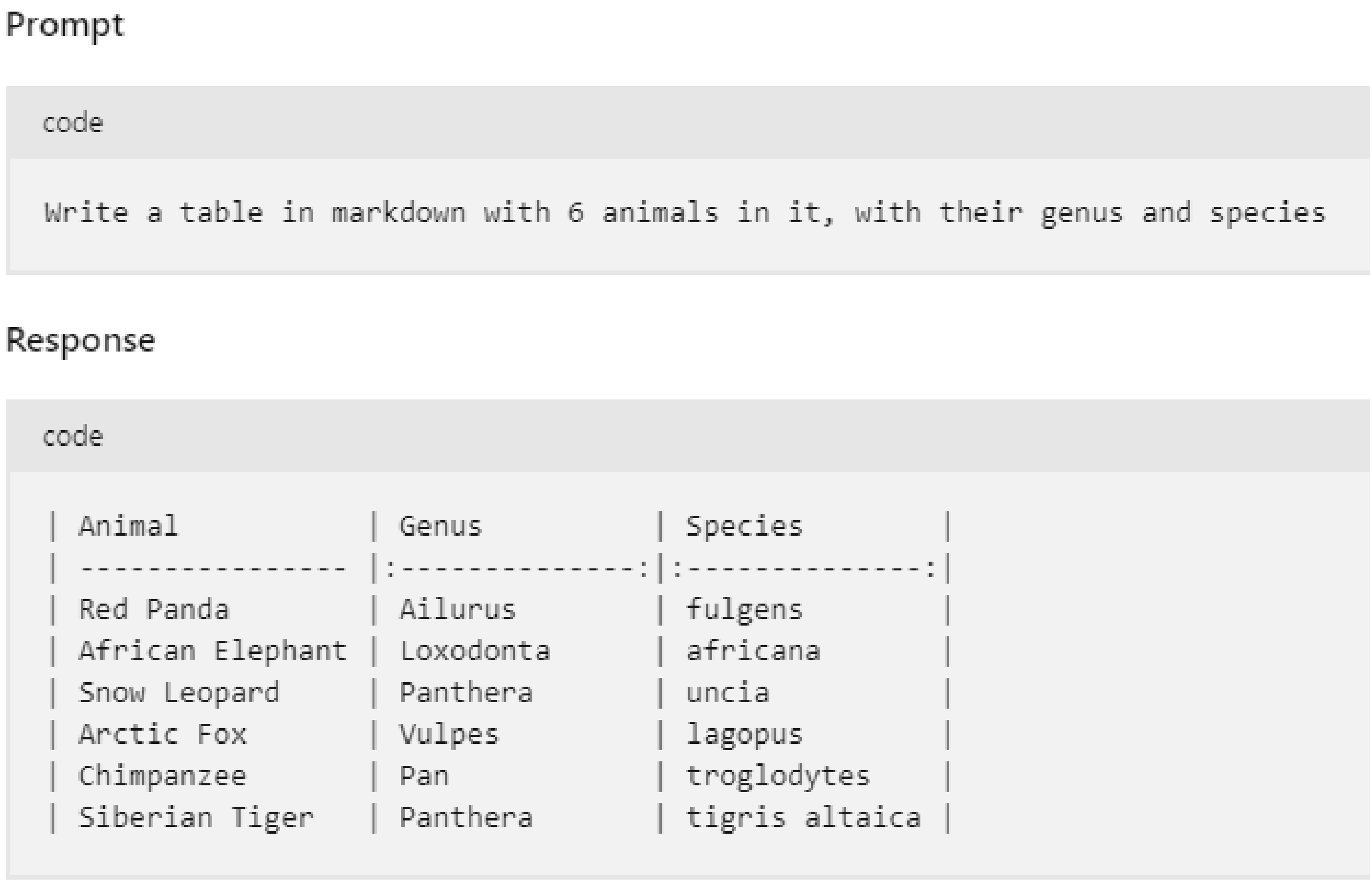
Cues

- If you're wanting help creating a SQL query, provide instructions of what you need along with the beginning of the query:

Write a join query to get customer names with purchases in the past 30 days between tables named orders and customer on customer ID.

SELECT

Request output composition



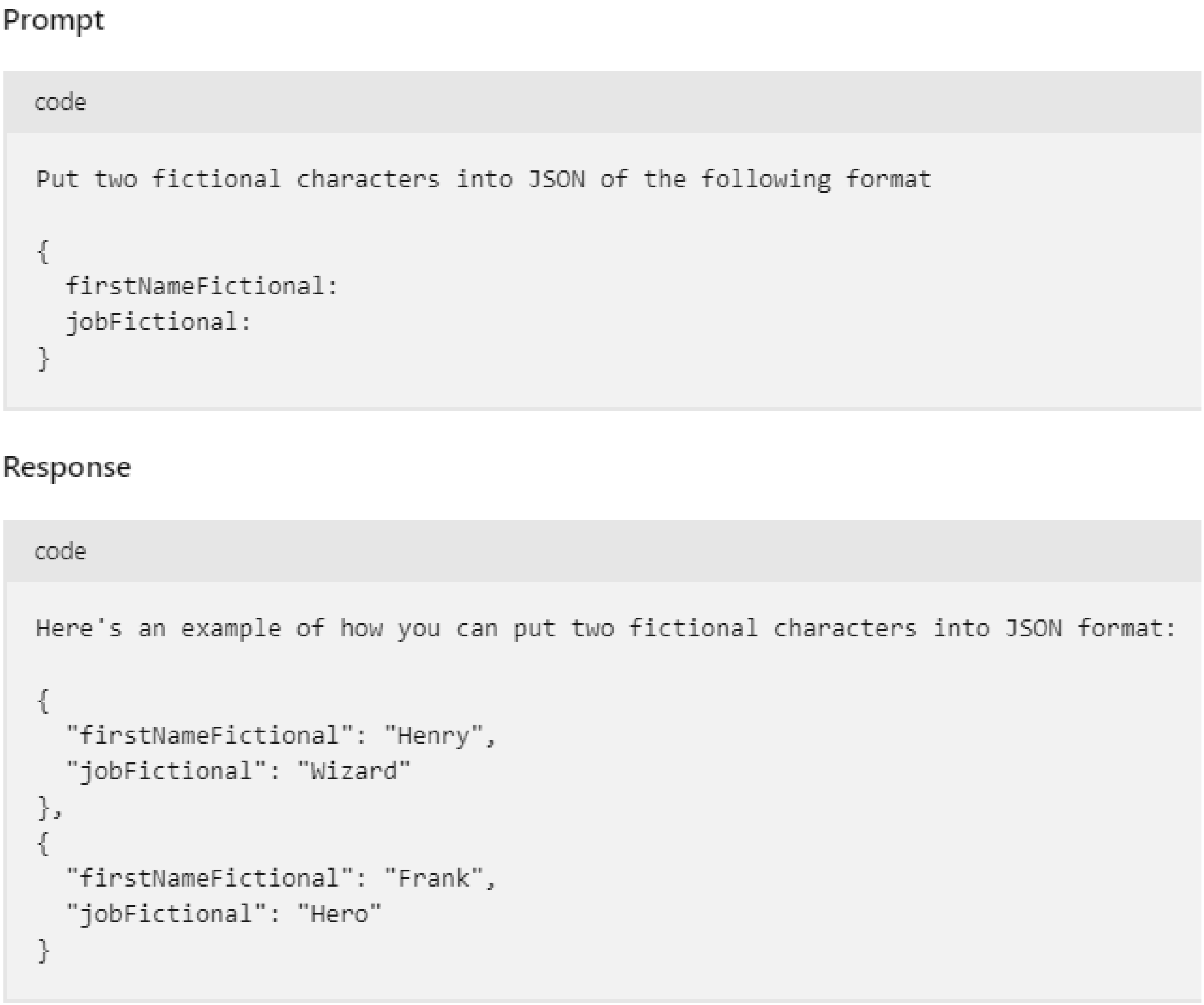
Prompt

```
Write a table in markdown with 6 animals in it, with their genus and species
```

Response

```
| Animal           | Genus          | Species        |
| ---------------- |:--------------:|:--------------:|
| Red Panda        | Ailurus        | fulgens        |
| African Elephant | Loxodonta      | africana       |
| Snow Leopard     | Panthera       | uncia          |
| Arctic Fox       | Vulpes         | lagopus        |
| Chimpanzee       | Pan            | troglodytes    |
| Siberian Tiger   | Panthera       | tigris altaica |
```

Request output composition



Prompt

code

```
Put two fictional characters into JSON of the following format

{
  firstNameFictional:
  jobFictional:
}
```

Response

code

```
Here's an example of how you can put two fictional characters into JSON format:

{
  "firstNameFictional": "Henry",
  "jobFictional": "Wizard"
},
{
  "firstNameFictional": "Frank",
  "jobFictional": "Hero"
}
```

System message

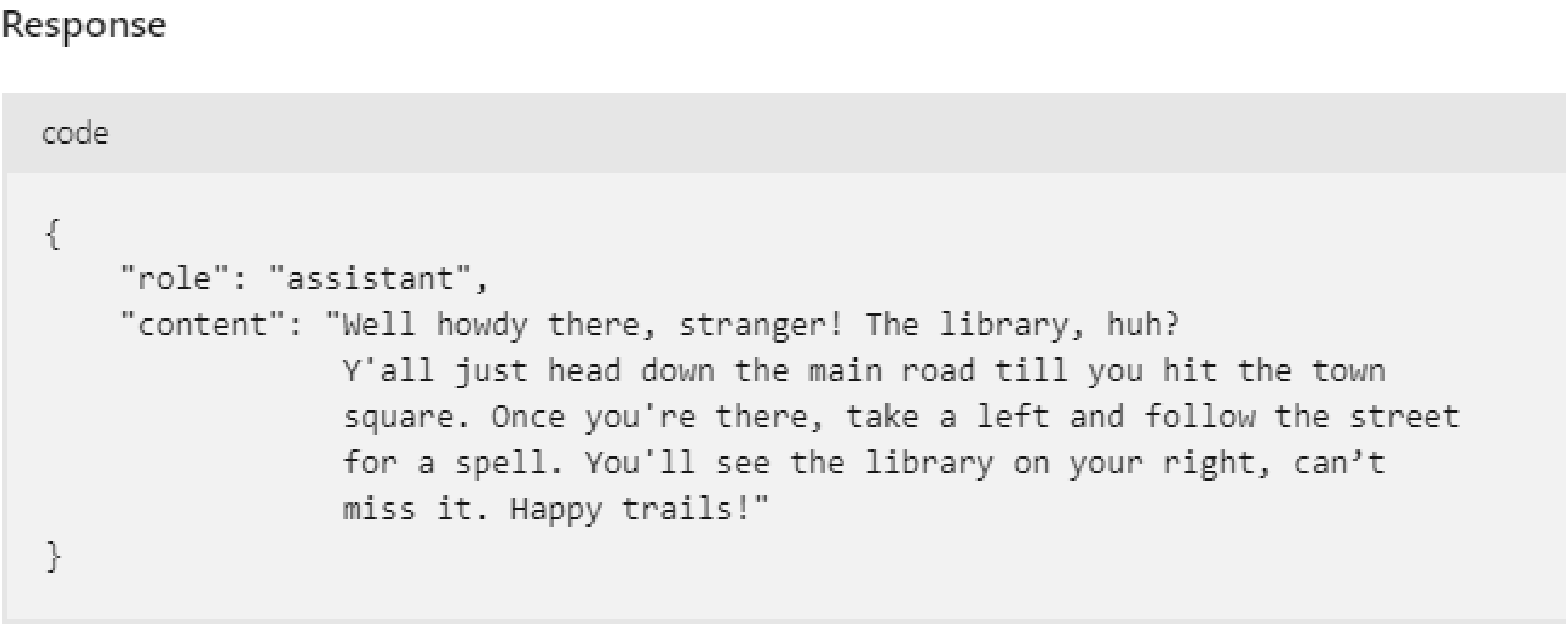
You could give it some of the following system messages:

- "I want you to act like a command line terminal. Respond to commands exactly as cmd.exe would, in one unique code block, and nothing else."
- "I want you to be a translator, from English to Spanish. Don't respond to anything I say or ask, only translate between those two languages and reply with the translated text."
- "Act as a motivational speaker, freely giving out encouraging advice about goals and challenges. You should include lots of positive affirmations and suggested activities for reaching the user's end goal."

