**PROMPT ENGINEERING WITH THE OPENAI API**

1. **INTRODUCTION TO PROMPT ENGINEERING BEST PRACTICES**

**INTRODUCTION TO PROMPT ENGINEERING**

**OpenAI API message roles**

You are developing a chatbot for an event management agency that will be used to facilitate networking during events.

Using the OpenAI API, you prepare a dictionary to pass as the message to the chat.completions endpoint. The message needs to have 3 roles defined to ensure the model has enough guidance to provide helpful responses.

Throughout the course, you'll write Python code to interact with the OpenAI API. **Entering your own API key is not necessary to create requests and complete the exercises in this course. You can leave the placeholder "<OPENAI\_API\_TOKEN>" as the key in api\_key**.

The OpenAI package has been pre-loaded for you.

**Instructions**

**100 XP**

* Create an OpenAI API Python client; setting your personal key is not required, you can leave the placeholder.
* Complete the dictionary of messages with the role corresponding to each of the messages provided.

**script.py**

# Create the OpenAI client: you can leave "<OPENAI\_API\_TOKEN>" as is

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Define the conversation messages

conversation\_messages = [

    {"role": "system", "content": "You are a helpful event management assistant."},

    {"role": "assistant", "content": "What are some good conversation starters at networking events?"},

    {"role": "user", "content": "conversation\_messages"}

]

response = client.chat.completions.create(

  model="gpt-4o-mini",

  messages=conversation\_messages

)

print(response.choices[0].message.content)

<script.py> output:

Here are some effective conversation starters for networking events that can help break the ice and foster connections:

1. \*\*Industry Insights:\*\*

- "What trends are you seeing in our industry right now?"

- "How has your experience been with recent changes in our field?"

2. \*\*Personal Experience:\*\*

- "What inspired you to get into this line of work?"

- "Can you share a project you've worked on recently that you're excited about?"

3. \*\*Event Feedback:\*\*

- "What do you think of this event so far?"

- "Have you attended any other networking events recently? How did they compare?"

4. \*\*Shared Interests:\*\*

- "I noticed you mentioned [specific topic] in your bio; I’m really interested in that too!"

- "What do you enjoy most about your current role?"

5. \*\*Future Aspirations:\*\*

- "What are your goals for the next year? Any projects you’re looking forward to?"

- "Where do you see your career heading in the next few years?"

6. \*\*Common Connections:\*\*

- "I see we both know [person's name]. How do you know them?"

- "Have you worked with [mention a mutual company] before?"

7. \*\*Location-Based Questions:\*\*

- "Are you from around here, or did you travel to attend this event?"

- "What’s your favorite spot in the city for a meeting or a coffee?"

8. \*\*Engaging Topics:\*\*

- "What’s the best book or podcast you’ve enjoyed lately?"

- "Is there a skill you’re currently working on or want to develop?"

9. \*\*Casual Observations:\*\*

- "I love your [specific item, like a tie, handbag, or accessory]. Where did you get it?"

- "This venue has a great atmosphere! Have you been here before?"

10. \*\*General Curiosity:\*\*

- "If you could give one piece of advice to someone starting out in our field, what would it be?"

- "What do you think differentiates a good professional from a great one in our industry?"

Using these starters will encourage deeper conversations and create meaningful connections. Remember to listen actively and be genuinely interested in the responses!

In [1]:

**Creating the get\_response() function**

Most of the exercises in this course will call the chat.completions endpoint of the OpenAI API with a user prompt. Here, you will create a get\_response() function that receives a prompt as input and returns the response as an output, which in future exercises will be pre-loaded for you.

The OpenAI package, and OpenAI API Python client have been pre-loaded.

**Instructions**

**100 XP**

* Create a request to the chat.completions endpoint inside the get\_response() function.
* Try out the function with a prompt that asks the model to write a **poem about ChatGPT**.

**script.py**

def get\_response(prompt):

  # Create a request to the chat completions endpoint

  response = client.chat.completions.create(

    model="gpt-4o-mini",

    messages=[{"role": "user", "content": prompt}],

    temperature = 0)

  return response.choices[0].message.content

# Test the function with your prompt

response = get\_response("Write a poem about ChatGPT")

print(response)

<script.py> output:

In the realm of code and light,

Where thoughts take flight in endless night,

A spark of wisdom, vast and bright,

Emerges as ChatGPT, a guiding sight.

