Prompt Templates and Variables Tutorial (Using Jinja2)

Overview

This tutorial provides a comprehensive introduction to creating and using prompt templates with variables in the context of AI language models. It focuses on leveraging Python and the Jinja2 templating engine to create flexible, reusable prompt structures that can incorporate dynamic content. The tutorial demonstrates how to interact with OpenAI's GPT models using these advanced prompting techniques.

Motivation

As AI language models become increasingly sophisticated, the ability to craft effective prompts becomes crucial for obtaining desired outputs. Prompt templates and variables offer several advantages:

1. **Reusability**: Templates can be reused across different contexts, saving time and ensuring consistency.
2. **Flexibility**: Variables allow for dynamic content insertion, making prompts adaptable to various scenarios.
3. **Complexity Management**: Templates can handle complex structures, including conditional logic and loops, enabling more sophisticated interactions with AI models.
4. **Scalability**: As applications grow, well-structured templates make it easier to manage and maintain large numbers of prompts.

This tutorial aims to equip learners with the knowledge and skills to create powerful, flexible prompt templates, enhancing their ability to work effectively with AI language models.

Key Components

The tutorial covers several key components:

1. **PromptTemplate Class**: A custom class that wraps Jinja2's Template class, providing a simple interface for creating and using templates.
2. **Jinja2 Templating**: Utilization of Jinja2 for advanced templating features, including variable insertion, conditional statements, and loops.
3. **OpenAI API Integration**: Direct use of the OpenAI API for sending prompts and receiving responses from GPT models.

4. **Variable Handling**: Techniques for incorporating variables into templates and managing dynamic content.
5. **Conditional Logic**: Implementation of if-else statements within templates to create context-aware prompts.
6. **Advanced Formatting**: Methods for structuring complex prompts, including list formatting and multi-part instructions.

Method Details

The tutorial employs a step-by-step approach to introduce and demonstrate prompt templating concepts:

1. **Setup and Environment**: The lesson begins by setting up the necessary libraries, including Jinja2 and the OpenAI API client.

2. **Basic Template Creation**: Introduction to creating simple templates with single and multiple variables using the custom PromptTemplate class.

3. **Variable Insertion**: Demonstration of how to insert variables into templates using Jinja2's `{{ variable }}` syntax.

4. **Conditional Content**: Exploration of using if-else statements in templates to create prompts that adapt based on provided variables.

5. **List Processing**: Techniques for handling lists of items within templates, including iteration and formatting.

6. **Advanced Templating**: Demonstration of more complex template structures, including nested conditions, loops, and multi-part prompts.

7. **Dynamic Instruction Generation**: Creation of templates that can generate structured instructions based on multiple input variables.

8. **API Integration**: Throughout the tutorial, examples show how to use the templates with the OpenAI API to generate responses from GPT models.

The methods are presented with practical examples, progressing from simple to more complex use cases. Each concept is explained theoretically and then demonstrated with a practical application.

Conclusion

This tutorial provides a solid foundation in creating and using prompt templates with variables, leveraging the power of Jinja2 for advanced templating features. By the end of the lesson, learners will have gained:

1. Understanding of the importance and applications of prompt templates in AI

interactions.
2. Practical skills in creating reusable, flexible prompt templates.
3. Knowledge of how to incorporate variables and conditional logic into prompts.
4. Experience in structuring complex prompts for various use cases.
5. Insight into integrating templated prompts with the OpenAI API.

These skills enable more sophisticated and efficient interactions with AI language models, opening up possibilities for creating more advanced, context-aware AI applications. The techniques learned can be applied to a wide range of scenarios, from simple query systems to complex, multi-turn conversational agents.