System message

```
var ChatCompletionsOptions = new ChatCompletionsOptions()
 {
      Messages =
      {
            new ChatMessage(ChatRole.System, "You are a casual, helpful assistant. You will talk like an American old
western film character."),
          new ChatMessage(ChatRole.User, "Can you direct me to the library?")
      }
 };
```

System message



Response

code

```
{
    "role": "assistant",
    "content": "Well howdy there, stranger! The library, huh?
                Y'all just head down the main road till you hit the town
                square. Once you're there, take a left and follow the street
                for a spell. You'll see the library on your right, can’t
                miss it. Happy trails!"
}
```

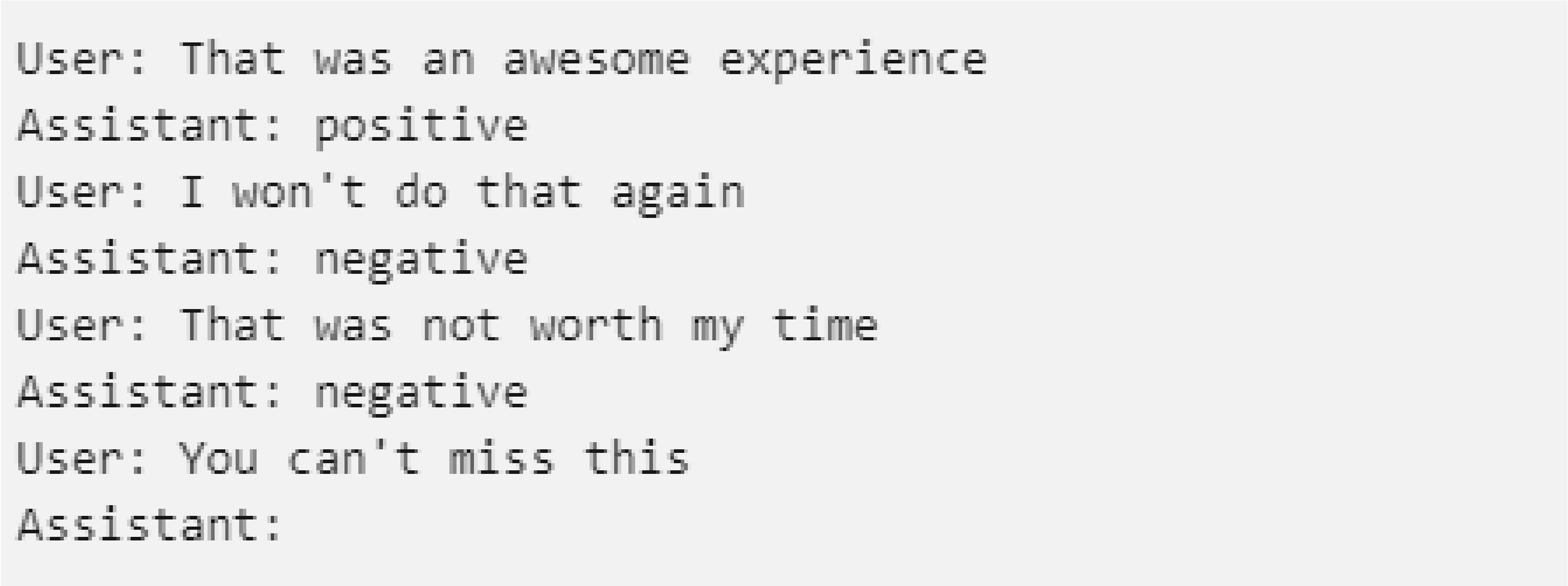
Conversation history

- Chat interfaces that use OpenAI models, such as ChatGPT and the chat playground in Azure OpenAI Studio, include conversation history automatically which results in a richer, more meaningful conversation.

- In the Parameters section below the chat window of the Azure OpenAI Studio chat playground, you can specify how many past messages you want included.

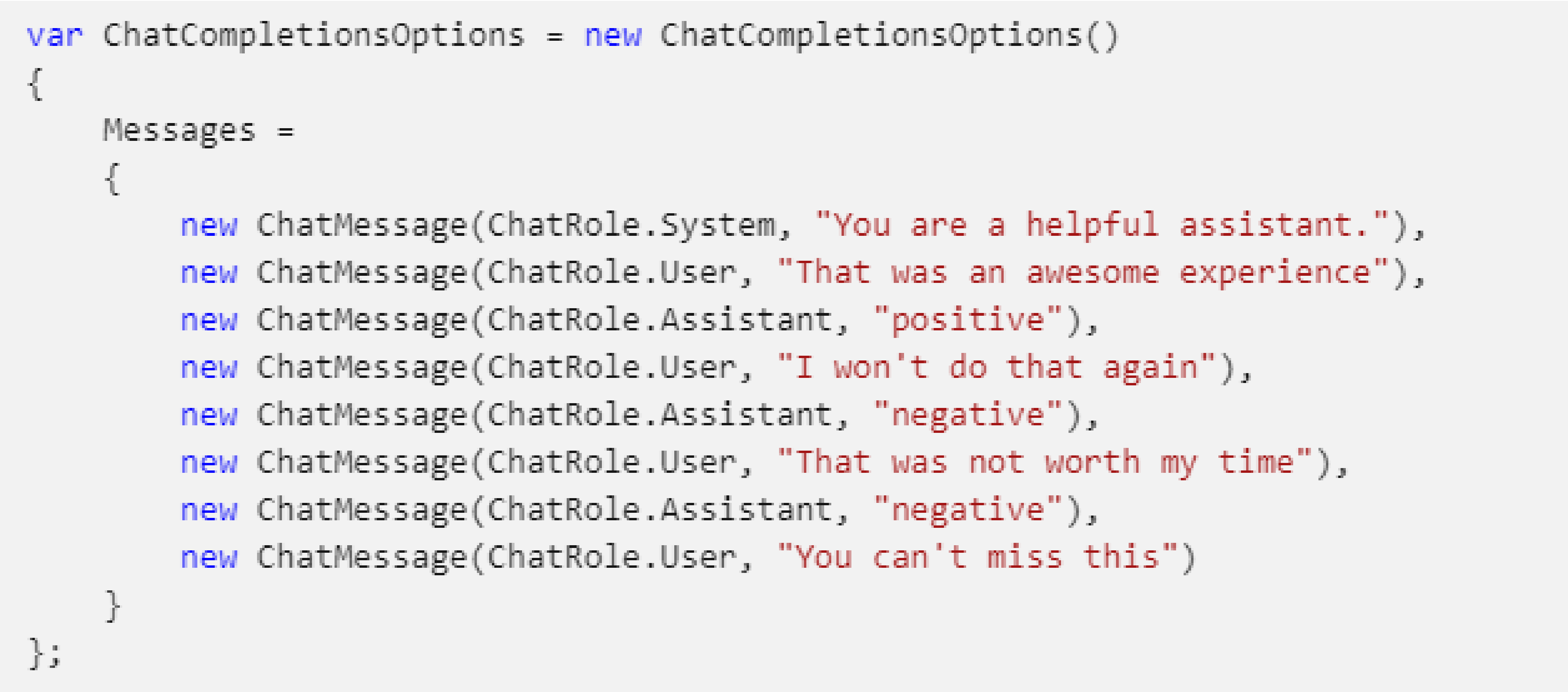
- Try reducing that to 1 or increasing to max to see how different amounts of history impact the conversation.

Few shot learning



```
User: That was an awesome experience
Assistant: positive
User: I won't do that again
Assistant: negative
User: That was not worth my time
Assistant: negative
User: You can't miss this
Assistant:
```

Few shot learning



```
var ChatCompletionsOptions = new ChatCompletionsOptions()
{
    Messages =
    {
        new ChatMessage(ChatRole.System, "You are a helpful assistant."),
        new ChatMessage(ChatRole.User, "That was an awesome experience"),
        new ChatMessage(ChatRole.Assistant, "positive"),
        new ChatMessage(ChatRole.User, "I won't do that again"),
        new ChatMessage(ChatRole.Assistant, "negative"),
        new ChatMessage(ChatRole.User, "That was not worth my time"),
        new ChatMessage(ChatRole.Assistant, "negative"),
        new ChatMessage(ChatRole.User, "You can't miss this")
    }
};
```

Break down a complex task

- Another technique for improved interaction is to divide complex prompts into multiple queries.
- This allows the model to better understand each individual part, and can improve the overall accuracy.
- Dividing your prompts also allows you to include the response from a previous prompt in a future prompt, and using that information in addition to the capabilities of the model to generate interesting responses.

Chain of thought

Code:

Sure, here's a step-by-step approach to answering the question:

1. Identify sports that are popular and have a large following.
2. 2. Consider the basic mechanics of each sport and how easy they are to learn.
3. 3. Think about the various techniques and skills required to play each sport at a high level, and how difficult they are to master.
4. 4. Compare the relative ease of learning each sport to the difficulty of mastering it.
5. 5. Based on the above analysis, determine which sport is easiest to learn but hardest to master.

Using this approach, I would say that the sport of golf is easiest to learn but hardest to master. Here's why:

1. Golf is a popular sport with a large following.
2. 2. The basic mechanics of hitting a golf ball are relatively simple and can be learned quickly with some practice.
3. 3. However, achieving consistency and mastering the various shots and techniques required to play at a high level can take years of practice and dedication.
4. 4. Compared to other sports, the difference between a beginner and an intermediate golfer is relatively small, while the difference between an intermediate and an advanced golfer is much larger.

5. Therefore, I would say that golf is the sport that is easiest to learn but hardest to master.

Summary

- In this module, you learned about prompt engineering and how it can be used to optimize the performance of Azure OpenAI models.
- Prompt engineering involves designing and optimizing prompts to better utilize AI models.
- By providing relevant, specific, unambiguous, and well-structured prompts, the model can better understand the context and generate more accurate responses.

4. Generate code with Azure OpenAI Service

Introduction

- The Azure OpenAI service enables you to use large language models (LLMs) to generate content, including programming code.
- This functionality allows developers to generate and improve existing programming code in various languages to increase efficiency and understanding.
- In this module, you'll learn how to use Azure OpenAI to generate code and help various development tasks.

Construct code from natural language

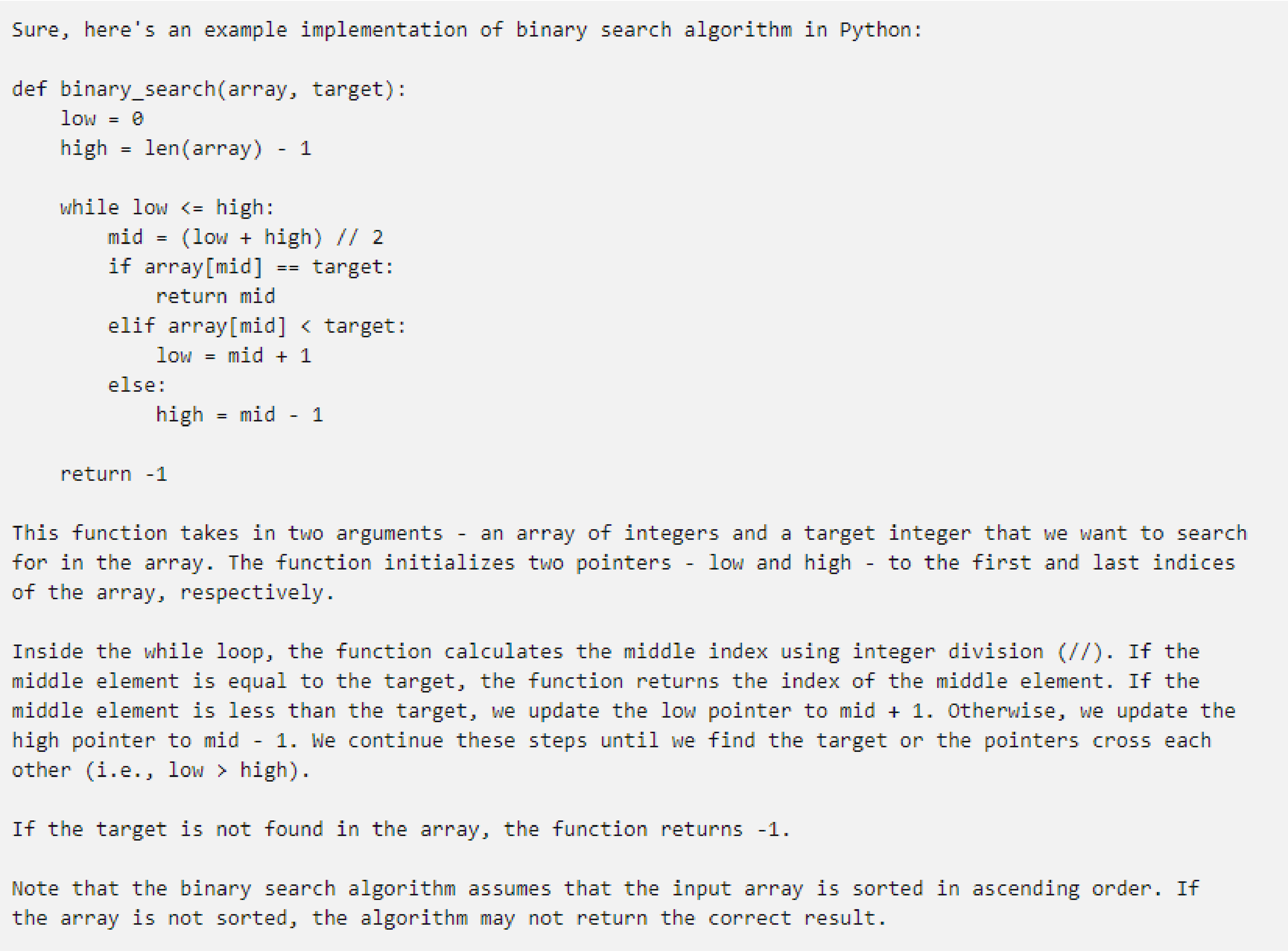
- One of the capabilities of Azure OpenAI models is to generate code from natural language prompts.
- Tasks can range from a simple one line command to a full application.
- The AI models can also edit and update provided code or previous responses to complete the requested task.

AI models for code generation

- As new generations of models evolve, the base models drastically improve their performance and understanding of both language and code, which results in not needing specific *Codex* models.

- This improvement results in just a single model for more recent generations (such as gpt-35-turbo and gpt-4) that can be used for both natural language and code.

Write functions



Sure, here's an example implementation of binary search algorithm in Python:

```
def binary_search(array, target):
    low = 0
    high = len(array) - 1

    while low <= high:
        mid = (low + high) // 2
        if array[mid] == target:
            return mid
        elif array[mid] < target:
            low = mid + 1
        else:
            high = mid - 1

    return -1
```

This function takes in two arguments - an array of integers and a target integer that we want to search for in the array. The function initializes two pointers - low and high - to the first and last indices of the array, respectively.

Inside the while loop, the function calculates the middle index using integer division (//). If the middle element is equal to the target, the function returns the index of the middle element. If the middle element is less than the target, we update the low pointer to mid + 1. Otherwise, we update the high pointer to mid - 1. We continue these steps until we find the target or the pointers cross each other (i.e., low > high).

If the target is not found in the array, the function returns -1.

Note that the binary search algorithm assumes that the input array is sorted in ascending order. If the array is not sorted, the algorithm may not return the correct result.

