Guidelines for Prompting

In this lesson, you'll practice two prompting principles and their related tactics in order to write effective prompts for large language models.

Setup

Load the API key and relevant Python libaries.

In [ ]:
```
print("hello")
```

In [ ]:
```
!python -m ensurepip --upgrade # install pip on jupyter lab
```

In [ ]:
```
pip install gradio # install graio for UI
```

In [1]:
```
import openai
import os
import gradio as gr
from dotenv import load_dotenv, find_dotenv

# Load environment variables
_ = load_dotenv(find_dotenv())
openai.api_key = os.getenv('OPENAI_API_KEY')

# Function to generate lyrics based on a description
def generate_lyrics(description):
    # Create a prompt for lyrics generation
    prompt = f"Write a song lyrics based on the following scene or description:\n{d

    # Create a chat completion
    chat_completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=300,  # Adjust as needed for response length
        temperature=0.7  # Adjust for creativity vs. accuracy
    )

    # Return the generated lyrics
    return chat_completion.choices[0].message['content']

# Gradio Interface
demo_lyrics_generator = gr.Interface(
    fn=generate_lyrics,
    inputs=gr.Textbox(label="Song Description", placeholder="Describe the scene or
    outputs="text",
    title="Lyrics Generator",
    description="Enter a description to generate lyrics based on the scene or song
)
```

```
# Launch the interface with a public URL
demo_lyrics_generator.launch(share=True)
```

```
/usr/local/lib/python3.9/site-packages/tqdm/auto.py:21: TqdmWarning: IProgress not f
ound. Please update jupyter and ipywidgets. See https://ipywidgets.readthedocs.io/e
n/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
Running on local URL:  http://127.0.0.1:7860
Running on public URL: https://8d15b95de679405c14.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Out[1]:

In [2]:
```
import openai
import os
import gradio as gr
from dotenv import load_dotenv, find_dotenv

# Load environment variables
_ = load_dotenv(find_dotenv())
openai.api_key = os.getenv('OPENAI_API_KEY')

# Function to generate lyrics based on description, genre, emotion, and language
def generate_lyrics(description, genre, emotion, language):
```

```
    # Create a more structured prompt for musical synchronization, rhyming, and mel
    prompt = (
        f"Write song lyrics in {language} based on the following scene or descripti
        f"Make sure the lyrics follow a {genre} style, evoke the emotion of {emotio
        f"include a melodic rhythm, and have a rhyming structure.\n\n"
        "Lyrics:"
    )

    # Create a chat completion
    chat_completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=300,  # Adjust for response length
        temperature=0.8  # Higher creativity for lyrical flow
    )

    # Return the generated lyrics
    return chat_completion.choices[0].message['content']

# Gradio Interface with enhanced inputs
demo_lyrics_generator = gr.Interface(
    fn=generate_lyrics,
    inputs=[
        gr.Textbox(label="Song Description", placeholder="Describe the scene or the
        gr.Dropdown(choices=["Pop", "Rock", "Classical", "Hip-hop", "Jazz", "Countr
        gr.Textbox(label="Emotion", placeholder="Enter the emotion (e.g., love, sad
        gr.Dropdown(choices=["English", "Telugu", "Hindi", "Tamil", "Kannada", "Mal
    ],
    outputs="text",
    title="AI Lyrics Generator",
    description="Enter a description, genre, emotion, and language to generate musi
)

# Launch the interface with a public URL
demo_lyrics_generator.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7861
Running on public URL: https://e2023eb4eafe8e9ad8.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Out[2]:

In [4]:
```
import openai
import os
import gradio as gr
from dotenv import load_dotenv, find_dotenv

# Load environment variables
_ = load_dotenv(find_dotenv())
openai.api_key = os.getenv('OPENAI_API_KEY')

# Define a fixed context for the application
fixed_context = "You are a lyrics generator."

# Function to generate lyrics based on a description
def generate_lyrics(description):
    prompt = f"Description: {description}\nGenerate song lyrics:"

    chat_completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=150,
        temperature=0.7
    )
```

```
    return chat_completion.choices[0].message['content']

# Gradio Interface
demo_lyrics_generator = gr.Interface(
    fn=generate_lyrics,
    inputs=gr.Textbox(label="Song Description", placeholder="Describe the scene or
    outputs="text",
    title="AI Lyrics Generator",
    description="Enter a description to generate song lyrics based on the scene.",
    theme="compact"  # Keep this if your version supports it
)

# Launch the interface with a public URL
demo_lyrics_generator.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7862
Running on public URL: https://6d6c3874200787eaa0.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Out[4]:

In [5]:
```
import openai
import os
import gradio as gr
```

