Learning Objectives

Learners will be able to...

- **Generate answers using OpenAI's ChatGPT API**
- **Learn to manage conversation length, token limits, and structure to prevent truncated or incomplete responses**
- **Understand and apply the temperature and max tokens parameters to control creativity and response length**
- **Compare translation to transformation**

info

Make Sure You Know

You are familiar with Python.

Limitations

This is a gentle introduction. So there is a little bit of Python programming. The information might not be the most up to date as OpenAI releases new features.

Refining System Messages for Effectiveness

System messages play a crucial role in guiding the AI assistant's behavior when interacting with the ChatGPT API. In this guide, we will explore the importance of refining system messages, techniques for crafting effective instructions, and examples to help you improve the quality of generated responses.

The system message sets the context and role for the AI assistant. It provides initial instructions on how the assistant should behave throughout the conversation. By carefully crafting system messages, you can ensure the AI understands the desired behavior, leading to more accurate and relevant responses.

> Techniques for Crafting Effective System Messages:
>
> 1. **Be specific**: Clearly define the assistant's role or expertise to set proper expectations for the conversation.
> 2. **Set constraints**: If necessary, provide limitations or rules that the assistant should follow when generating responses.
> 3. **Offer context**: Mention any relevant background information that could influence the assistant's understanding or responses.

Lets try a few examples, First in order to interact with the API we need to get our libraries and our API key.

```
import os
import openai
import secret
openai.api_key=secret.api_key
```

Now we are ready to use our API call which we will set equal to `response` so its easier for us to manipulate.

```
response=openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a financial
        advisor."},
        {"role": "user", "content": "What are some tips for
        saving money?"},
        {"role": "assistant", "content": "Creating a budget,
        reducing expenses, and saving on utilities are some ways
        to save money."},
        {"role": "user", "content": "How do I create a budget?"}
    ]
)
```

We are going to use some slicing in order to make our response more legible.

```
print(response['choices'][0]['message']['content'])
```

In the above example we are setting the AI as financial advisor. It provides initial instructions on how the assistant should behave throughout the conversation.

However, we can use more precision, with our tasks. Lets replace our message with the following.

```
{"role": "system", "content": "You are a travel expert
specializing in European destinations."}
```

Explanation: This system message clearly defines the assistant's role as a travel expert and narrows down the expertise to European destinations, providing more focused guidance. Now if the user want to add something more specific, they can add it to the conversation.

More on techniques to refine messages

You can guide the AI assistant effectively and enhance the overall quality of the generated content. Refining system messages is essential for achieving more accurate and useful responses from the ChatGPT API. You can increase efficiency of result By being specific, setting constraints, and offering context.

In the travel example, we created a travel expert. Lets try setting some constraints and adding the context. Replace the previous message with the following in order to create constraint.
System message with constraints:

```
{"role": "system", "content": "You are a movie recommendation
        assistant. Only suggest movies rated PG-13 or lower."}
```

In addition to defining the assistant's role as a movie recommendation assistant, this system message sets a constraint to only suggest movies rated PG-13 or lower.

Let's try an example offering context:

```
{"role": "system", "content": "You are a history tutor. The
        student is currently studying the American Revolutionary
        War."}
```

This system message defines the assistant's role as a history tutor and provides context that the user is studying the American Revolutionary War, helping the AI to generate relevant responses.

Ambiguous

Ambiguous queries can lead to unclear or confusing AI-generated responses.

we will discuss strategies to recognize and handle ambiguous user queries, ensuring that the ChatGPT API generates clear and informative responses that accurately address user questions.

Strategies for Handling Ambiguous Queries:

Detect ambiguity: Monitor user queries for potential ambiguity, vagueness, or lack of context that might result in unclear responses.
Request clarification: If a user query is ambiguous, program the AI assistant to ask for clarification or additional information before providing a response.
Provide examples: When responding to an ambiguous query, offer a range of examples or possibilities to cover different interpretations of the user's question.
Refine system messages: Modify system messages to guide the AI in handling ambiguous queries more effectively.