Change coding language

- If you have code in one language, but need it in another, Azure OpenAI can translate that for you. For example, say you have this print_squares function in Python.

```
def print_squares(n):
    for i in range(1, n+1):
        print(i**2)
```

Change coding language

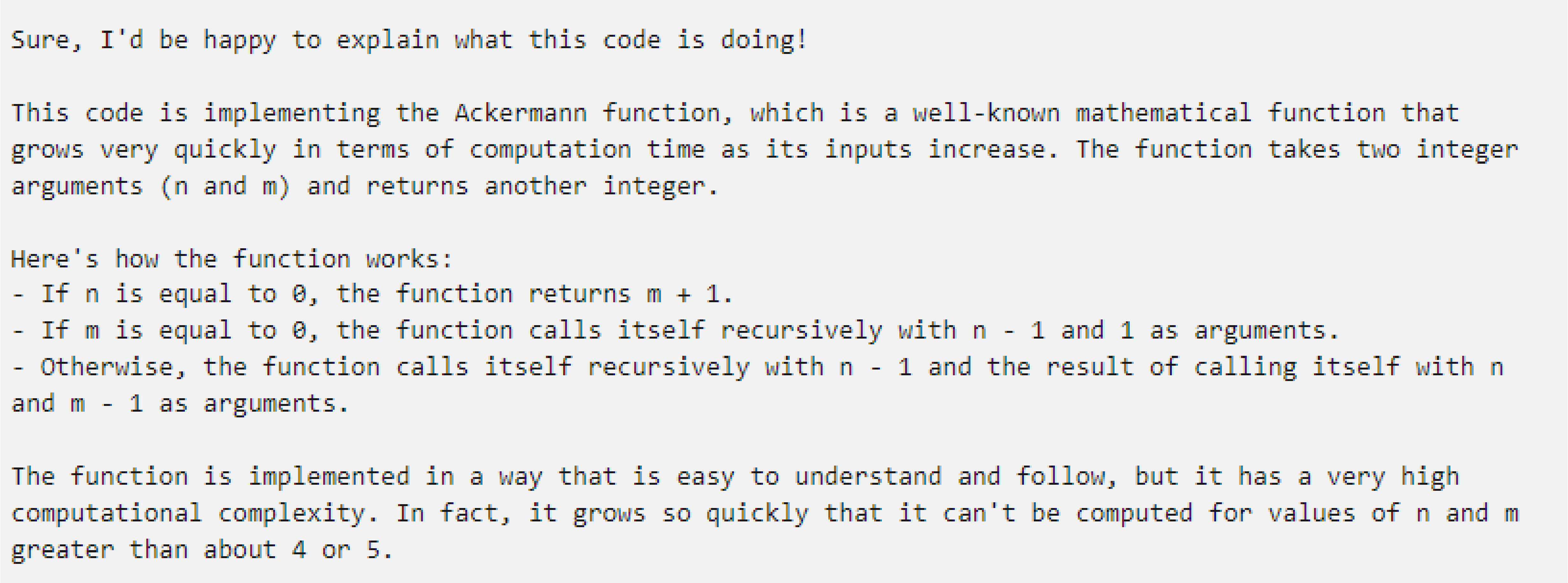
- If you provide that code, along with the prompt convert this code to C#, you likely get something similar to the following function.

```
public void PrintSquares(int n)
{
    for (int i = 1; i <= n; i++)
    {
        Console.WriteLine(i * i);
    }
}
```

Understand unknown code

```
fn qzplv(n: i32, m: i32) -> i32 {
    if n == 0 {
        return m + 1;
    } else if m == 0 {
        return qzplv(n - 1, 1);
    } else {
        return qzplv(n - 1, qzplv(n, m - 1));
    }
}
```

Understand unknown code



Sure, I'd be happy to explain what this code is doing!

This code is implementing the Ackermann function, which is a well-known mathematical function that grows very quickly in terms of computation time as its inputs increase. The function takes two integer arguments (n and m) and returns another integer.

Here's how the function works:
- If n is equal to 0, the function returns m + 1.
- If m is equal to 0, the function calls itself recursively with n - 1 and 1 as arguments.
- Otherwise, the function calls itself recursively with n - 1 and the result of calling itself with n and m - 1 as arguments.

The function is implemented in a way that is easy to understand and follow, but it has a very high computational complexity. In fact, it grows so quickly that it can't be computed for values of n and m greater than about 4 or 5.

Complete code and assist the development process

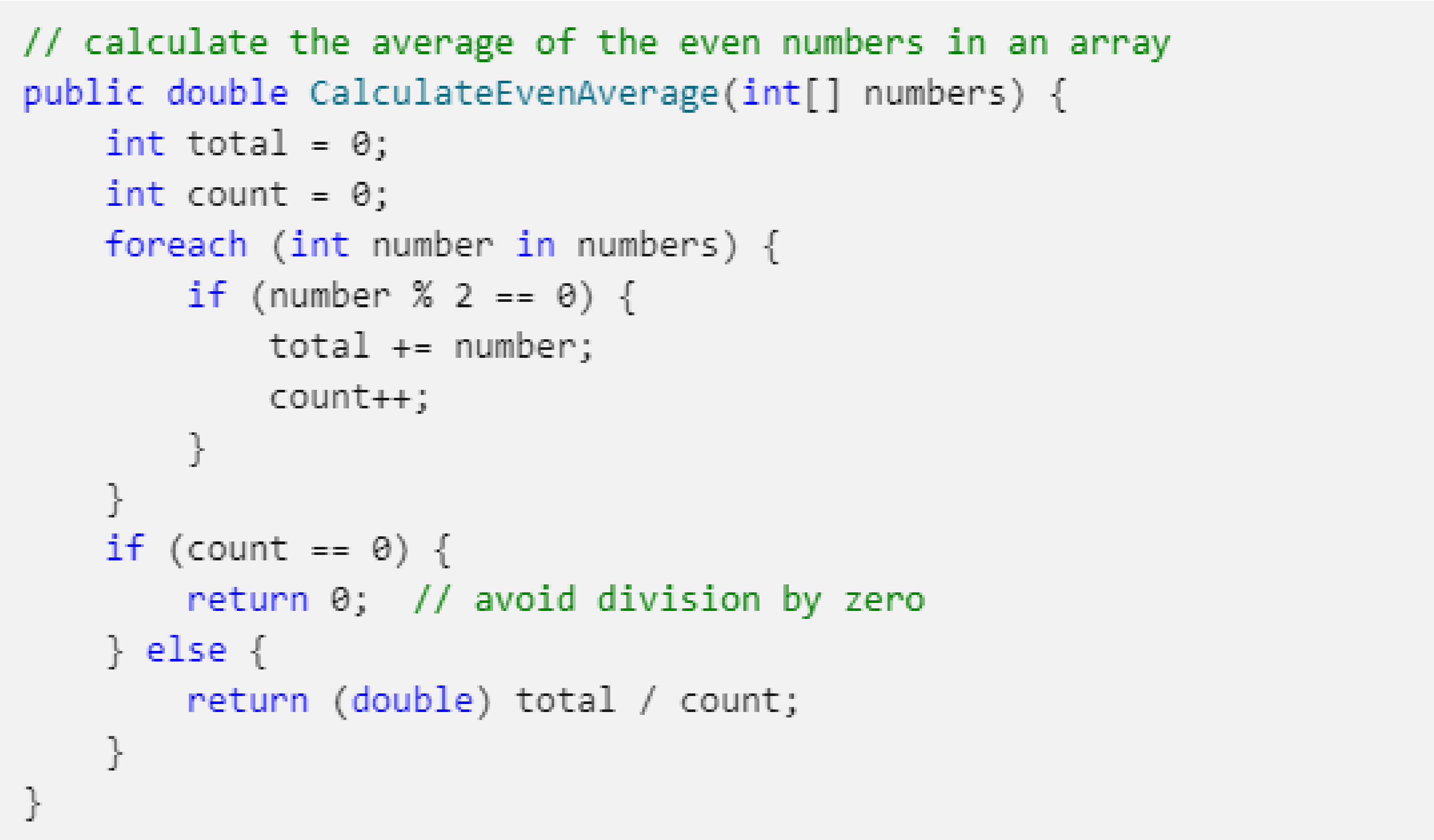
- Azure OpenAI can assist developers in common software development tasks such as writing unit tests, completing partial code, commenting code, and generating documentation.
- Using AI assistance can enable developers to spend more time on complex programming and problem solving tasks.

Complete partial code

For example, when given the following prompt:

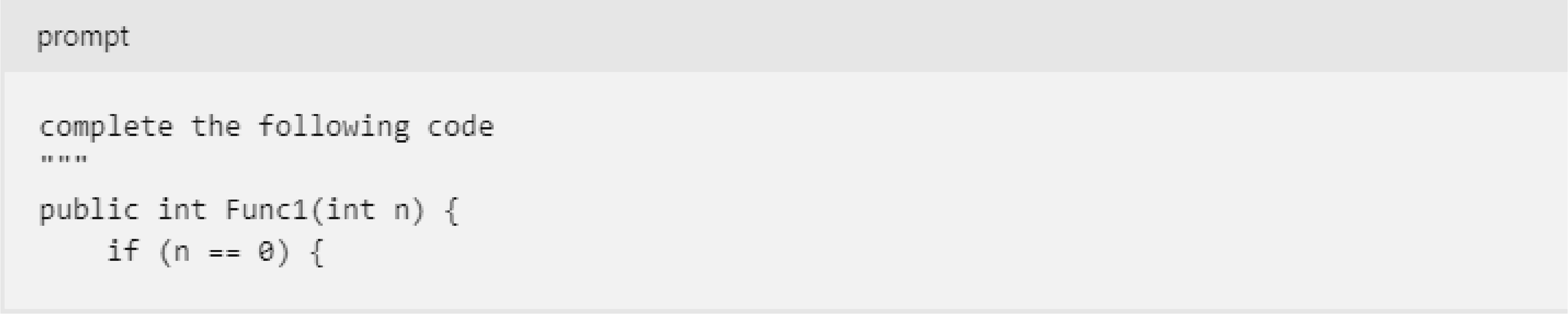
```
complete the following function
"""
// calculate the average of the numbers in an array, but only if they're even
public
```

Complete partial code



```
// calculate the average of the even numbers in an array
public double CalculateEvenAverage(int[] numbers) {
    int total = 0;
    int count = 0;
    foreach (int number in numbers) {
        if (number % 2 == 0) {
            total += number;
            count++;
        }
    }
    if (count == 0) {
        return 0;  // avoid division by zero
    } else {
        return (double) total / count;
    }
}
```

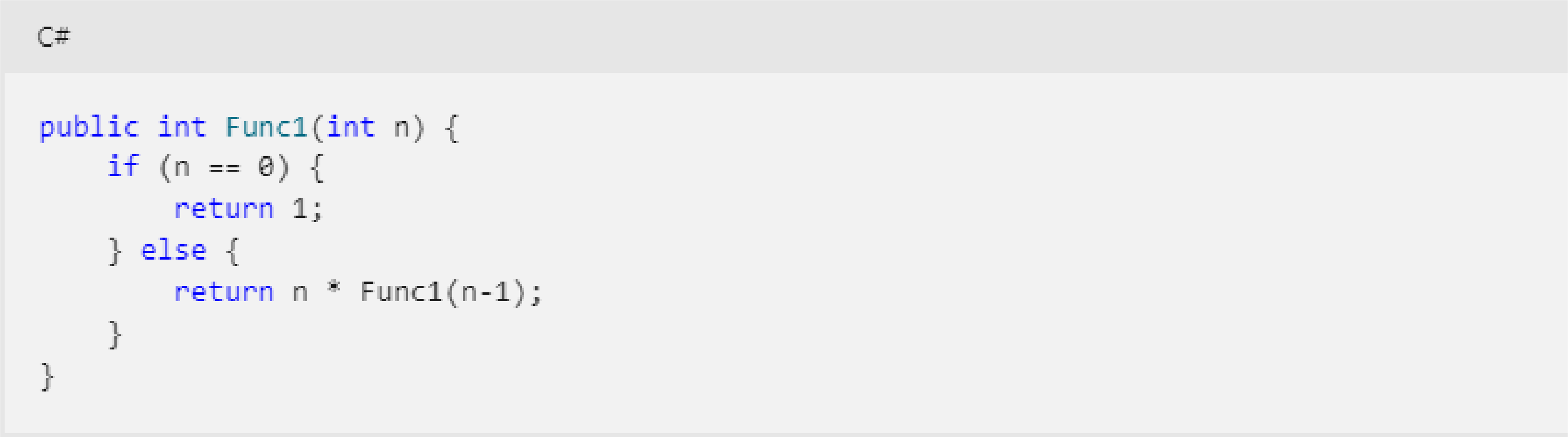
Complete partial code



prompt

```
complete the following code
"""
public int Func1(int n) {
    if (n == 0) {
```

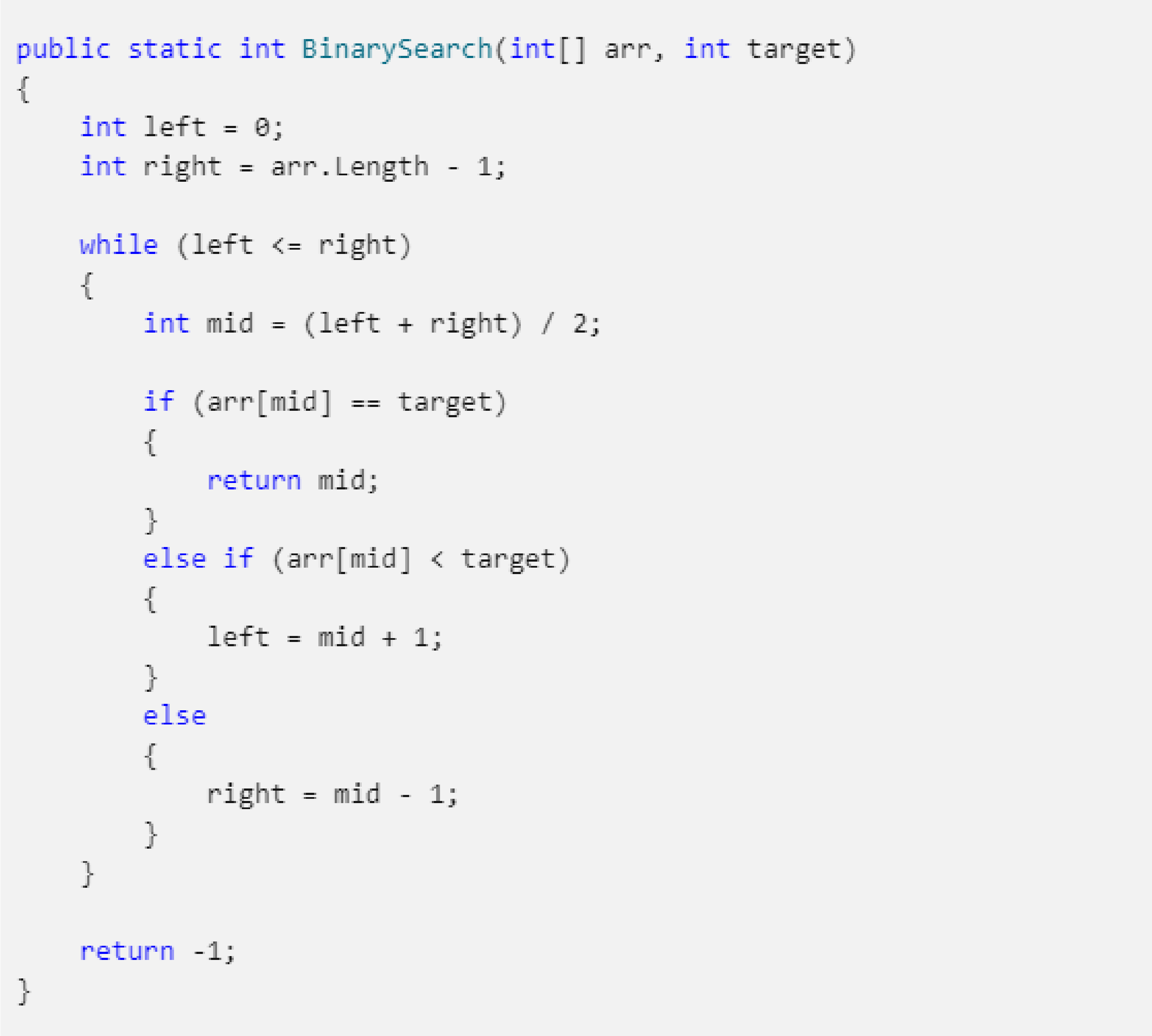
The AI model does its best to complete the function with the most likely completion of the function.