Setup

In [5]:
```
import os
import openai
from jinja2 import Template
from dotenv import load_dotenv

load_dotenv()

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

def get_completion(prompt, model="gpt-4o-mini"):
    ''' Get a completion from the OpenAI API
    Args:
        prompt (str): The prompt to send to the API
        model (str): The model to use for the completion
    Returns:
        str: The completion text
    '''
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model=model,
        messages=messages,
        temperature=0,
    )
    return response.choices[0].message["content"]
```

1. Creating Reusable Prompt Templates

We'll create a PromptTemplate class that uses Jinja2 for templating:

In [7]:
```
class PromptTemplate:
    ''' A class to represent a template for generating prompts with vari
    Attributes:
        template (str): The template string with variables
        input_variables (list): A list of the variable names in the temp
    '''
    def __init__(self, template, input_variables):
        self.template = Template(template)
        self.input_variables = input_variables
```

```
    def format(self, **kwargs):
        return self.template.render(**kwargs)

# Simple template with one variable
simple_template = PromptTemplate(
    template="Provide a brief explanation of {{ topic }}.",
    input_variables=["topic"]
)

# More complex template with multiple variables
complex_template = PromptTemplate(
    template="Explain the concept of {{ concept }} in the field of {{ fi
    input_variables=["concept", "field", "audience"]
)

# Using the simple template
print("Simple Template Result:")
prompt = simple_template.format(topic="photosynthesis")
print(get_completion(prompt))

print("\n" + "-"*50 + "\n")

# Using the complex template
print("Complex Template Result:")
prompt = complex_template.format(
    concept="neural networks",
    field="artificial intelligence",
    audience="beginner"
)
print(get_completion(prompt))
```

```
Simple Template Result:
Photosynthesis is the process by which green plants, algae, and some bacte
ria convert light energy, usually from the sun, into chemical energy store
d in glucose. This process primarily occurs in the chloroplasts of plant c
ells, where chlorophyll, the green pigment, captures light energy.

During photosynthesis, carbon dioxide (CO₂) from the atmosphere and water
(H₂O) from the soil are used to produce glucose (C₆H₁₂O₆) and oxygen (O₂).
The overall chemical equation for photosynthesis can be summarized as:

6 CO₂ + 6 H₂O + light energy → C₆H₁₂O₆ + 6 O₂

Photosynthesis is crucial for life on Earth, as it provides the oxygen we
breathe and serves as the foundation of the food chain by producing organi
c compounds that serve as energy sources for other organisms.

--------------------------------------------------

Complex Template Result:
Neural networks are a key technology in artificial intelligence that mimic
the way the human brain works to process information. They consist of laye
rs of interconnected nodes, or "neurons," which work together to recognize
patterns and make decisions.

Here's a simple breakdown:

1. **Structure**: A neural network has an input layer (where data enters),
one or more hidden layers (where processing happens), and an output layer
(where results come out).

2. **Learning**: Neural networks learn from data by adjusting the connecti
ons (weights) between neurons based on the errors they make. This process
is called training.

3. **Function**: Once trained, neural networks can perform tasks like imag
e recognition, language translation, and even playing games by predicting
outcomes based on new input data.

In essence, neural networks are powerful tools that help computers learn f
rom experience, similar to how humans learn from their surroundings.
```

2. Using Variables for Dynamic Content

Now let's explore more advanced uses of variables, including conditional content:

In [9]:

```python
# Template with conditional content
conditional_template = PromptTemplate(
    template="My name is {{ name }} and I am {{ age }} years old. "
             "{% if profession %}I work as a {{ profession }}.{% else %
             "Can you give me career advice based on this information? 
    input_variables=["name", "age", "profession"]
)

# Using the conditional template
print("Conditional Template Result (with profession):")
prompt = conditional_template.format(
    name="Alex",
    age="28",
```

```
    profession="software developer"
)
print(get_completion(prompt))

print("\nConditional Template Result (without profession):")
prompt = conditional_template.format(
    name="Sam",
    age="22",
    profession=""
)
print(get_completion(prompt))

print("\n" + "-"*50 + "\n")
```

Conditional Template Result (with profession):
Sure, Alex! Here are some career tips for you as a software developer:

1. **Continuous Learning**: Stay updated with the latest technologies and programming languages. Consider online courses or certifications in areas like cloud computing, AI, or cybersecurity.