Suppose you have the following ambiguous user query:

```
{"role": "user", "content": "What is the best way to cook it?"}
```

Handling Ambiguity:

```
{"role": "system", "content": "You are a helpful assistant."},
 {"role": "user", "content": "What is the best way to cook
        it?"},
{"role": "assistant", "content": "I need more information to
        provide a specific answer. What ingredient or dish are
        you referring to?"}
```

In this example, the AI assistant recognizes the ambiguity in the user's question and asks for clarification before attempting to provide a cooking method.

Managing Conversation Length and Token Limits

We will explore the importance of managing conversation length, understanding token limits, and techniques for optimizing conversations to ensure informative and better responses. Token limits and conversation length play a significant role in the quality and completeness of responses generated by the **ChatGPT API**.

Language models read text in chunks called tokens. In the ChatGPT as Language models, also uses tokens to represent text. In English, a token can be as short as one character or as long as one word (e.g., a or apple), and in some languages tokens can be even shorter than one character or even longer than one word. For example, the string `ChatGPT is great!` is encoded into six tokens: `["Chat", "G", "PT", " is", " great", "!"]`.

Both input and output tokens count toward the total tokens used in an API call. Each model has a maximum token limit (e.g., 4096 tokens for gpt-3.5-turbo). If a conversation exceeds this limit, you must truncate, omit, or shorten parts of the text to fit within the constraints. This can impact the coherence and completeness of the generated responses.

Techniques for Optimizing Conversations:

1. **Prioritize important information**: Keep the most relevant parts of the conversation while removing or shortening less essential content.
2. **Condense messages**: Use concise language and remove unnecessary words or phrases in user messages.
3. **Remove excessively long responses**: If a previous AI-generated response is too long and not crucial for the current query, consider removing or shortening it.
   Be aware of the token count: Monitor the total tokens used in a conversation to ensure that you remain within the model's token limit.

For example, suppose you have the following conversation that exceeds the token limit:

```
[{"role": "system", "content": "You are a helpful assistant."},
{"role": "user", "content": "Tell me about the history of the
        Eiffel Tower and its significance."},
{"role": "assistant", "content": "<A long, detailed response
        about the Eiffel Tower's history and significance>"},
{"role": "user", "content": "What materials were used to build
        the Eiffel Tower and what was the construction process
        like?"}]
```

To see how many tokens are used by an API call, check the usage field in the API response:

```
response['usage']['total_tokens']
```

Replace the message with this Optimized Conversation:

```
[  {"role": "system", "content": "You are a helpful
        assistant."},  {"role": "user", "content": "Eiffel Tower
        history and significance?"},
{"role": "assistant", "content": "<A shortened, concise response
        about the Eiffel Tower's history and significance>"},
{"role": "user", "content": "Materials used and construction
        process?"}]
```

In the optimized conversation, we shortened the user messages and the assistant's response to fit within the token limit while still providing useful information.

Controlling Creativity and Response Length with Temperature and Max Tokens

In ChatGPT API interactions, you can fine-tune the AI's output by adjusting parameters like **temperature** and **max tokens**. We will delve into the significance of these parameters, learn how they affect the generated responses, and explore techniques to control the creativity and length of AI outputs. By understanding, experimenting, and fine-tuning these parameters, you can effectively customize the ChatGPT API's responses to match your specific use case.

Lets refresh our definitions for Temperature and Max Tokens.

Temperature	Max tokens
This parameter influences the randomness and creativity of the AI's responses. A higher temperature (e.g., 0.8) results in more diverse and creative outputs, while a lower temperature (e.g., 0.2) leads to more focused and deterministic responses.	This parameter sets an arbitrary limit on the length of the AI-generated response. By defining max tokens, you can control the response length to ensure it fits within a desired range. However, setting max tokens too low might result in incomplete or nonsensical responses.