C#

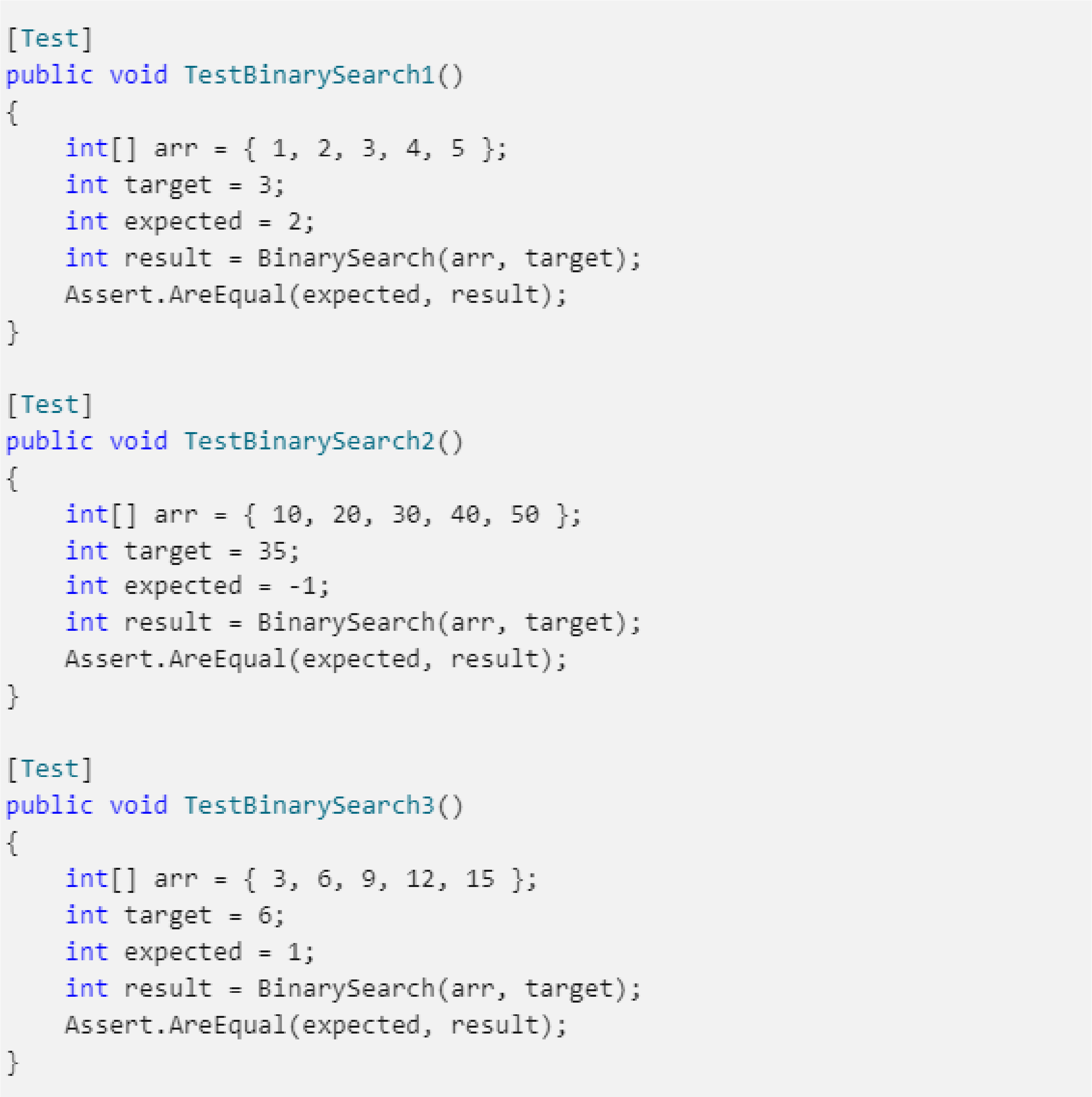
```csharp
public int Func1(int n) {
    if (n == 0) {
        return 1;
    } else {
        return n * Func1(n-1);
    }
}
```

Write unit tests



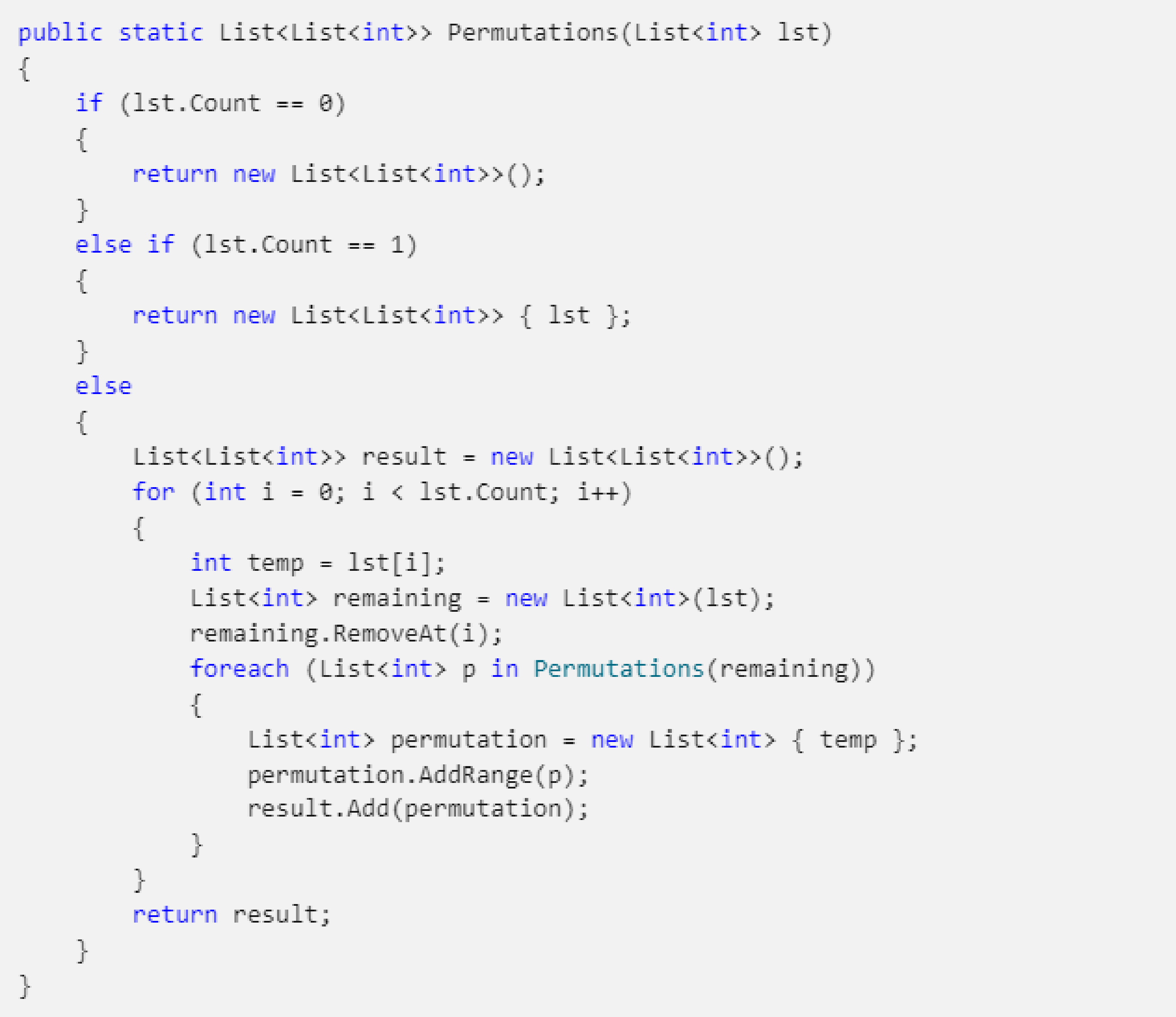
```
public static int BinarySearch(int[] arr, int target)
{
    int left = 0;
    int right = arr.Length - 1;

    while (left <= right)
    {
        int mid = (left + right) / 2;

        if (arr[mid] == target)
        {
            return mid;
        }
        else if (arr[mid] < target)
        {
            left = mid + 1;
        }
        else
        {
            right = mid - 1;
        }
    }

    return -1;
}
```

Write unit tests



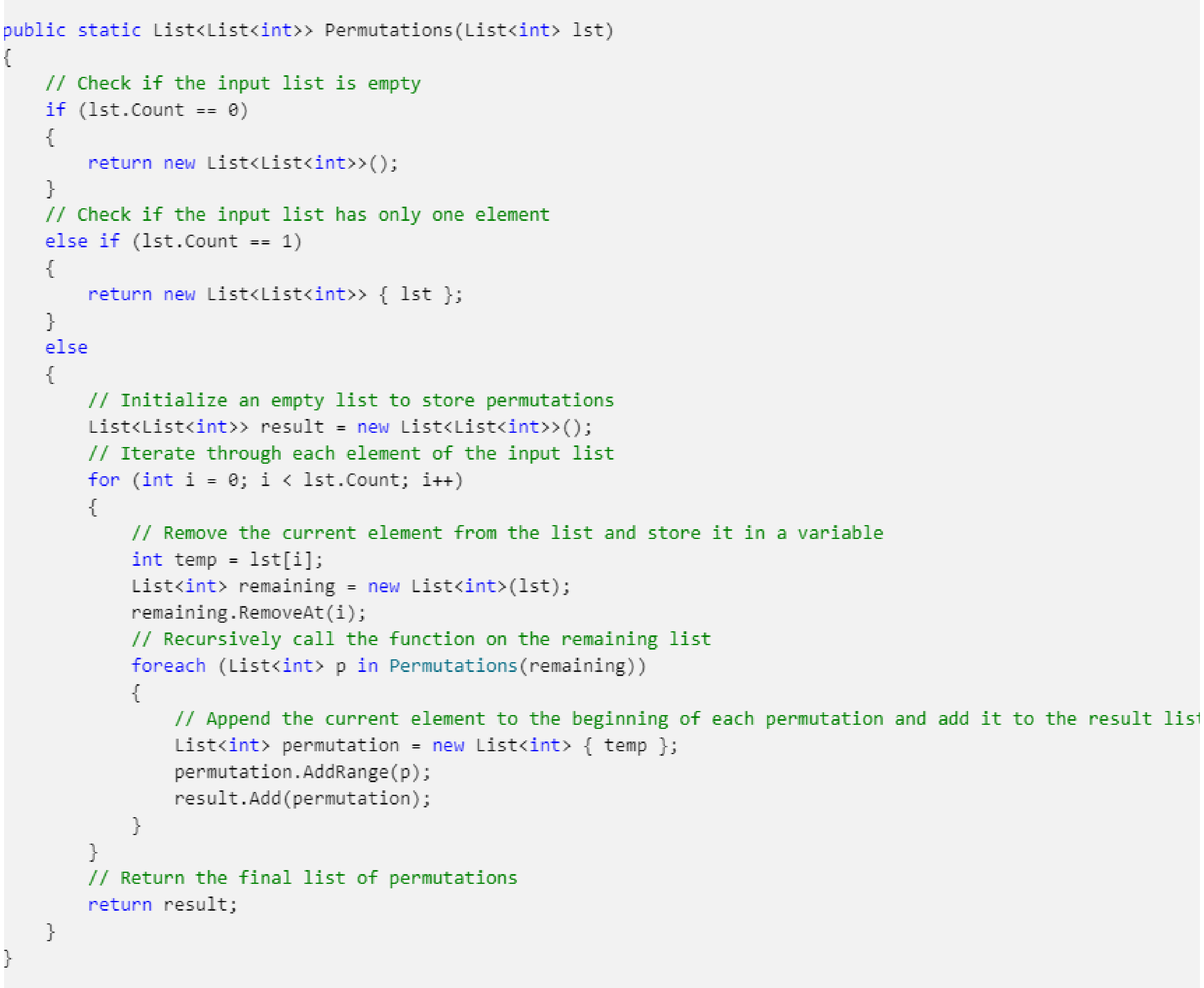
```
[Test]
public void TestBinarySearch1()
{
    int[] arr = { 1, 2, 3, 4, 5 };
    int target = 3;
    int expected = 2;
    int result = BinarySearch(arr, target);
    Assert.AreEqual(expected, result);
}

[Test]
public void TestBinarySearch2()
{
    int[] arr = { 10, 20, 30, 40, 50 };
    int target = 35;
    int expected = -1;
    int result = BinarySearch(arr, target);
    Assert.AreEqual(expected, result);
}

[Test]
public void TestBinarySearch3()
{
    int[] arr = { 3, 6, 9, 12, 15 };
    int target = 6;
    int expected = 1;
    int result = BinarySearch(arr, target);
    Assert.AreEqual(expected, result);
}
```