```
from dotenv import load_dotenv, find_dotenv

# Load environment variables
_ = load_dotenv(find_dotenv())
openai.api_key = os.getenv('OPENAI_API_KEY')

# Function to generate lyrics based on description, genre, emotion, and languages
def generate_lyrics(description, genre, emotion, languages, english_script):
    # Prepare the languages as a comma-separated string
    languages_str = ", ".join(languages)

    # Create a prompt for structured lyrics generation with multiple languages and
    prompt = (
        f"Write song lyrics in {languages_str} based on the following scene or desc
        f"Make sure the lyrics follow a {genre} style, evoke the emotion of {emotio
        f"include a melodic rhythm, and have a rhyming structure.\n"
    )

    # Add a note to generate lyrics in the English script if required
    if english_script:
        prompt += (
            f"Please provide the text in {languages_str} but written using the Engl
            f"maintain the original {languages_str} meaning while using English let
        )

    prompt += "Lyrics:"

    # Create a chat completion
    chat_completion = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}],
        max_tokens=300,  # Adjust for response length
        temperature=0.8  # Higher creativity for lyrical flow
    )

    # Return the generated lyrics
    return chat_completion.choices[0].message['content']

# Gradio Interface with enhanced inputs
demo_lyrics_generator = gr.Interface(
    fn=generate_lyrics,
    inputs=[
        gr.Textbox(label="Song Description", placeholder="Describe the scene or the
        gr.Dropdown(choices=["Pop", "Rock", "Classical", "Hip-hop", "Jazz", "Countr
        gr.Textbox(label="Emotion", placeholder="Enter the emotion (e.g., love, sad
        gr.CheckboxGroup(choices=["English", "Telugu", "Hindi", "Tamil", "Kannada",
                         label="Languages", value=["English"]),
        gr.Checkbox(label="Provide in English Script", value=False)
    ],
    outputs="text",
    title="AI Lyrics Generator",
    description="Enter a description, genre, emotion, and select one or more langua
)

# Launch the interface with a public URL
demo_lyrics_generator.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7863
Running on public URL: https://ab1d63dce3eca809cd.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Out[5]:

In [7]:
```python
import openai
import os
import gradio as gr
from dotenv import load_dotenv, find_dotenv

# Load environment variables
_ = load_dotenv(find_dotenv())
openai.api_key = os.getenv('OPENAI_API_KEY')

# Function to generate lyrics based on description, genre, emotion, and languages
def generate_lyrics(description, genre, emotion, languages, english_script):
    output_lyrics = ""

    for language in languages:
        # Create a prompt for each selected language
        prompt = (
            f"Write song lyrics in {language} based on the following scene or descr
            f"Make sure the lyrics follow a {genre} style, evoke the emotion of {em
```

```
            f"include a melodic rhythm, and have a rhyming structure.\n\n"
            "Lyrics:"
        )

        # Create a chat completion for each language
        chat_completion = openai.ChatCompletion.create(
            model="gpt-3.5-turbo",
            messages=[{"role": "user", "content": prompt}],
            max_tokens=3000,  # Adjust for response length
            temperature=0.8  # Higher creativity for lyrical flow
        )

        # Get the generated lyrics
        lyrics = chat_completion.choices[0].message['content']

        # Append the language and its generated lyrics to the output
        output_lyrics += f"Language: {language}\n{lyrics}\n\n"

        # If the English script is requested and the language is not English
        if english_script and language != "English":
            english_script_prompt = (
                f"Provide the following lyrics in {language} but written using the
                "The text should maintain the original meaning while using English
                f"{lyrics}"
            )

            # Get the English transliteration
            english_script_completion = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=[{"role": "user", "content": english_script_prompt}],
                max_tokens=300,
                temperature=0.7
            )

            # Get the English script version and append it
            english_script_lyrics = english_script_completion.choices[0].message['c
            output_lyrics += f"English Script for {language}:\n{english_script_lyri

    return output_lyrics

# Gradio Interface with enhanced inputs
demo_lyrics_generator = gr.Interface(
    fn=generate_lyrics,
    inputs=[
        gr.Textbox(label="Song Description", placeholder="Describe the scene or the
        gr.Dropdown(choices=["Pop", "Rock", "Classical", "Hip-hop", "Jazz", "Countr
        gr.Textbox(label="Emotion", placeholder="Enter the emotion (e.g., love, sad
        gr.CheckboxGroup(choices=["English", "Telugu", "Hindi", "Tamil", "Kannada",
                         label="Languages", value=["English"]),
        gr.Checkbox(label="Provide in English Script", value=False)
    ],
    outputs="text",
    title="AI Lyrics Generator",
    description="Enter a description, genre, emotion, and select one or more langua
)
```

```
# Launch the interface with a public URL
demo_lyrics_generator.launch(share=True)
```

```
Running on local URL:  http://127.0.0.1:7865
Running on public URL: https://fc9a83d4b9e74cbe86.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, ru
n `gradio deploy` from Terminal to deploy to Spaces (https://huggingface.co/spaces)
```

Out[7]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In this course, we've provided some code that loads the OpenAI API key for you.