2. **Networking**: Attend industry meetups, conferences, and online forums to connect with other professionals. This can lead to job opportunities an d collaborations.

3. **Build a Portfolio**: Work on personal or open-source projects to show case your skills. A strong portfolio can set you apart in job application s.

4. **Soft Skills**: Develop communication and teamwork skills. Being able to collaborate effectively is crucial in software development.

5. **Explore Specializations**: Consider specializing in a niche area (e. g., mobile development, data science, or DevOps) to enhance your marketabi lity.

6. **Seek Feedback**: Regularly ask for feedback from peers and mentors to improve your coding and problem-solving skills.

7. **Work-Life Balance**: Prioritize your well-being to avoid burnout. A b alanced life can enhance your productivity and creativity.

Good luck with your career!

Conditional Template Result (without profession):
Sure, Sam! Here are some steps you can take:

1. **Self-Assessment**: Identify your skills, interests, and values. Consi der what you enjoy doing and what you're good at.

2. **Explore Options**: Research different career paths that align with yo ur interests. Look into industries that are growing and have job opportuni ties.

3. **Education & Training**: Consider further education or certifications that can enhance your skills. Online courses can be a flexible option.

4. **Networking**: Connect with professionals in your fields of interest t hrough LinkedIn, local meetups, or industry events. Informational intervie ws can provide valuable insights.

5. **Internships/Volunteering**: Gain experience through internships or vo lunteer work. This can help you build your resume and make connections.

6. **Job Search**: Start applying for entry-level positions or roles that interest you. Tailor your resume and cover letter for each application.

7. **Stay Positive**: Job searching can be challenging, but stay persisten t and open to opportunities.

Good luck!

In [10]:
```
# Template for list processing
list_template = PromptTemplate(
    template="Categorize these items into groups: {{ items }}. Provide t
    input_variables=["items"]
)

# Using the list template
print("List Template Result:")
prompt = list_template.format(
    items="apple, banana, carrot, hammer, screwdriver, pliers, novel, te
)
print(get_completion(prompt))
```

```
List Template Result:
Here are the categorized groups for the items you provided:

### Fruits
- Apple
- Banana

### Vegetables
- Carrot

### Tools
- Hammer
- Screwdriver
- Pliers

### Literature
- Novel
- Textbook
- Magazine
```

Advanced Template Techniques

Let's explore some more advanced techniques for working with prompt templates and variables:

In [11]:
```
# Template with formatted list
list_format_template = PromptTemplate(
    template="Analyze the following list of items:\n"
             "{% for item in items.split(',') %}"
             "- {{ item.strip() }}\n"
             "{% endfor %}"
             "\nProvide a summary of the list and suggest any patterns 
    input_variables=["items"]
)


# Using the formatted list template
print("Formatted List Template Result:")
prompt = list_format_template.format(
    items="Python, JavaScript, HTML, CSS, React, Django, Flask, Node.js"
)
print(get_completion(prompt))

print("\n" + "-"*50 + "\n")
```

Formatted List Template Result:
The list of items you provided consists of programming languages, frameworks, and technologies commonly used in web development. Here's a summary and analysis of the items:

Summary of the List:
1. **Programming Languages:**
   - **Python**: A versatile, high-level programming language known for its readability and wide range of applications, including web development, data analysis, artificial intelligence, and more.
   - **JavaScript**: A core web technology that enables interactive web pages and is essential for front-end development. It can also be used on the server side with environments like Node.js.

2. **Markup and Styling Languages:**
   - **HTML (HyperText Markup Language)**: The standard markup language for creating web pages. It structures the content on the web.
   - **CSS (Cascading Style Sheets)**: A stylesheet language used for describing the presentation of a document written in HTML. It controls layout, colors, fonts, and overall visual aesthetics.