Add comments and generate documentation



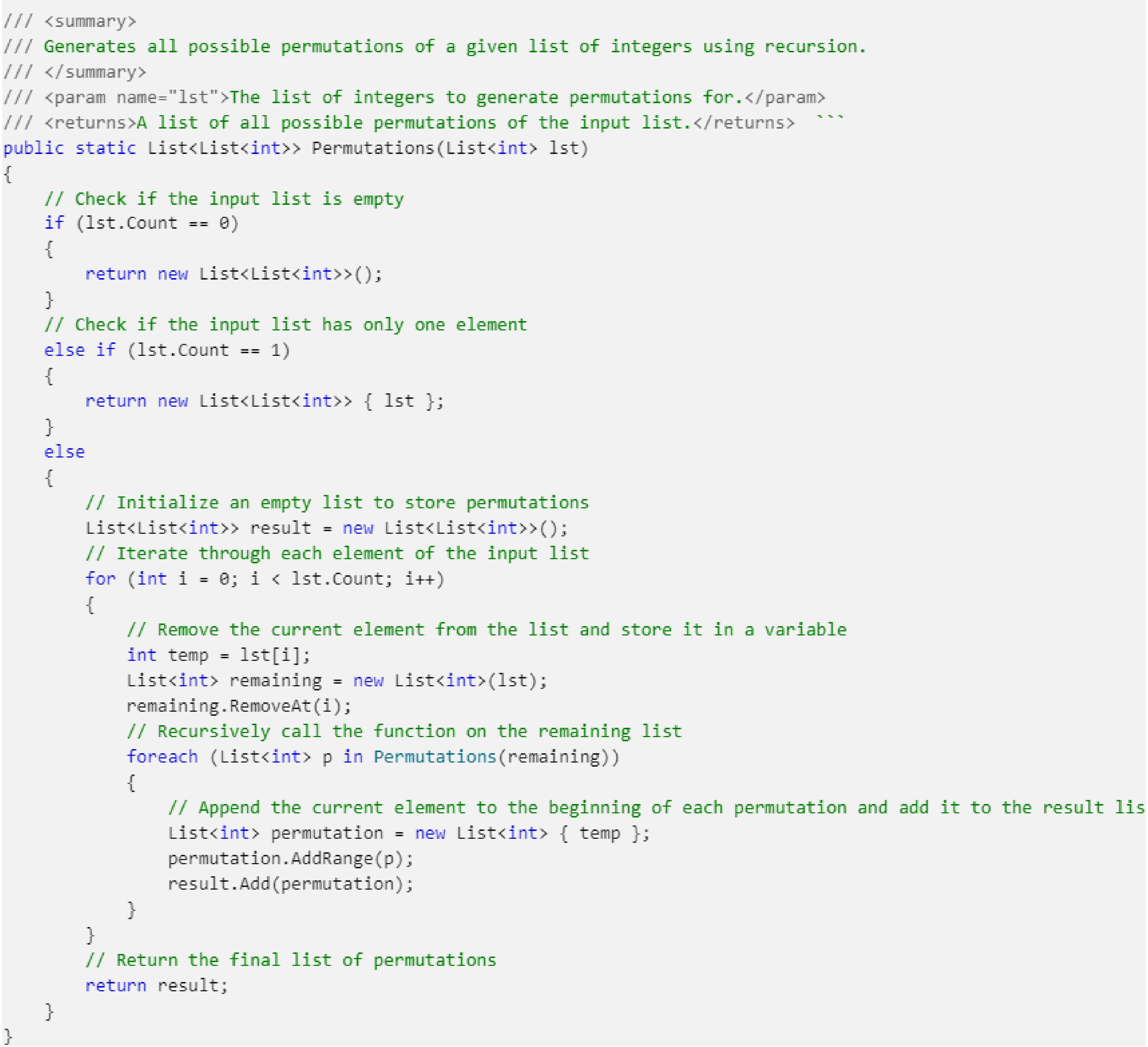
```
public static List<List<int>> Permutations(List<int> lst)
{
    if (lst.Count == 0)
    {
        return new List<List<int>>();
    }
    else if (lst.Count == 1)
    {
        return new List<List<int>> { lst };
    }
    else
    {
        List<List<int>> result = new List<List<int>>();
        for (int i = 0; i < lst.Count; i++)
        {
            int temp = lst[i];
            List<int> remaining = new List<int>(lst);
            remaining.RemoveAt(i);
            foreach (List<int> p in Permutations(remaining))
            {
                List<int> permutation = new List<int> { temp };
                permutation.AddRange(p);
                result.Add(permutation);
            }
        }
        return result;
    }
}
```

Add comments and generate documentation



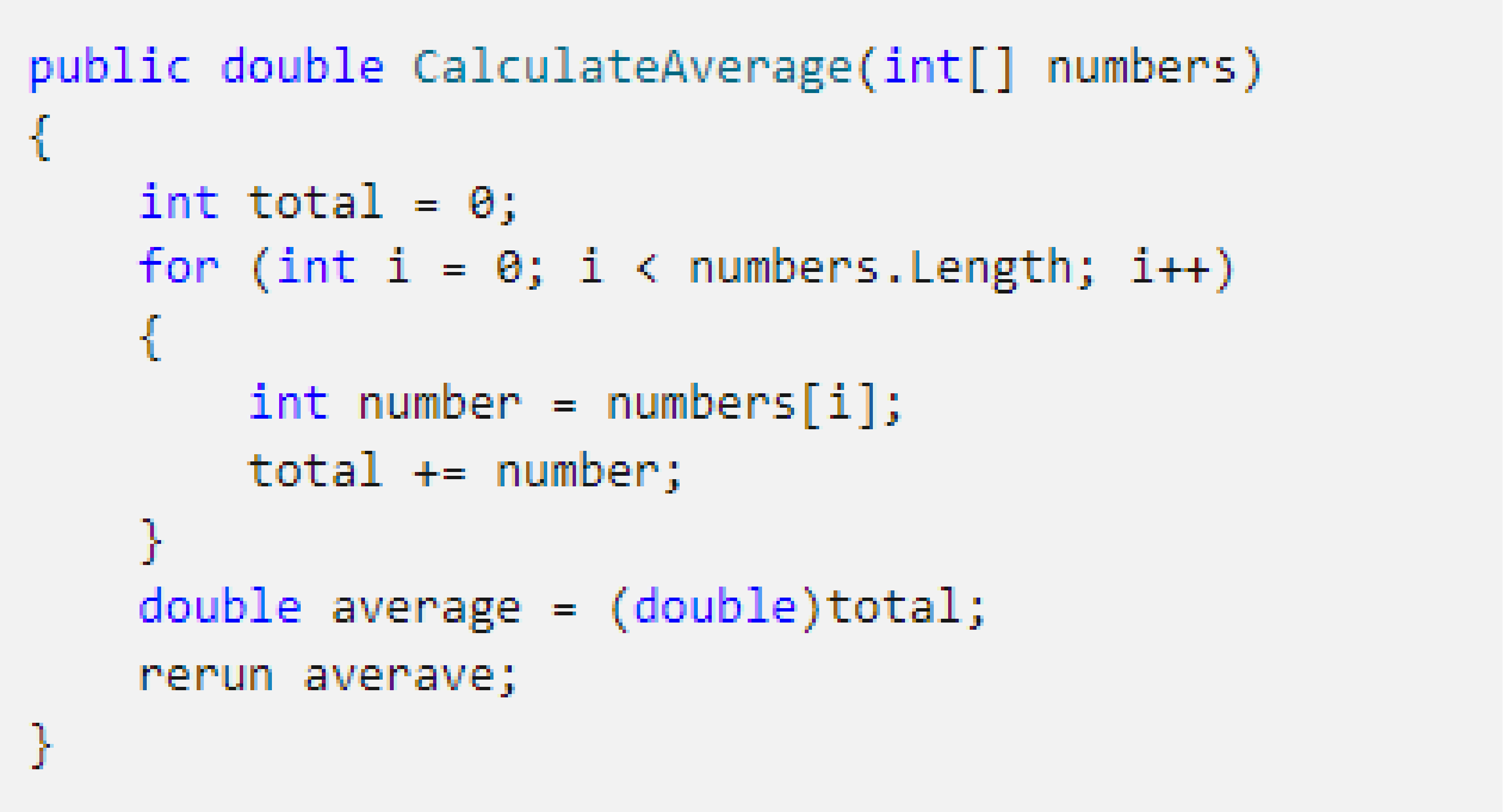
```
public static List<List<int>> Permutations(List<int> lst)
{
    // Check if the input list is empty
    if (lst.Count == 0)
    {
        return new List<List<int>>();
    }
    // Check if the input list has only one element
    else if (lst.Count == 1)
    {
        return new List<List<int>> { lst };
    }
    else
    {
        // Initialize an empty list to store permutations
        List<List<int>> result = new List<List<int>>();
        // Iterate through each element of the input list
        for (int i = 0; i < lst.Count; i++)
        {
            // Remove the current element from the list and store it in a variable
            int temp = lst[i];
            List<int> remaining = new List<int>(lst);
            remaining.RemoveAt(i);
            // Recursively call the function on the remaining list
            foreach (List<int> p in Permutations(remaining))
            {
                // Append the current element to the beginning of each permutation and add it to the result lis
                List<int> permutation = new List<int> { temp };
                permutation.AddRange(p);
                result.Add(permutation);
            }
        }
        // Return the final list of permutations
        return result;
    }
}
```

Add comments and generate documentation



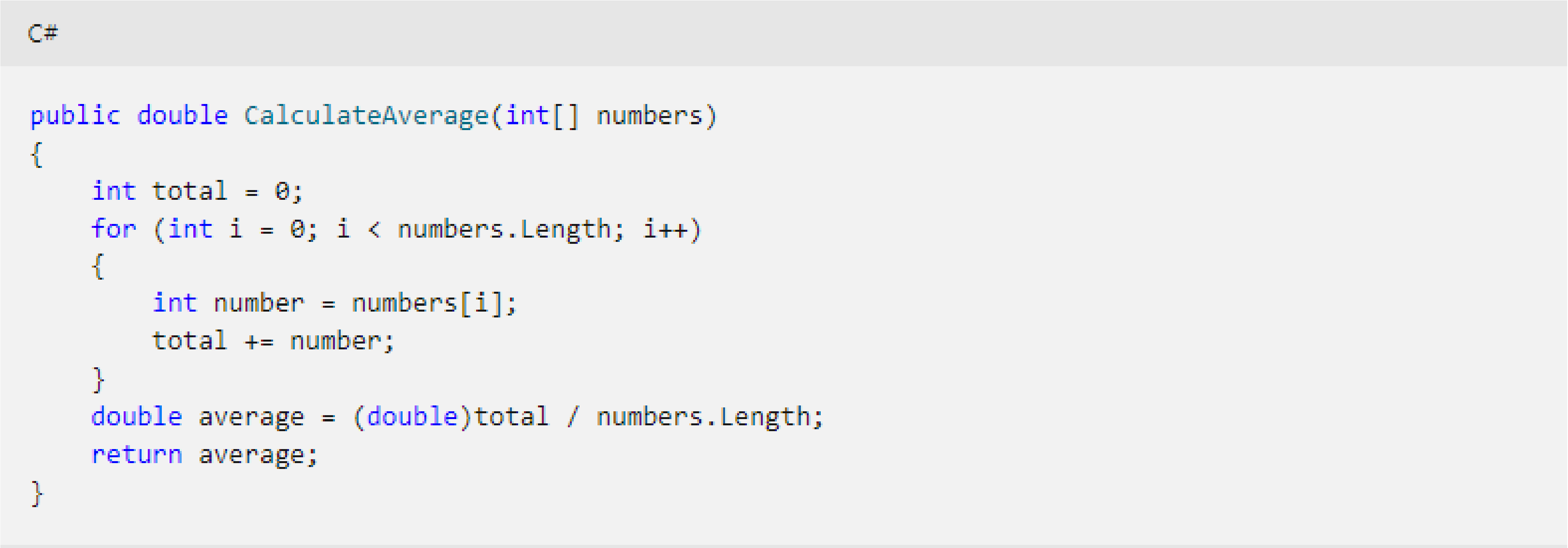
````
/// <summary>
/// Generates all possible permutations of a given list of integers using recursion.
/// </summary>
/// <param name="lst">The list of integers to generate permutations for.</param>
/// <returns>A list of all possible permutations of the input list.</returns>  ```
public static List<List<int>> Permutations(List<int> lst)
{
    // Check if the input list is empty
    if (lst.Count == 0)
    {
        return new List<List<int>>();
    }
    // Check if the input list has only one element
    else if (lst.Count == 1)
    {
        return new List<List<int>> { lst };
    }
    else
    {
        // Initialize an empty list to store permutations
        List<List<int>> result = new List<List<int>>();
        // Iterate through each element of the input list
        for (int i = 0; i < lst.Count; i++)
        {
            // Remove the current element from the list and store it in a variable
            int temp = lst[i];
            List<int> remaining = new List<int>(lst);
            remaining.RemoveAt(i);
            // Recursively call the function on the remaining list
            foreach (List<int> p in Permutations(remaining))
            {
                // Append the current element to the beginning of each permutation and add it to the result lis
                List<int> permutation = new List<int> { temp };
                permutation.AddRange(p);
                result.Add(permutation);
            }
        }
        // Return the final list of permutations
        return result;
    }
}
````

