Prompt templates

Language models take text as input - that text is commonly referred to as a prompt. Typically this is not simply a hardcoded string but rather a combination of a template, some examples, and user input. LangChain provides several classes and functions to make constructing and working with prompts easy.

What is a prompt template?

A prompt template refers to a reproducible way to generate a prompt. It contains a text string ("the template"), that can take in a set of parameters from the end user and generates a prompt.

A prompt template can contain:

- instructions to the language model,
- a set of few shot examples to help the language model generate a better response,
- a question to the language model.

Here's the simplest example:

```
from langchain import PromptTemplate


template = """/
You are a naming consultant for new companies.
```

```
What is a good name for a company that makes {product}?
"""

prompt = PromptTemplate.from_template(template)
prompt.format(product="colorful socks")
```

```
    You are a naming consultant for new companies.
    What is a good name for a company that makes colorful socks?
```

Create a prompt template

You can create simple hardcoded prompts using the `PromptTemplate` class. Prompt templates can take any number of input variables, and can be formatted to generate a prompt.

```
from langchain import PromptTemplate

# An example prompt with no input variables
no_input_prompt = PromptTemplate(input_variables=[], template="Tell me a joke.")
no_input_prompt.format()
# -> "Tell me a joke."

# An example prompt with one input variable
one_input_prompt = PromptTemplate(input_variables=["adjective"], template="Tell me a {adjective} joke.")
one_input_prompt.format(adjective="funny")
# -> "Tell me a funny joke."

# An example prompt with multiple input variables
multiple_input_prompt = PromptTemplate(
```

```
    input_variables=["adjective", "content"],
    template="Tell me a {adjective} joke about {content}."
)
multiple_input_prompt.format(adjective="funny", content="chickens")
# -> "Tell me a funny joke about chickens."
```

If you do not wish to specify `input_variables` manually, you can also create a `PromptTemplate` using `from_template` class method. `langchain` will automatically infer the `input_variables` based on the `template` passed.

```
template = "Tell me a {adjective} joke about {content}."

prompt_template = PromptTemplate.from_template(template)
prompt_template.input_variables
# -> ['adjective', 'content']
prompt_template.format(adjective="funny", content="chickens")
# -> Tell me a funny joke about chickens.
```

You can create custom prompt templates that format the prompt in any way you want. For more information, see Custom Prompt Templates.

Chat prompt template

Chat Models take a list of chat messages as input - this list commonly referred to as a `prompt`. These chat messages differ from raw string (which you would pass into a LLM model) in that every message is associated with a `role`.

For example, in OpenAI Chat Completion API, a chat message can be associated with the AI, human or system role. The model is supposed to follow instruction from system chat message more closely.

LangChain provides several prompt templates to make constructing and working with prompts easily. You are encouraged to use these chat related prompt templates instead of `PromptTemplate` when querying chat models to fully exploit the potential of underlying chat model.

```
from langchain.prompts import (
    ChatPromptTemplate,
    PromptTemplate,
    SystemMessagePromptTemplate,
    AIMessagePromptTemplate,
    HumanMessagePromptTemplate,
)
from langchain.schema import (
    AIMessage,
    HumanMessage,
    SystemMessage
)
```

To create a message template associated with a role, you use `MessagePromptTemplate`.

For convenience, there is a `from_template` method exposed on the template. If you were to use this template, this is what it would look like:

```
template="You are a helpful assistant that translates {input_language} to {output_language}."
system_message_prompt = SystemMessagePromptTemplate.from_template(template)
human_template="{text}"
human_message_prompt = HumanMessagePromptTemplate.from_template(human_template)
```

If you wanted to construct the `MessagePromptTemplate` more directly, you could create a PromptTemplate outside and then pass it in, eg:

```
prompt=PromptTemplate(
    template="You are a helpful assistant that translates {input_language} to {output_language}.",
    input_variables=["input_language", "output_language"],
)
system_message_prompt_2 = SystemMessagePromptTemplate(prompt=prompt)

assert system_message_prompt == system_message_prompt_2
```

After that, you can build a `ChatPromptTemplate` from one or more `MessagePromptTemplates`. You can use `ChatPromptTemplate`'s `format_prompt` -- this returns a `PromptValue`, which you can convert to a string or Message object, depending on whether you want to use the formatted value as input to an llm or chat model.

```
chat_prompt = ChatPromptTemplate.from_messages([system_message_prompt, human_message_prompt])

# get a chat completion from the formatted messages
chat_prompt.format_prompt(input_language="English", output_language="French", text="I love
programming.").to_messages()
```

```
    [SystemMessage(content='You are a helpful assistant that translates English to French.',
additional_kwargs={}),
     HumanMessage(content='I love programming.', additional_kwargs={})]
```