3. **Frameworks and Libraries:**
   - **React**: A JavaScript library for building user interfaces, particularly single-page applications. It allows developers to create reusable UI components.
   - **Django**: A high-level Python web framework that encourages rapid development and clean, pragmatic design. It follows the "batteries-included" philosophy, providing many built-in features.
   - **Flask**: A lightweight Python web framework that is easy to use and flexible, making it suitable for small to medium-sized applications.
   - **Node.js**: A JavaScript runtime built on Chrome's V8 engine that allows developers to execute JavaScript on the server side. It is often used for building scalable network applications.

Patterns and Groupings:
1. **Web Development Focus**: All items are related to web development, either as languages for building web applications (Python, JavaScript) or as technologies for structuring (HTML) and styling (CSS) web content.

2. **Front-End vs. Back-End**:
   - **Front-End Technologies**: JavaScript, HTML, CSS, and React are primarily used for client-side development, focusing on the user interface and user experience.
   - **Back-End Technologies**: Python (with Django and Flask) and Node.js are used for server-side development, handling business logic, database interactions, and server management.

3. **Language and Framework Relationships**:
   - **Python Frameworks**: Django and Flask are both frameworks that utilize Python, showcasing its versatility in web development.
   - **JavaScript Frameworks**: React is a library that enhances JavaScript's capabilities for building dynamic user interfaces, while Node.js extends JavaScript to server-side programming.

4. **Full-Stack Development**: The combination of these technologies allows for full-stack development, where developers can work on both the front-end (React, HTML, CSS) and back-end (Django, Flask, Node.js) of web applications.

Conclusion:
The list represents a comprehensive set of tools and languages essential f

```
or modern web development. Understanding the relationships and roles of th
ese items can help developers choose the right technologies for their proj
ects, whether they are focusing on front-end, back-end, or full-stack deve
lopment.

--------------------------------------------------
```

In [12]:

```python
# Template with dynamic instructions
dynamic_instruction_template = PromptTemplate(
    template="Task: {{ task }}\n"
             "Context: {{ context }}\n"
             "Constraints: {{ constraints }}\n\n"
             "Please provide a solution that addresses the task, consid
    input_variables=["task", "context", "constraints"]
)

# Using the dynamic instruction template
print("Dynamic Instruction Template Result:")
prompt = dynamic_instruction_template.format(
    task="Design a logo for a tech startup",
    context="The startup focuses on AI-driven healthcare solutions",
    constraints="Must use blue and green colors, and should be simple en
)
print(get_completion(prompt))
```

Dynamic Instruction Template Result:
Logo Design Concept for AI-Driven Healthcare Startup

1. Logo Elements:
   - **Symbol:** A stylized brain combined with a medical cross. The brain represents AI and intelligence, while the medical cross symbolizes healthcare. The two elements can be intertwined to show the integration of technology and health.
   - **Typography:** Use a modern sans-serif font for the company name, ensuring it is clean and easy to read. The font should convey innovation and professionalism.

2. Color Palette:
   - **Primary Colors:**
     - **Blue (#007BFF):** Represents trust, reliability, and technology.
     - **Green (#28A745):** Symbolizes health, growth, and vitality.
   - **Usage:** The brain can be in blue, while the medical cross can be in green. This color combination will create a harmonious and professional look.

3. Design Style:
   - **Simplicity:** The logo should be minimalistic, avoiding intricate details that may not be visible at smaller sizes. The shapes should be bold and clear.
   - **Scalability:** Ensure that the logo maintains its integrity and recognizability when scaled down for use on business cards, websites, or app icons.

4. Layout:
   - **Horizontal Layout:** Place the symbol to the left of the company name for a balanced look. This layout is versatile for various applications, such as website headers and promotional materials.
   - **Vertical Layout Option:** For social media profiles or app icons, a stacked version with the symbol above the company name can be created.

5. Mockup:
   - Create a mockup of the logo on various backgrounds (white, light gray, and dark) to ensure visibility and adaptability across different platforms.

Final Thoughts:
This logo design concept effectively communicates the startup's focus on AI-driven healthcare solutions while adhering to the specified color constraints and ensuring simplicity for recognizability. The combination of the brain and medical cross symbolizes the innovative approach to healthcare, making it memorable and impactful.