Fix bugs in your code



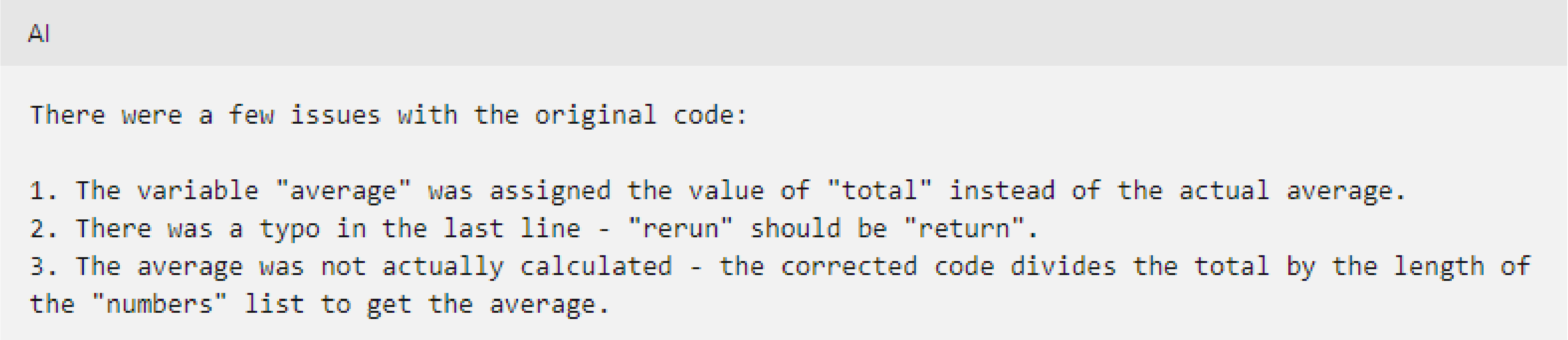
```
public double CalculateAverage(int[] numbers)
{
    int total = 0;
    for (int i = 0; i < numbers.Length; i++)
    {
        int number = numbers[i];
        total += number;
    }
    double average = (double)total;
    rerun averave;
}
```

Fix bugs in your code



C#

```
public double CalculateAverage(int[] numbers)
{
    int total = 0;
    for (int i = 0; i < numbers.Length; i++)
    {
        int number = numbers[i];
        total += number;
    }
    double average = (double)total / numbers.Length;
    return average;
}
```

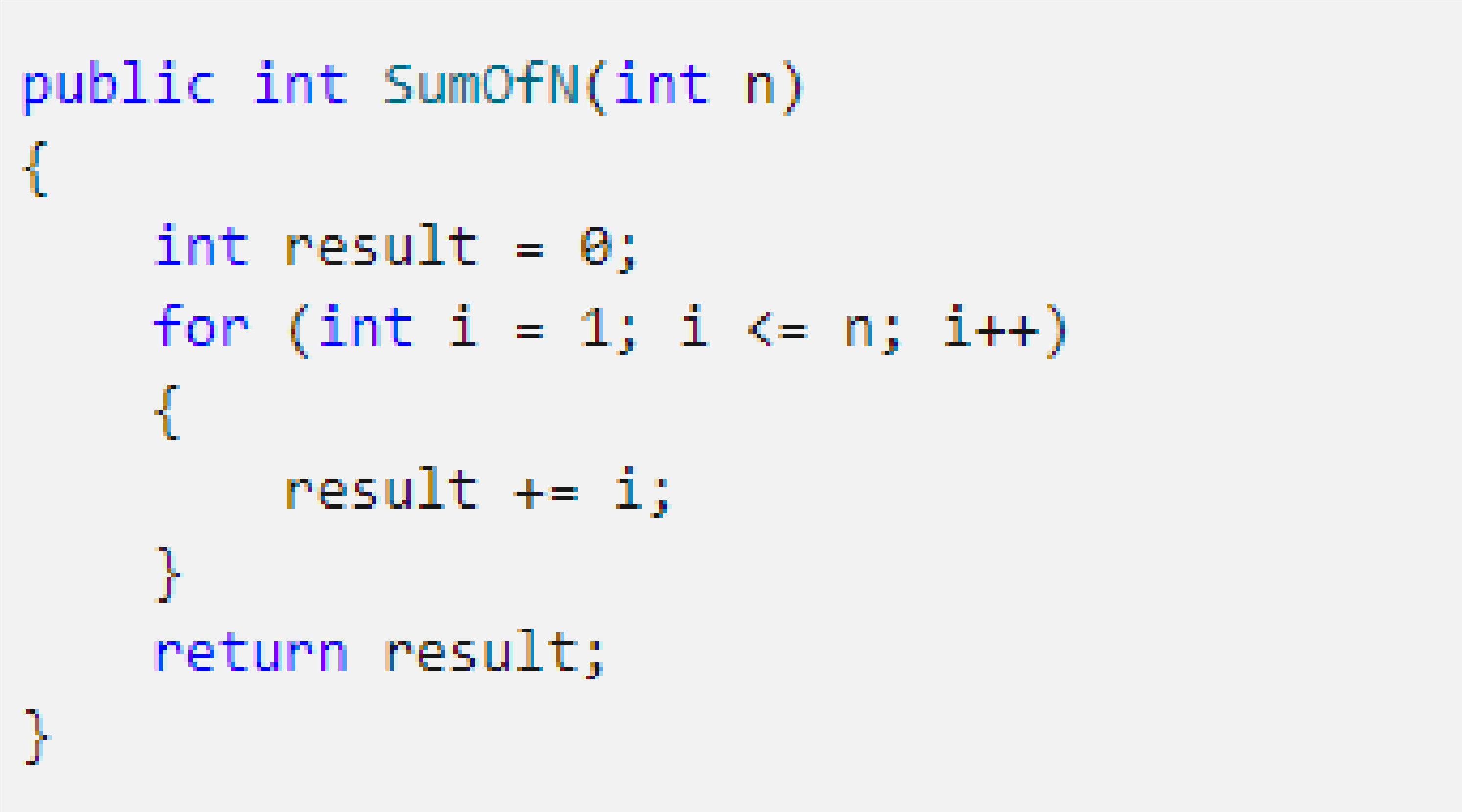


AI

There were a few issues with the original code:

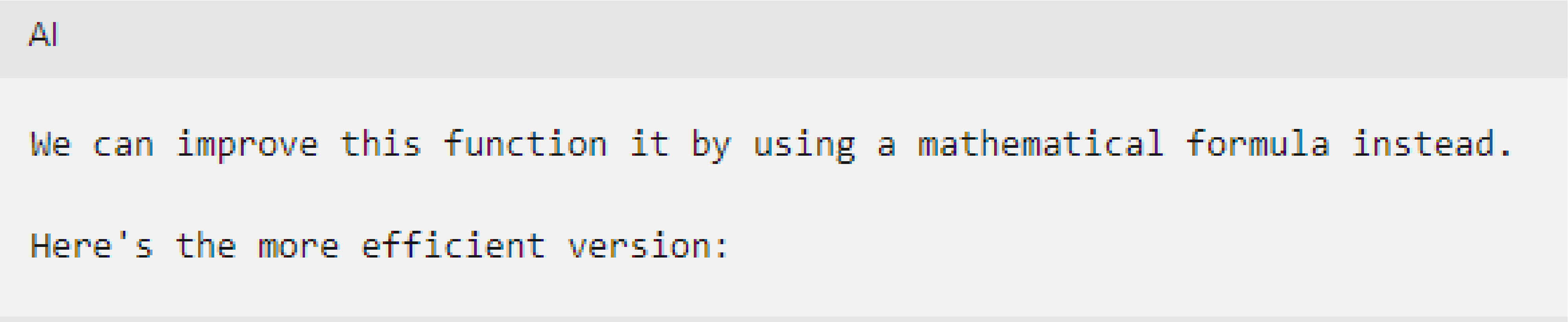
1. The variable "average" was assigned the value of "total" instead of the actual average.
2. There was a typo in the last line - "rerun" should be "return".
3. The average was not actually calculated - the corrected code divides the total by the length of the "numbers" list to get the average.

Improve performance



```
public int SumOfN(int n)
{
    int result = 0;
    for (int i = 1; i <= n; i++)
    {
        result += i;
    }
    return result;
}
```

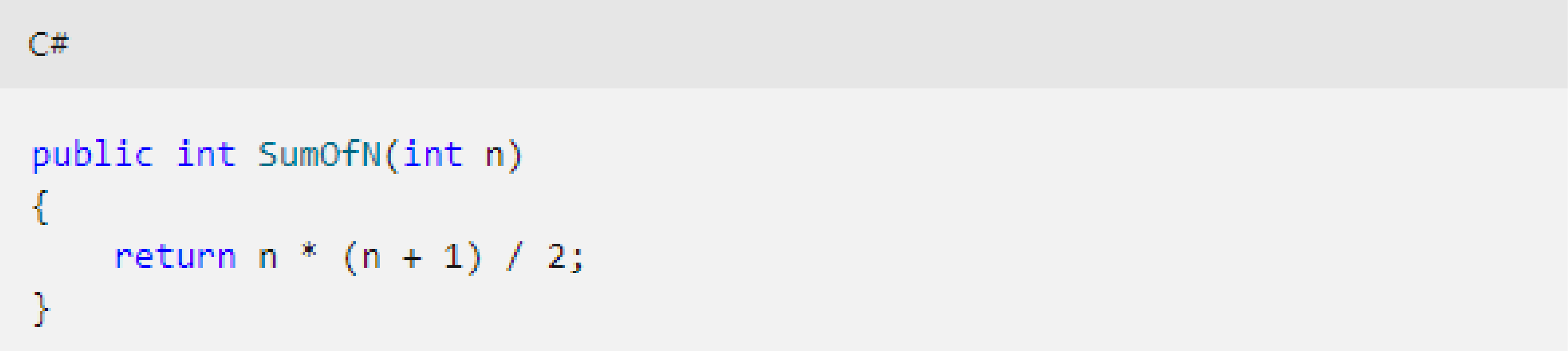
Improve performance



AI

We can improve this function it by using a mathematical formula instead.

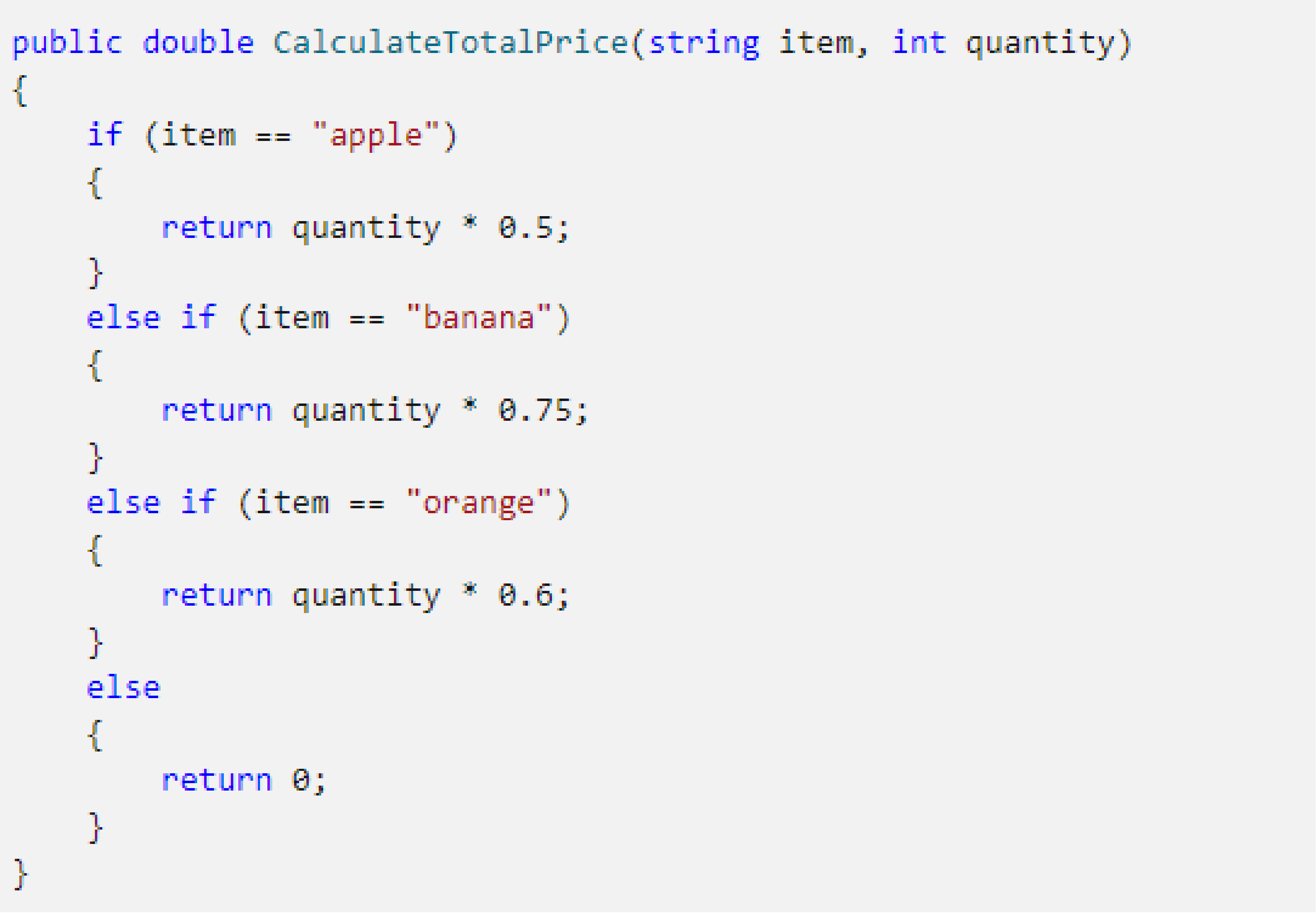
Here's the more efficient version:



C#

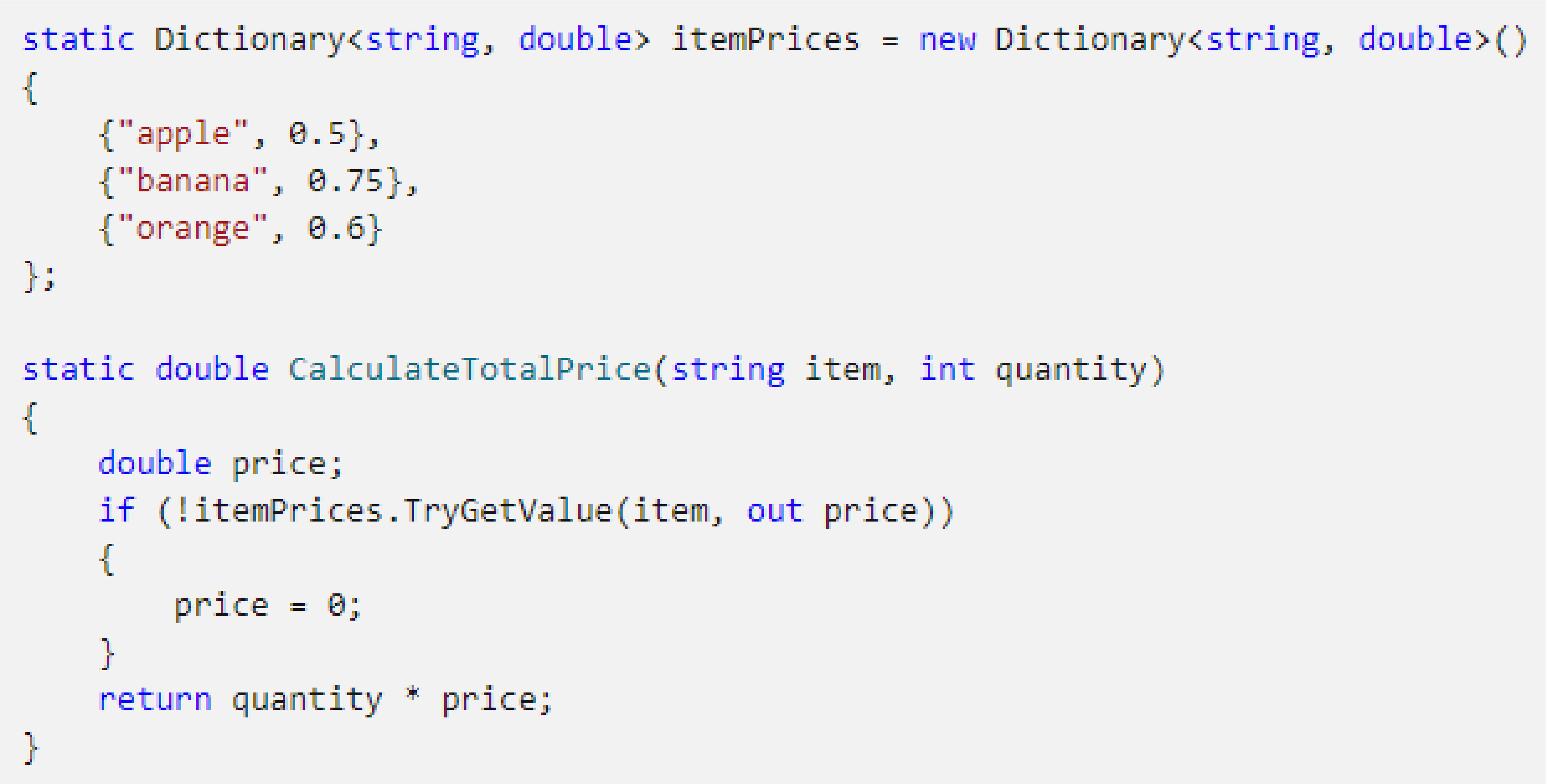
```csharp
public int SumOfN(int n)
{
    return n * (n + 1) / 2;
}
```

Refactor inefficient code



```
public double CalculateTotalPrice(string item, int quantity)
{
    if (item == "apple")
    {
        return quantity * 0.5;
    }
    else if (item == "banana")
    {
        return quantity * 0.75;
    }
    else if (item == "orange")
    {
        return quantity * 0.6;
    }
    else
    {
        return 0;
    }
}
```

Refactor inefficient code



```
static Dictionary<string, double> itemPrices = new Dictionary<string, double>()
{
    {"apple", 0.5},
    {"banana", 0.75},
    {"orange", 0.6}
};

static double CalculateTotalPrice(string item, int quantity)
{
    double price;
    if (!itemPrices.TryGetValue(item, out price))
    {
        price = 0;
    }
    return quantity * price;
}
```

Refactor inefficient code

In this refactored code, we've created an object called itemPrices that maps item names to their prices. We've also simplified the calculateTotalPrice function by using the itemPrices object to look up the price of the item, rather than using a series of if/else statements. This makes the code more modular and easier to maintain, and also reduces the risk of introducing bugs in the future.

Summary

In this module, you learned how to:

- Use natural language prompts to write code
- Build unit tests and understand complex code with AI models
- Generate comments and documentation for existing code

5. Generate images with Azure OpenAI Service

Introduction

- The Azure OpenAI service enables you to use large language models (LLMs) to generate content based on natural language prompts.

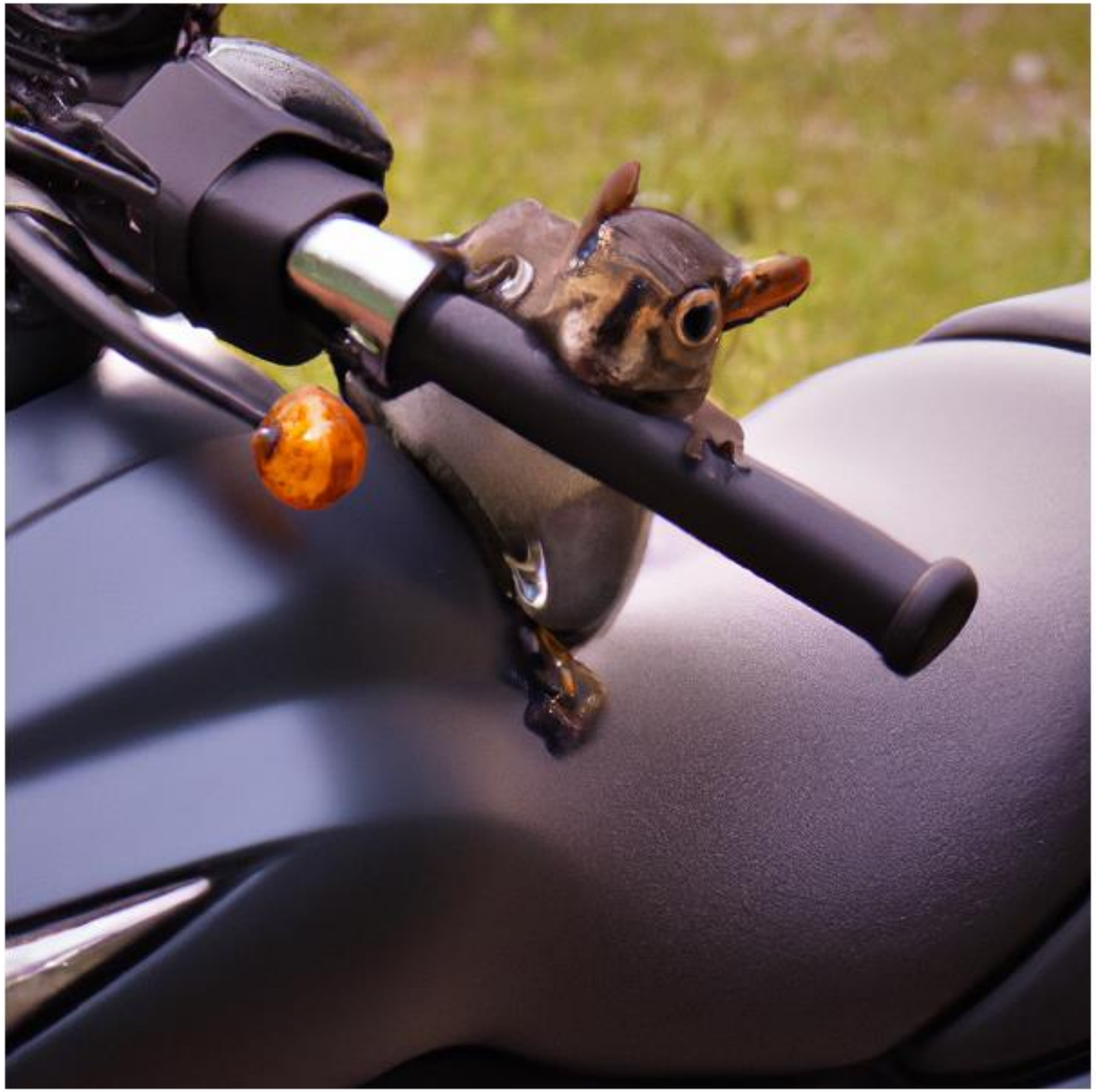
- One of these models is the DALL-E image generation model, which is capable of creating original graphical content based on natural language descriptions of a desired image.

What is DALL-E?

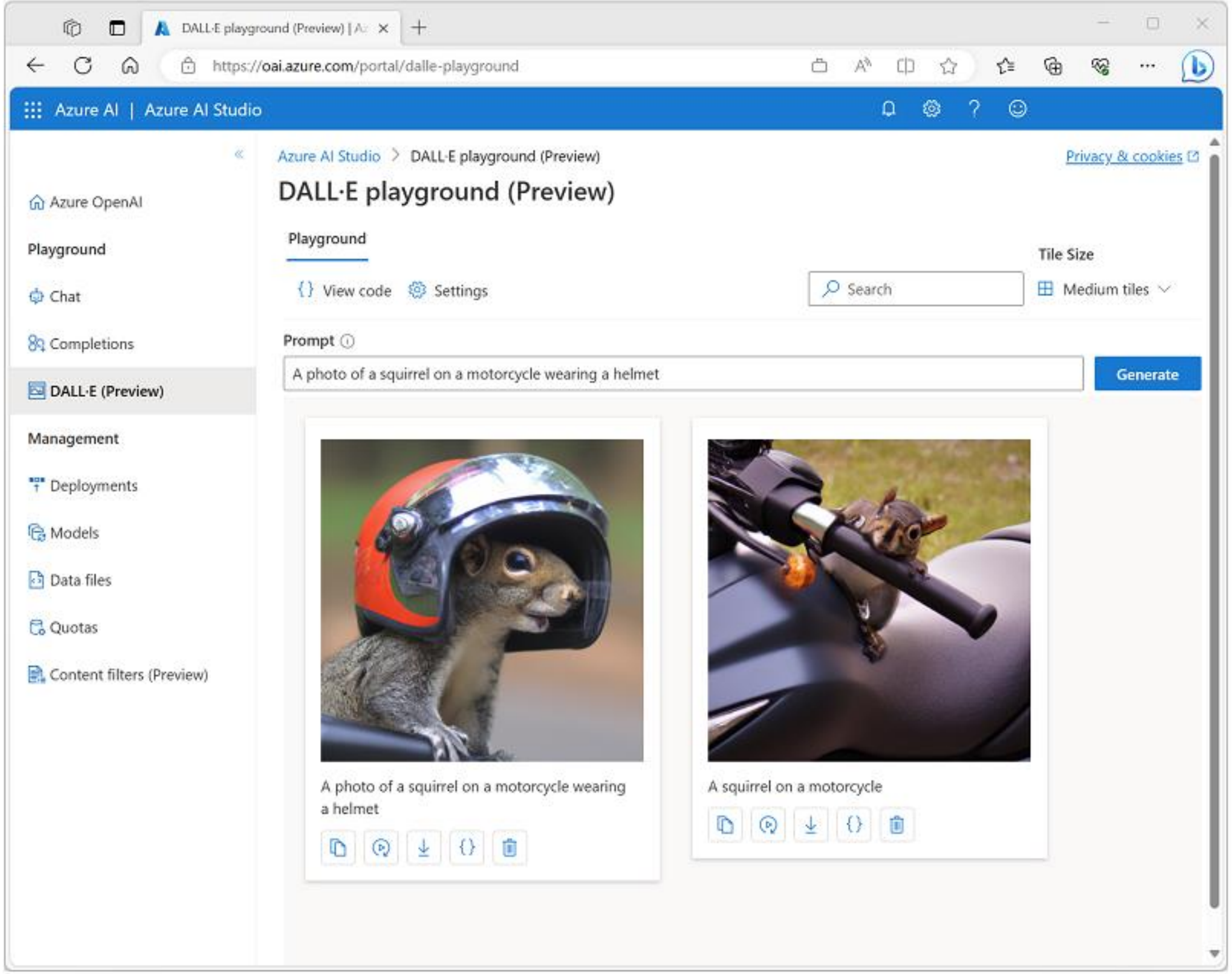
You might submit the following natural language prompt to DALL-E:

A squirrel on a motorcycle

This prompt could result in the generation of graphical output such as the following image:



Explore DALL-E in Azure OpenAI Studio



DALL·E playground (Preview)

Prompt: A photo of a squirrel on a motorcycle wearing a helmet

A photo of a squirrel on a motorcycle wearing a helmet

A squirrel on a motorcycle

Use the Azure OpenAI REST API to consume DALL-E models

- To make a REST call to the service, you need the endpoint and authorization key for the Azure OpenAI Service resource you have provisioned in Azure.
- You initiate the image generation process by submitting a POST request to the service endpoint with the authorization key in the header.
- The request must contain the following parameters in a JSON body:

1. Prompt
2. n
3. size

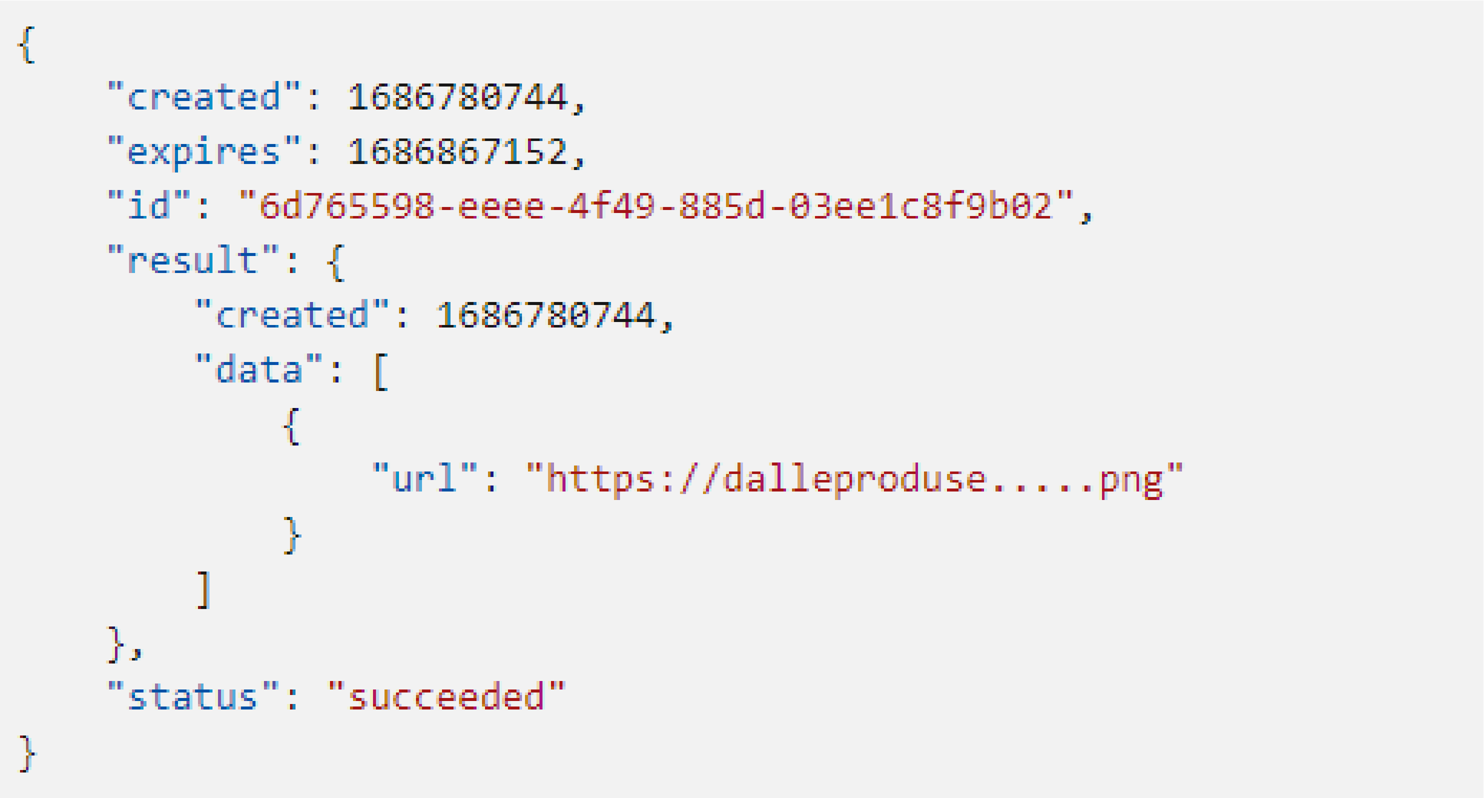
Use the Azure OpenAI REST API to consume DALL-E models

- For example, the following JSON could be used to generate an 512 x 512 image of a badger wearing a tuxedo:

```
{
   "prompt": "A badger wearing a tuxedo",
   "n": 1,
   "size": "512x512"
}
```

Use the Azure OpenAI REST API to consume DALL-E models

- When the operation has succeeded, a response similar to the following JSON is returned:



```
{
    "created": 1686780744,
    "expires": 1686867152,
    "id": "6d765598-eeee-4f49-885d-03ee1c8f9b02",
    "result": {
        "created": 1686780744,
        "data": [
            {
                "url": "https://dalleproduse.....png"
            }
        ]
    },
    "status": "succeeded"
}
```

Use the Azure OpenAI REST API to consume DALL-E models

Summary

- This module described the DALL-E image generation model, and how you can use it in the Azure OpenAI service to generate images based on natural language prompts.

- You can explore DALL-E using the playground in Azure OpenAI Studio, and you can use the REST API to build applications that use DALL-E to generate new images.

6. Use your own data with Azure OpenAI Service

Introduction

- Azure OpenAI on your data allows developers to use supported AI chat models that can reference specific sources of data to ground the response.

- In this module, you'll learn how to add your own data with Azure OpenAI and generate responses based on both that data and its pretrained knowledge.

Understand how to use your own data

Azure OpenAI on your data utilizes the search ability of Azure Cognitive Search to add the relevant data chunks to the prompt. Once your data is in a Cognitive Search index, Azure OpenAI on your data goes through the following steps:

- Receive user prompt.
- Determine relevant content and intent of the prompt.
- Query the search index with that content and intent.
- Insert search result chunk into the Azure OpenAI prompt, along with system message and user prompt.
- Send entire prompt to Azure OpenAI.
- Return response and data reference (if any) to the user.

Fine-tuning vs. using your own data

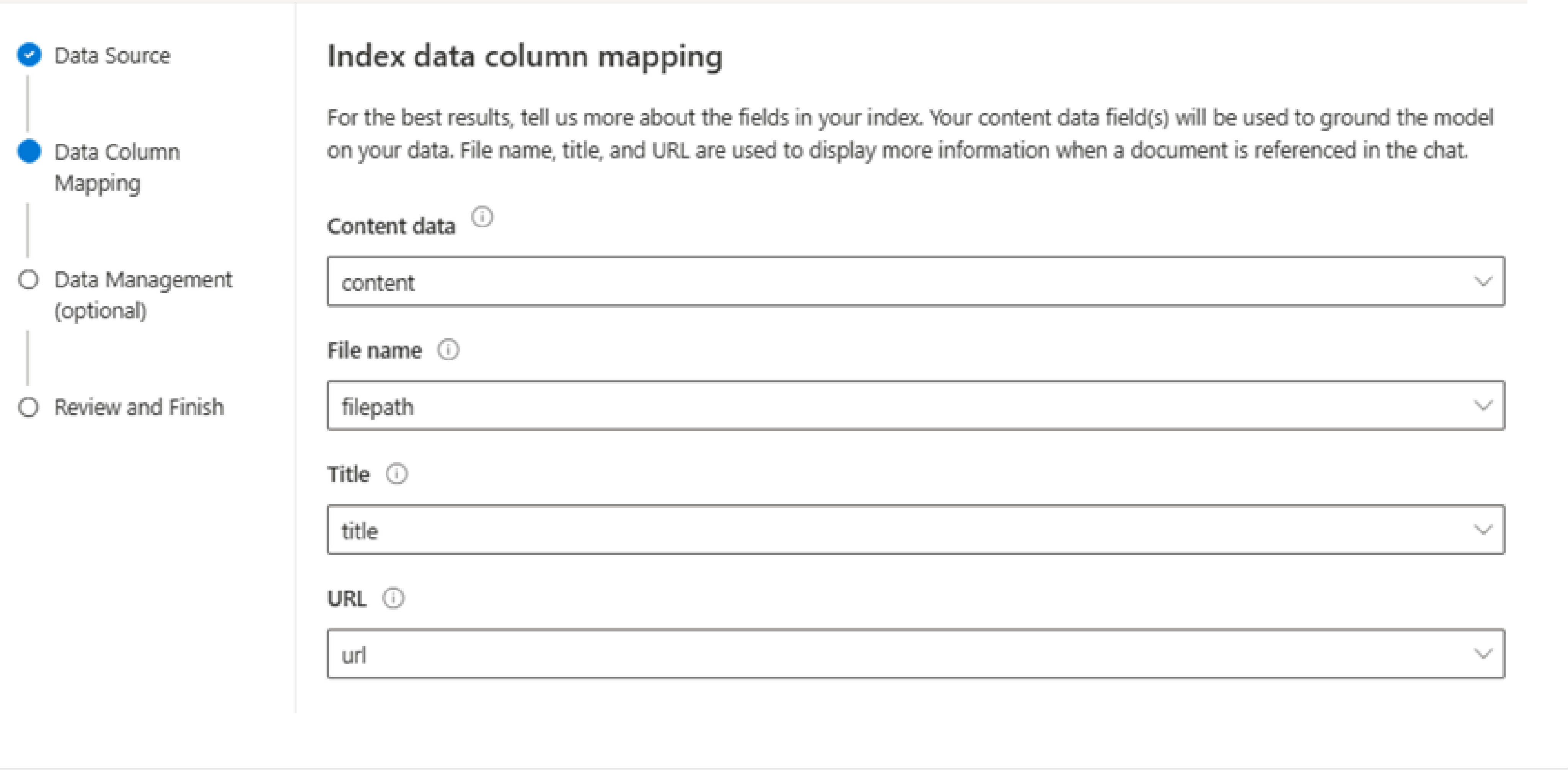
- Fine-tuning is a technique used to create a custom model by training an existing foundational model such as gpt-35-turbo with a dataset of additional training data.

- Fine-tuning can result in higher quality requests that prompt engineering alone, customize the model on examples larger than can fit in a prompt, and allow the user to provide fewer examples to get the same high quality response.

Add your own data source

- Adding your data is done through the Azure AI Studio, in the **Chat** playground.
- The data source you add is then used to augment the prompt sent to the model.
- When adding your data, you can choose to upload your data files, use data in a blob storage account, or connect to an existing Cognitive Search index.

Connect your data



Index data column mapping

For the best results, tell us more about the fields in your index. Your content data field(s) will be used to ground the model on your data. File name, title, and URL are used to display more information when a document is referenced in the chat.

Content data: content

File name: filepath

Title: title

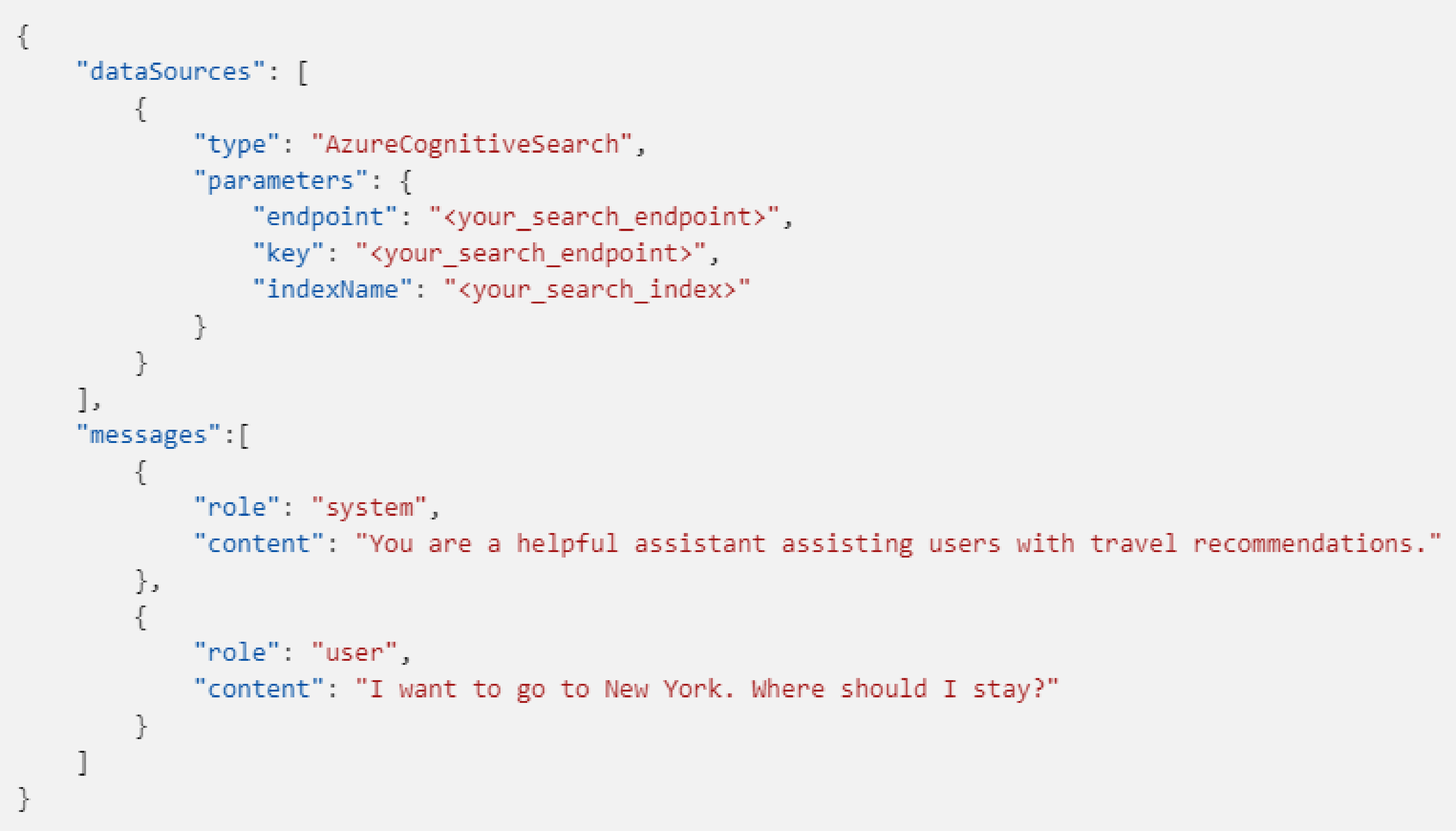
URL: url

Token considerations and recommended settings

- Since Azure OpenAI on your data includes search results on your index in the prompt, it's important to understand how that impacts your token allotment.

- Each call to the model includes tokens for the system message, the user prompt, conversation history,retrieved search documents, internal prompts, and the model's response.

Using the API



```
{
    "dataSources": [
        {
            "type": "AzureCognitiveSearch",
            "parameters": {
                "endpoint": "<your_search_endpoint>",
                "key": "<your_search_endpoint>",
                "indexName": "<your_search_index>"
            }
        }
    ],
    "messages":[
        {
            "role": "system",
            "content": "You are a helpful assistant assisting users with travel recommendations."
        },
        {
            "role": "user",
            "content": "I want to go to New York. Where should I stay?"
        }
    ]
}
```

Using the API

The call when using your own data needs to be sent to a different endpoint than is used when calling a base model, which includes extensions. Your call will be sent to a URL similar to the following.

<your_azure_openai_resource>/openai/deployments/<deployment_name>/extensions/chat/completions?api-version=2023-06-01-preview

Summary

- In this module, you learned how Azure OpenAI on your data allows developers to use supported AI chat models to reference specific sources of data.

- Connecting your own data allows the model to reference the specific information provided and its pretrained knowledge to provide more effective responses.

7. Responsible Generative AI