

```
= Giving a Chat Model Memory
:order: 5
:type: lesson
```

For a chat model to be helpful, it needs to remember what messages have been sent and received.

Without the ability to `_remember_` the chat model will not be able to act according to the context of the conversation.

For example, without a memory the conversation may go in circles:

```
[user] Hi, my name is Martin
```

```
[chat model] Hi, nice to meet you Martin
```

```
[user] Do you have a name?
```

```
[chat model] I am the chat model. Nice to meet you. What is your name?
```

```
== Conversation Memory
```

In the previous lesson, you created a chat model that responds to user's questions about surf conditions.

```
[%collapsible]
.Reveal the code
====
[source,python]
----
include:../3-chat-models/code/chat-model-context.py[tag=**]
----
====
```

In this lesson, you will add a memory to this program.

LangChain supports several [link:https://python.langchain.com/v0.2/docs/integrations/memory/](https://python.langchain.com/v0.2/docs/integrations/memory/)`[memory components^]`, which support different scenarios and storage solutions.

You are going to use the in-memory ``ChatMessageHistory`` memory component to temporarily store the conversation history between you and the chat model.

In the next lesson, you will then use the [link:https://python.langchain.com/v0.2/docs/integrations/memory/neo4j_chat_message_history/](https://python.langchain.com/v0.2/docs/integrations/memory/neo4j_chat_message_history/)`[Neo4j Chat Message History]` memory component to persist the conversation history in a Neo4j database.

```
== Add History to the Prompt
```

As each call to the LLM is stateless, you need to include the chat history in every call to the LLM.

You can modify the prompt template to include the chat history as a list of messages using a `MessagesPlaceholder` object.

```
[source, python]
----
include::code/chat-model-memory.py[tag=import-messages]

include::code/chat-model-memory.py[tag=prompt]
----
```

The `chat_history` variable will contain the conversation history.

== Chat Message History

To keep the message history, you will need to wrap the `chat_chain` in a link:[https://python.langchain.com/v0.2/docs/concepts/#runnable-interface\[_Runnable_\]](https://python.langchain.com/v0.2/docs/concepts/#runnable-interface[_Runnable_]).

Specifically, you will use the link:[https://api.python.langchain.com/en/latest/runnables/langchain_core_runnables.history.RunnableWithMessageHistory.html\[_RunnableWithMessageHistory_\]](https://api.python.langchain.com/en/latest/runnables/langchain_core_runnables.history.RunnableWithMessageHistory.html[_RunnableWithMessageHistory_]) runnable which will use the memory component to store and retrieve the conversation history.

First, you will need to create a `ChatMessageHistory` memory component and a function that the `RunnableWithMessageHistory` will use to get the memory component.

```
[source, python]
----
include::code/chat-model-memory.py[tag=import-memory]

include::code/chat-model-memory.py[tag=memory]
----
```

The `get_memory` function will return the `ChatMessageHistory` memory component.

Note how it expects a `session_id` parameter, this would be used to identify the specific conversation (or session).

As there will only be one conversation in memory at a time, you can ignore this parameter.

You can now create a new chain using the `RunnableWithMessageHistory`, passing the `chat_chain` and the `get_memory` function.

```
[source, python]
----
include::code/chat-model-memory.py[tag=import-runnable]

include::code/chat-model-memory.py[tag=chat-history]
----
```

The `input_messages_key` and `history_messages_key` parameters are the

keys in the prompt that will be populated with the user's question and the chat history.

== Invoke the Chat Model

When you call the ``chat_with_message_history`` chain, the user's question and the response will be stored in the ``ChatMessageHistory`` memory component.

Every subsequent call to the ``chat_with_message_history`` chain will include the chat history in the prompt.

When you ask the chat model multiple questions, the LLM will use the context from the previous questions when responding.

```
[source, python]
```

```
----
response = chat_with_message_history.invoke(
    {
        "context": current_weather,
        "question": "Hi, I am at Watergate Bay. What is the surf like?"
    },
    config={"configurable": {"session_id": "none"}}
)
print(response)

response = chat_with_message_history.invoke(
    {
        "context": current_weather,
        "question": "Where I am?"
    },
    config={"configurable": {"session_id": "none"}}
)
print(response)
----
```

```
[user] Hi, I am at Watergate Bay. What is the surf like?
```

```
[chat model] Dude, stoked you're at Watergate Bay! The surf is lookin'
pretty chill, about 3ft waves rollin' in. But watch out for those onshore
winds, they might mess with your flow.
```

```
[user] Where I am?
```

```
[chat model] You're at Watergate Bay, dude!
```

```
[NOTE]
```

```
====
```

To invoke the chat model you need to specify a ``session_id`` in the ``config`` which will be passed to the ``get_memory`` function.

As there is no need to store multiple conversations in memory, you can use

the same `session_id` for all conversations.

You will use the `session_id` in the next lesson when storing the conversation history in Neo4j.

====

Try creating a simple loop and ask the chat model a few questions:

[source, python]

```
include::code/chat-model-memory.py[tag=loop]
```

[%collapsible]

.Click to reveal the complete code.

====

[source,python]

```
include::code/chat-model-memory.py[tag=**]
```

====

== Check Your Understanding

```
include::questions/1-state.adoc[leveloffset=+1]
```

[.summary]

== Lesson Summary

In this lesson, you learned how to use the `ChatMessageHistory` to store the conversation history between you and the LLM.

In the next lesson, you will learn how to use Langchain to interact with Neo4j and store the conversation history in a graph database.