In [ ]:
```
import openai
import os
```

```
from dotenv import load_dotenv, find_dotenv
_ = load_dotenv(find_dotenv())

openai.api_key  = os.getenv('OPENAI_API_KEY')
```

helper function

Throughout this course, we will use OpenAI's `gpt-3.5-turbo` model and the chat completions endpoint.

This helper function will make it easier to use prompts and look at the generated outputs. **Note**: In June 2023, OpenAI updated gpt-3.5-turbo. The results you see in the notebook may be slightly different than those in the video. Some of the prompts have also been slightly modified to product the desired results.

In [ ]:
```
def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0, # this is the degree of randomness of the model's output
    )
    return response.choices[0].message["content"]
```

Note: This and all other lab notebooks of this course use OpenAI library version `0.27.0`.

In order to use the OpenAI library version `1.0.0`, here is the code that you would use instead for the `get_completion` function:

```
client = openai.OpenAI()

def get_completion(prompt, model="gpt-3.5-turbo"):
    messages = [{"role": "user", "content": prompt}]
    response = client.chat.completions.create(
        model=model,
        messages=messages,
        temperature=0
    )
    return response.choices[0].message.content
```

Prompting Principles

- **Principle 1: Write clear and specific instructions**
- **Principle 2: Give the model time to "think"**

Tactics

Tactic 1: Use delimiters to clearly indicate distinct parts of the input

- Delimiters can be anything like: ```, """, < >, `<tag> </tag>`, `:`

In [ ]:
```
text = f"""
You should express what you want a model to do by \
providing instructions that are as clear and \
specific as you can possibly make them. \
This will guide the model towards the desired output, \
and reduce the chances of receiving irrelevant \
or incorrect responses. Don't confuse writing a \
clear prompt with writing a short prompt. \
In many cases, longer prompts provide more clarity \
and context for the model, which can lead to \
more detailed and relevant outputs.
"""
prompt = f"""
Summarize the text delimited by triple backticks \
into a single sentence.
```{text}```
"""
response = get_completion(prompt)
print(response)
```

Tactic 2: Ask for a structured output

- JSON, HTML

In [ ]:
```
prompt = f"""
Generate a list of three made-up book titles along \
with their authors and genres.
Provide them in JSON format with the following keys:
book_id, title, author, genre.
"""
response = get_completion(prompt)
print(response)
```

Tactic 3: Ask the model to check whether conditions are satisfied

In [ ]:
```
text_1 = f"""
Making a cup of tea is easy! First, you need to get some \
water boiling. While that's happening, \
grab a cup and put a tea bag in it. Once the water is \
hot enough, just pour it over the tea bag. \
Let it sit for a bit so the tea can steep. After a \
few minutes, take out the tea bag. If you \
like, you can add some sugar or milk to taste. \
And that's it! You've got yourself a delicious \
cup of tea to enjoy.
"""
prompt = f"""
You will be provided with text delimited by triple quotes.
If it contains a sequence of instructions, \
re-write those instructions in the following format:
```
```

```
Step 1 - ...
Step 2 - …
…
Step N - …

If the text does not contain a sequence of instructions, \
then simply write \"No steps provided.\"

\"\"\"{text_1}\"\"\"
"""
response = get_completion(prompt)
print("Completion for Text 1:")
print(response)
```

In [ ]:
```
text_2 = f"""
The sun is shining brightly today, and the birds are \
singing. It's a beautiful day to go for a \
walk in the park. The flowers are blooming, and the \
trees are swaying gently in the breeze. People \
are out and about, enjoying the lovely weather. \
Some are having picnics, while others are playing \
games or simply relaxing on the grass. It's a \
perfect day to spend time outdoors and appreciate the \
beauty of nature.
"""
prompt = f"""
You will be provided with text delimited by triple quotes.
If it contains a sequence of instructions, \
re-write those instructions in the following format:

Step 1 - ...
Step 2 - …
…
Step N - …

If the text does not contain a sequence of instructions, \
then simply write \"No steps provided.\"

\"\"\"{text_2}\"\"\"
"""
response = get_completion(prompt)
print("Completion for Text 2:")
print(response)
```