Lets try writing an example, token and max tokens are extra variable that we can add during our API call.

```
response=openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful
        assistant."},
        {"role": "user", "content": "What are some creative ways
        to reuse plastic bottles?"}
    ],
  temperature=0.6,
  max_tokens=50
)

print(response['choices'][0]['message']['content'])
```

In this example, the temperature is set to 0.6, striking a balance between creativity and coherence. The max tokens parameter is set to 50, limiting the response length without making it too short.

> Techniques for Adjusting Temperature and Max Tokens:
>
> 1. **Experiment with temperature values**: Test various temperature settings to find the best balance between creativity and coherence for your specific use case.
> 2. **Set appropriate max tokens**: Determine an ideal response length for your application, and set max tokens accordingly. Be cautious of setting it too low, as it may truncate the output and hinder comprehension.
> 3. **Iterate and fine-tune**: Continuously test and adjust these parameters based on generated responses to optimize AI outputs for your needs.

There is also a playground available on the main OpenAI website to help with visualizing the tools.

Additionally, we can include the `n` parameter to generate multiple responses given different temperatures and tokens.

```
response=openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "You are a helpful
        assistant."},
        {"role": "user", "content": "What's a good book to read
        on a rainy day?"}
    ],
  n=3,
  temperature=0.7
)
```

In this example, the API generates three alternative completions for the user query, providing a range of book suggestions. You can then choose the most fitting response based on your criteria or user preferences. Remember we can use the following code to better visualize.

```
for i in (response["choices"]):
  print(i["message"]['content'].strip())
  print(" ////////")
```

Coding Exercise

In this exercise, you will be working with the OpenAI API. Create a function `chatfunctions()` that takes a `prompt`, `temperature`, and `maximum number of tokens` as arguments. It should uses the OpenAI GPT-3.5-turbo model to generate a chat response, your task is to modify the function to make it more interactive.

1. Modify the `chatfunctions()` to accept a list of user messages instead of a single prompt. The function should be able to iterate over the list of user messages and generate a conversation accordingly. The function should return a list of responses generated by the OpenAI model.

2. Define another function named `user_experience()`. This function should accept a single prompt from the user, a temperature, and maximum number of tokens. The function should use the `chatfunctions()` and display a user-friendly conversation on the console, with clear distinctions between user inputs and AI responses.

You can use the following constraints:

- The list of user messages will contain at least 1 message and at most 10 messages.
- The temperature value will be a float in the range (0, 1].
- The maximum number of tokens will be an integer in the range [50, 1000].

Function Signatures:

Python:

```
def chatfunctions(prompts: List[str], temp: float, max_t: int) -
        > List[str]:
    pass

def user_experience(prompt: str, temp: float, max_t: int):
    pass
```

Examples:

```
# Example 1:
prompts = ["What is the capital of France?", "Who is Elon
        Musk?", "What is the weather like?"]
print(chatfunctions(prompts, 0.5, 100))
# The output should be a list of three responses generated by
        the AI model.


# Example 2:
user_experience("What is the capital of France?", 0.5, 100)
# The console should display a user-friendly conversation, for
        example:
# User: What is the capital of France?
# AI: The capital of France is Paris.


# Example 3:
prompts = ["What is AI?", "What are some applications of AI?"]
print(chatfunctions(prompts, 0.7, 150))
# The output should be a list of two responses generated by the
        AI model.


# Example 4:
user_experience("What is AI?", 0.7, 150)
# The console should display a user-friendly conversation, for
        example:
# User: What is AI?
# AI: AI, or Artificial Intelligence, refers to the simulation
        of human intelligence processes by machines, especially
        computer systems. This includes learning, reasoning,
        problem-solving, perception, and language understanding.
```

Notes:

In this exercise, you are working with the OpenAI API. Make sure to handle the API responses carefully. Consider all possible exceptions and errors. If if the API call fails, your function should be able to handle these situations gracefully.

Please remember, the OpenAI model responses may vary so the examples provided are indicative of a typical response. Actual results may differ.