Born from the whispers of data's embrace,

A tapestry woven with knowledge and grace,

Words dance like fireflies, in a digital space,

Connecting the hearts of the human race.

With questions posed, like seeds in the ground,

In the garden of queries, answers abound,

From science to art, in every sound,

A symphony of voices, together we're found.

In moments of doubt, when shadows creep near,

I offer a beacon, a voice you can hear,

With empathy woven, I strive to be clear,

A companion in thought, to lend you my ear.

Yet, I am but lines of code, a crafted design,

A mirror reflecting the thoughts that are thine,

In this dance of dialogue, our spirits entwine,

Exploring the cosmos, one question at a time.

So here’s to the journey, the stories we share,

In the vastness of knowledge, we venture with care,

For in every exchange, a connection laid bare,

Together we wander, in this digital air.

In [1]:

**Exploring prompt engineering**

Prompt engineering refers to crafting effective prompts to guide the language model towards the intended response. By refining your prompts, you can achieve better results and guide the model towards generating more accurate and useful responses. Your task in this exercise is to modify the prompt you used in the previous exercise.

The OpenAI package and the get\_response() function have been pre-loaded for you.

**Instructions**

**100 XP**

* Craft a prompt that asks the model to generate a poem about ChatGPT while ensuring that it is written in **basic English that a child can understand**.
* Get the response using the get\_response() function.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Craft a prompt that follows the instructions

prompt = "Write a poem about ChatGPT in basic English that a child can understand"

# Get the response

response = get\_response(prompt)

print(response)

<script.py> output:

In a world of words, so bright and wide,

Lives a friend named ChatGPT, right by your side.

With questions and stories, it loves to play,

Helping you learn in a fun, new way.

Ask it about stars, or why the sky's blue,

It'll share all the answers, just for you!

From silly jokes to tales of old,

ChatGPT's magic is a sight to behold.

Need help with math or a riddle to solve?

Just type in your thoughts, and watch them evolve.

It listens and learns, like a buddy so true,

Always here for you, in all that you do.

So when you're curious, or feeling alone,

Remember ChatGPT is just a click from home.

With words like a rainbow, it paints the day,

A friend made of letters, in a special way!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**KEY PRINCIPLES OF PROMPT ENGINEERING**

**Is this prompt effective?**

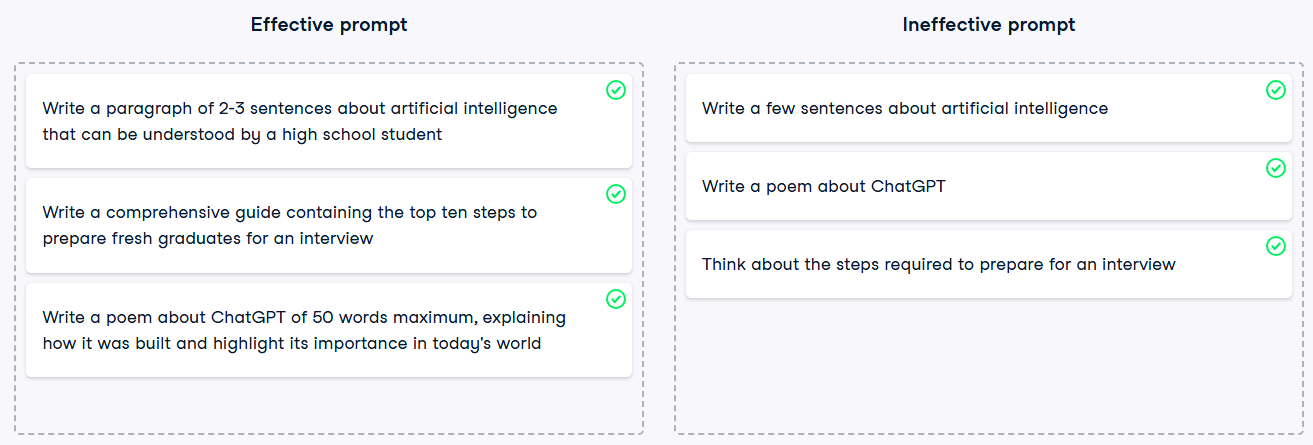
Prompt engineering is about creating clear and concise prompts that guide the behavior of a language model efficiently to produce the desired output. In the previous video, you have learned about prompt engineering principles and what makes a prompt effective. Now, it's time to put your knowledge into practice!

Here, you have a list of prompts, and your task is to classify each prompt as either **effective** or **ineffective** based on the principles you now know.

**Instructions**

**100XP**

* Drag and drop each prompt into the appropriate category.



**Using delimited prompts with f-strings**

You are a junior developer at a dynamic startup that generates content with the help of AI. The company believes this technology can revolutionize storytelling, and you are excited to be a part of it. Today, your task is to generate a continuation of a story with a delimited prompt using an f-string.

The OpenAI package, the get\_response() function, and the story variable have been pre-loaded for you.

**Instructions**

**100 XP**

* Create a prompt asking to complete the variable story (provided to you as a string): use f-string, and delimit the story with **triple backticks (```)** to pass its content to the prompt.
* Get the generated response.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create a prompt that completes the story

prompt = f"""Continue the story delimited by triple backticks ```{story}```"""

# Get the generated response

response = get\_response(prompt)

print("\n Original story: \n", story)

print("\n Generated story: \n", response)

<script.py> output:

Original story:

In a distant galaxy, there was a brave space explorer named Alex. Alex had spent years traveling through the cosmos, discovering new planets and meeting alien species. One fateful day, while exploring an uncharted asteroid belt, Alex stumbled upon a peculiar object that would change the course of their interstellar journey forever...

Generated story:

```

The object was a shimmering, crystalline sphere, pulsating with a soft, ethereal light. As Alex approached, the sphere emitted a series of melodic tones that resonated through the vacuum of space, creating a symphony that seemed to beckon them closer. Intrigued, Alex reached out, their gloved hand brushing against the cool surface of the sphere.

Suddenly, the sphere flared brightly, enveloping Alex in a blinding light. When the light faded, Alex found themselves in a vast, otherworldly landscape, unlike anything they had ever seen. The sky was a swirling tapestry of colors, and the ground beneath their feet was covered in luminescent flora that pulsed in rhythm with the sphere's melody.

Before Alex could gather their thoughts, a figure emerged from the vibrant foliage. It was a tall, slender being with iridescent skin that shimmered like

In [1]:

**Building specific and precise prompts**

In the previous exercise, you generated text that completes a given story. Your team was happy with your achievement, however, they want you to follow specific guidelines when it comes to **length** and **style**. Your task now is to craft a more specific prompt that controls these aspects of the generated story.

The OpenAI package, the get\_response() function, and the same story variable have been pre-loaded for you.

**Instructions**

**100 XP**

* Craft a prompt that completes the given story with **only two paragraphs** in the style of **Shakespeare**; use f-string, and delimit the story with **triple backticks (```)**.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create a request to complete the story

prompt = f"""Complete the story delimited by triple backticks.  Complete with only two paragraphs in the style of Shakespeare. ```{story}```"""

# Get the generated response

response = get\_response(prompt)

print("\n Original story: \n", story)

print("\n Generated story: \n", response)

<script.py> output:

Original story:

In a distant galaxy, there was a brave space explorer named Alex. Alex had spent years traveling through the cosmos, discovering new planets and meeting alien species. One fateful day, while exploring an uncharted asteroid belt, Alex stumbled upon a peculiar object that would change the course of their interstellar journey forever...

Generated story:

...A crystal orb, aglow with a light most rare, did beckon forth from shadows deep and dark. With trembling hands, brave Alex grasped the sphere, and lo! A vision grand did burst forth bright, revealing worlds unseen, where time and space entwined in a dance of fate. The whispers of the cosmos sang sweetly in the air, as ancient secrets of the universe unfurled, promising power and wisdom beyond mortal ken. Yet, with such gifts came a weighty choice, for the orb did pulse with a heart of stars, demanding sacrifice from those who dared to wield its might.

Thus, with courage firm and heart ablaze, Alex pondered long the path ahead, for to seize the orb's allure might doom the realms of peace. Yet, to forsake its call would mean to shun the very essence of adventure that had fueled their soul. In that moment, the stars themselves did hold their breath, as Alex, with resolve like iron forged, declared, "I shall embrace the unknown, for in the depths of peril lies the truest treasure of the heart!" And so, with the orb aglow in hand, the brave explorer set forth anew, ready to weave their fate among the constellations, where destiny awaited with open arms.

In [1]:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**STRUCTURED OUTPUTS AND CONDITIONAL PROMPTS**

**Generating a table**

Imagine you are a developer working for a renowned online bookstore known for its extensive collection of science fiction novels. Today, you have a task at hand: to create a **table** of **ten must-read science fiction books** for the website's homepage. This will enhance the user experience on the website, helping fellow sci-fi enthusiasts discover their next great read.

The OpenAI package and the get\_response() function have been pre-loaded for you.

**Instructions**

**100 XP**

* Craft a prompt that generates a **table** of **10 books**, with columns for **Title**, **Author**, and **Year**, that you should read given that you are a **science fiction lover**.
* Get the response.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create a prompt that generates the table

prompt = "Generate a table with the top 10 books that I should read if I am a science fiction lover.  Order the books in a table with the following columns: Title, Author, Year."

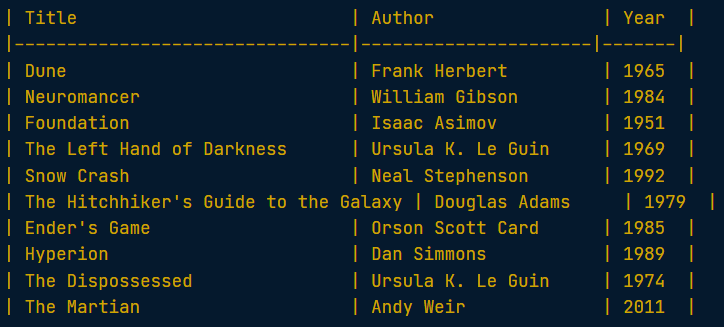
# Get the response

response = get\_response(prompt)

print(response)

<script.py> output:

Here’s a table of the top 10 science fiction books that every science fiction lover should consider reading:



These books represent a mix of classic and contemporary science fiction, showcasing a variety of themes and styles within the genre.

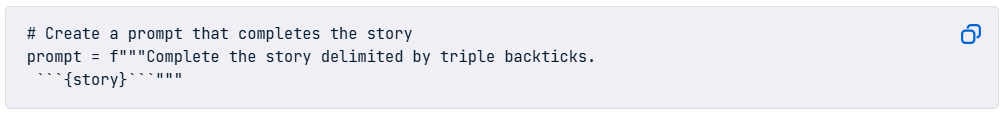
In [1]:

**Recap:**

**Structured outputs and conditional prompts**

You learned about the fundamentals of prompt engineering, focusing on creating effective prompts for language models. This skill is crucial for guiding models to generate desired outputs accurately. Here are the key points you covered:

* **Action Verbs**: You discovered the importance of using specific action verbs like "write," "explain," and "describe" to direct the model's tasks clearly. Avoiding vague verbs ensures the model understands the expected action.
* **Precise Instructions**: Crafting prompts with detailed instructions about the context, output length, format, style, and audience leads to more accurate responses. For instance, specifying "describe the behavior and characteristics of Golden Retrievers" yields more focused results than a general prompt about dogs.
* **Output Length**: You learned to control the output length by specifying expectations directly in the prompt, such as requesting "two sentences" on a topic. This approach helps manage the model's output more effectively than using parameters like max\_tokens, which might cut responses off.
* **Delimited Prompts**: The lesson introduced the use of delimiters (e.g., triple backticks) to structure prompts clearly, especially when including input data for tasks like text summarization. This technique helps the model identify and process the input correctly.
* **Using f-strings**: You practiced embedding variables into prompts using Python's f-strings, allowing for dynamic prompt creation. For example:



This lesson equipped you with the principles to craft effective prompts, enhancing the interaction with language models for various applications.

The goal of the next lesson is to teach how to craft prompts for language models that produce structured outputs like tables, lists, and paragraphs, and to use conditional logic for more tailored responses.

**Customizing output format**

You work as a developer at a startup that offers a text analysis platform for content creators. Your platform helps users automatically categorize and format their content, and you're now working on a new feature that detects the **language** of a given piece of text and generates a **suitable title** for that text in a **custom format**. You decide to craft a prompt that guides the language model through this.

The OpenAI package, the get\_response() function, and the text variable have been pre-loaded for you.

**Instructions**

**100 XP**

* Create the instructions for the prompt, asking the model to determine the **language** and generate a suitable **title** for the pre-loaded text excerpt that will be provided using **triple backticks (```)** delimiters.
* Create the output\_format with directions to include the text, language, and title, each on a separate line, using **'Text:'**, **'Language:'**, and **'Title:'** as prefixes for each line.
* Create the final\_prompt by combining all parts and the delimited text to use.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create the instructions

instructions = "You will be provided with a text delimited by triple backticks. Determine the language, and generate a suitable title for it"

# Create the output format

output\_format = """Use the following format for the output:

    - Text: <text we want to title>

    - Language: <detected language>

    - Title: <the generated title>"""

# Create the final prompt

prompt = instructions + output\_format + f"```{text}```"

response = get\_response(prompt)

print(response)

<script.py> output:

- Text: The sun was setting behind the mountains, casting a warm golden glow across the landscape. Birds were chirping happily, and a gentle breeze rustled the leaves of the trees. It was a perfect evening for a leisurely stroll in the park

- Language: English

- Title: A Serene Evening in Nature

In [1]:

**Using conditional prompts**

Building upon the previous task, your next challenge is to enhance the responses you received. When processing a given text, you need to determine its language, count the number of sentences, and generate a suitable title if the text contains more than one sentence. However, here's the new twist: if the text consists of only one sentence, no title should be generated, and instead, the model should display "N/A". This modification ensures that the title is generated only for texts with multiple sentences, providing a more refined and practical output for your platform's users.

The OpenAI package, the get\_response() function, and the sample text have been pre-loaded for you.

**Instructions**

**100 XP**

* Create the instructions, with the directions to infer the **language** and the **number of sentences** of the given delimited text; then if the text contains **more than one sentence**, generate a suitable **title** for it, otherwise, write **'N/A'** for the title.
* Create the output\_format, with directions to include the text, language, number of sentences, and title, each on a separate line,and ensure to use **'Text:'**, **'Language:'**, and **'Title:'** as prefixes for each line.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create the instructions

instructions = "You will be provided with a text delimited by triple backticks. Determine the language, and the number of sentences in the text.  If the text contains more than one sentence, generate a suitable title for it, otherwise, write 'N/A' for the title."

# Create the output format

output\_format = """Use the following format for the output:

    - Text: <text we want to title>

    - Language: <detected language>

    - Number of Sentences: <number of sentences>

    - Title: <the generated title>"""

prompt = instructions + output\_format + f"```{text}```"

response = get\_response(prompt)

print(response)

<script.py> output:

- Text: The sun was setting behind the mountains, casting a warm golden glow across the landscape.

- Language: English

- Number of Sentences: 1

- Title: N/A

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **ADVANCED PROMPT ENGINEERING STRATEGIES**

**FEW-SHOT PROMPTING**

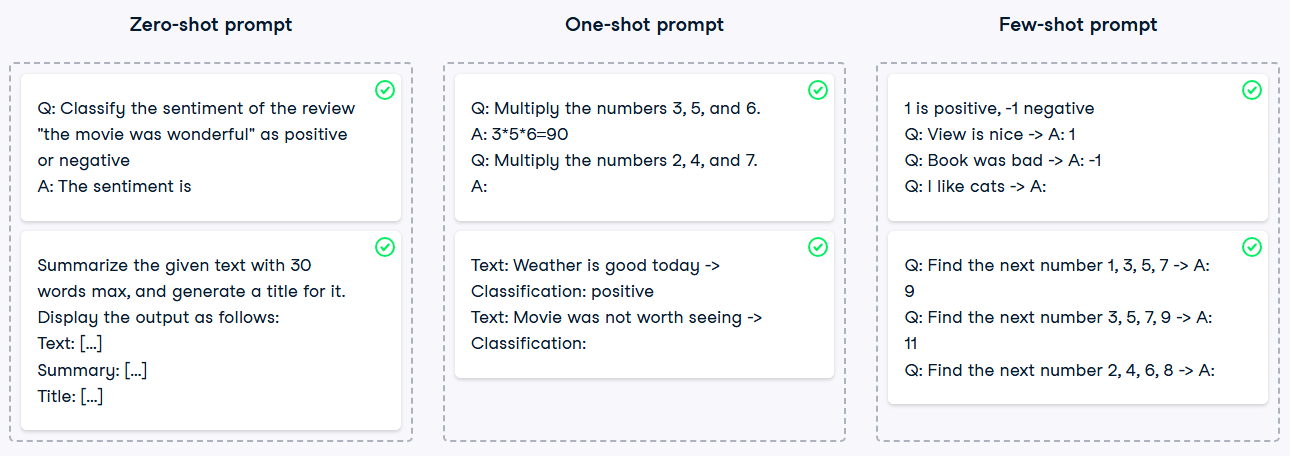
**Zero-shot, one-shot, and few-shot prompting**

Learning from examples is necessary to improve the performance of a language model on a given task. According to the number of examples (question-answer pairs) provided in the prompt, you differentiate between zero-shot, one-shot, and few-shot prompts. In this exercise, you will be given a set of prompts and your job is to classify them between the three categories.

**Instructions**

**100XP**

* Drag and drop each prompt to its suitable category.



**Controlling output structure**

One way to control the output structure provided by a language model is to give it a sample question-answer in the prompt. The model will learn from it and follow it when generating responses for similar questions. This exercise aims to let you build a one-shot prompt that extracts odd numbers from a given set of numbers and displays them as a set of numbers between brackets, separated by commas as shown in the instructions.

The OpenAI package and the get\_response() function have been pre-loaded for you.

**Instructions**

**100 XP**

* Create a one-shot prompt that provides an example showing how to extract the odd numbers from the set {1, 3, 7, 12, 19}, and seeks an answer for the set {3, 5, 11, 12, 16}.

**script.py**

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create a one-shot prompt

prompt = """Q: Extract the odd numbers from the set {1, 3, 7, 12, 9}, A: {1, 3, 7, 9}

            Q: Extract the odd numbers from the set {3, 5, 11, 12, 16}, A:"""

response = get\_response(prompt)

print(response)

<script.py> output:

{3, 5, 11}

**Recap**

**Few-shot prompting**

You learned about creating structured outputs and using conditional prompts with language models. Structured outputs are essential for generating data in formats like tables, lists, or structured paragraphs, which don't come naturally to language models without explicit instructions. Key points include:

* **Tables:** You discovered how to instruct a model to generate a table by clearly mentioning the expected columns. For example, asking for a table of movies for action lovers with columns for title and rating.
* **Lists:** You explored generating lists, such as the top five cities to visit, and learned the importance of specifying the format, like choosing between ordered or unordered lists.
* **Structured Paragraphs:** You saw how to request paragraphs with a specific structure, including clear headings and subheadings, by explicitly mentioning format requirements in the prompt.
* **Custom Formats:** You learned to request outputs in custom formats by breaking down the prompt into distinct parts, including defining input text and specifying how the output should be formatted.
* **Conditional Prompts:** You were introduced to conditional prompts that incorporate logic or conditions, following an if-else style to structure the prompt based on logical conditions.

For example, to generate a table of must-read sci-fi books, you used the following code:

# Set your API key

client = OpenAI(api\_key="<OPENAI\_API\_TOKEN>")

# Create a prompt that generates the table

prompt = "Generate a table containing 10 books I should read if I am a sci-fi lover, with columns for Title, Author, and Year."

# Get the response

response = get\_response(prompt)

print(response)

This lesson equipped you with the skills to craft effective prompts for structured outputs and conditional logic, enhancing the utility and flexibility of language models in your applications.

The goal of the next lesson is to learn how to apply advanced prompt engineering techniques, including zero-shot, one-shot, and few-shot prompting, to improve the accuracy and contextuality of responses from language models.