Tactic 4: "Few-shot" prompting

In [ ]:
```
prompt = f"""
Your task is to answer in a consistent style.

<child>: Teach me about patience.

<grandparent>: The river that carves the deepest \
valley flows from a modest spring; the \
grandest symphony originates from a single note; \
the most intricate tapestry begins with a solitary thread.
```

```
<child>: Teach me about resilience.
"""
response = get_completion(prompt)
print(response)
```

Principle 2: Give the model time to "think"

Tactic 1: Specify the steps required to complete a task

In [ ]:
```
text = f"""
In a charming village, siblings Jack and Jill set out on \
a quest to fetch water from a hilltop \
well. As they climbed, singing joyfully, misfortune \
struck—Jack tripped on a stone and tumbled \
down the hill, with Jill following suit. \
Though slightly battered, the pair returned home to \
comforting embraces. Despite the mishap, \
their adventurous spirits remained undimmed, and they \
continued exploring with delight.
"""
# example 1
prompt_1 = f"""
Perform the following actions:
1 - Summarize the following text delimited by triple \
backticks with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the following \
keys: french_summary, num_names.

Separate your answers with line breaks.

Text:
```{text}```
"""
response = get_completion(prompt_1)
print("Completion for prompt 1:")
print(response)
```

Ask for output in a specified format

In [ ]:
```
prompt_2 = f"""
Your task is to perform the following actions:
1 - Summarize the following text delimited by
  <> with 1 sentence.
2 - Translate the summary into French.
3 - List each name in the French summary.
4 - Output a json object that contains the
  following keys: french_summary, num_names.

Use the following format:
Text: <text to summarize>
Summary: <summary>
```
```

```
Translation: <summary translation>
Names: <list of names in summary>
Output JSON: <json with summary and num_names>

Text: <{text}>
"""
response = get_completion(prompt_2)
print("\nCompletion for prompt 2:")
print(response)
```

Tactic 2: Instruct the model to work out its own solution before rushing to a conclusion

In [ ]:
```
prompt = f"""
Determine if the student's solution is correct or not.

Question:
I'm building a solar power installation and I need \
 help working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost \
me a flat $100k per year, and an additional $10 / square \
foot
What is the total cost for the first year of operations
as a function of the number of square feet.

Student's Solution:
Let x be the size of the installation in square feet.
Costs:
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000
"""
response = get_completion(prompt)
print(response)
```

Note that the student's solution is actually not correct.

We can fix this by instructing the model to work out its own solution first.

In [ ]:
```
prompt = f"""
Your task is to determine if the student's solution \
is correct or not.
To solve the problem do the following:
- First, work out your own solution to the problem including the final total.
- Then compare your solution to the student's solution \
and evaluate if the student's solution is correct or not.
Don't decide if the student's solution is correct until
you have done the problem yourself.

Use the following format:
Question:
```

```
```
question here
```
Student's solution:
```
student's solution here
```
Actual solution:
```
steps to work out the solution and your solution here
```
Is the student's solution the same as actual solution \
just calculated:
```
yes or no
```
Student grade:
```
correct or incorrect
```


Question:
```
I'm building a solar power installation and I need help \
working out the financials.
- Land costs $100 / square foot
- I can buy solar panels for $250 / square foot
- I negotiated a contract for maintenance that will cost \
me a flat $100k per year, and an additional $10 / square \
foot
What is the total cost for the first year of operations \
as a function of the number of square feet.
```
Student's solution:
```
Let x be the size of the installation in square feet.
Costs:
1. Land cost: 100x
2. Solar panel cost: 250x
3. Maintenance cost: 100,000 + 100x
Total cost: 100x + 250x + 100,000 + 100x = 450x + 100,000
```
Actual solution:
"""
response = get_completion(prompt)
print(response)
```

Model Limitations: Hallucinations

- Boie is a real company, the product name is not real.

In [ ]:
```
prompt = f"""
Tell me about AeroGlide UltraSlim Smart Toothbrush by Boie
"""
```

```
response = get_completion(prompt)
print(response)
```

Try experimenting on your own!

In [ ]:

Notes on using the OpenAI API outside of this classroom

To install the OpenAI Python library:

```
!pip install openai
```

The library needs to be configured with your account's secret key, which is available on the website.

You can either set it as the `OPENAI_API_KEY` environment variable before using the library:

```
!export OPENAI_API_KEY='sk-...'
```

Or, set `openai.api_key` to its value:

```
import openai
openai.api_key = "sk-..."
```

A note about the backslash

- In the course, we are using a backslash `\` to make the text fit on the screen without inserting newline '\n' characters.
- GPT-3 isn't really affected whether you insert newline characters or not. But when working with LLMs in general, you may consider whether newline characters in your prompt may affect the model's performance.

In [ ]: