

```
= Agents
:order: 7
:type: lesson
:disable-cache: true
```

In this lesson, you will learn how to create an
link:[https://python.langchain.com/docs/modules/agents\[agent^\]](https://python.langchain.com/docs/modules/agents[agent^]).

Agents wrap a model and give it access to a set of `_tools_`. These tools may access additional data sources, APIs, or functionality. The model is used to determine which of the tools to use to complete a task.

The agent you will create will be able to chat about movies and search YouTube for movie trailers.

```
== Tools
```

A tool is a specific abstraction around a function that makes it easy for a language model to interact with it.
link:[https://python.langchain.com/v0.2/docs/integrations/tools/\[Langchain provides several tools^\]](https://python.langchain.com/v0.2/docs/integrations/tools/[Langchain provides several tools^]) out of the box, and you can create tools to extend the functionality of your agents.

You will use the
link:[https://python.langchain.com/v0.2/docs/integrations/tools/youtube/\[YouTube Tool^\]](https://python.langchain.com/v0.2/docs/integrations/tools/youtube/[YouTube Tool^]) to search YouTube for movie trailers.

```
== Movie trailer agent
```

Review the program below, before running it.

```
[source,python]
```

```
----
```

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

```
----
```

```
[%collapsible]
```

```
.Click here to see a typical output from this program
```

```
====
```

```
[user] Find a movie where aliens land on earth.
```

```
[chat model] Sure, I can help you with that. One movie I would recommend where aliens land on Earth is "Arrival" (2016). It's a science fiction film directed by Denis Villeneuve. The story follows a linguist who is recruited by the military to help communicate with an alien species that has landed on Earth. It's a thought-provoking and visually stunning movie that explores themes of communication, time, and the human experience. I hope you enjoy it!
```

```
====
```

Based on your understanding from previous lessons, you should be able to identify the following:

- . A chat model is being used to have a conversation about movies
- . The prompt which sets the context for the LLM and the input variables
- . That memory is used to store the conversation history in a Neo4j database

In addition to the above, the following is new:

```
. A tool is created using the chain:
+
[source,python]
----
include::code/chat-agent.py[tag=tools]
----
. An agent is created that uses the tool:
+
[source, python]
----
include::code/chat-agent.py[tag=agent]
----
. The agent is wrapped in a `RunnableWithMessageHistory` chain that allows
it to interact with the memory:
+
[source, python]
----
include::code/chat-agent.py[tag=chat_agent]
----
```

Add your `openai_api_key` and update the `Neo4jGraph` connection details to run the program.

```
[%collapsible]
.Click to reveal your Sandbox connection details
====
Connection URL:: [copy]#bolt://{sandbox-ip}:{sandbox-boltPort}#
Username:: [copy]#{sandbox-username}#
Password:: [copy]#{sandbox-password}#
=====
```

=== Creating tools

Tools are interfaces that an agent can interact with. You can [link:https://python.langchain.com/v0.2/docs/how_to/custom_tools/](https://python.langchain.com/v0.2/docs/how_to/custom_tools/) [create custom tools^] able to perform any functionality you want.

In this example, the Tool is created from a function. The function is the `movie_chat.invoke` method.

```
[source, python]
----
include::code/chat-agent.py[tag=tools]
----
```

The ``name`` and ``description`` help the LLM select which tool to use when presented with a question. The ``func`` parameter is the function that will be called when the tool is selected. The ``return_direct`` flag indicates that the tool will return the result directly.

Agents support multiple tools, so you pass them to the agent as a list (``tools``).

=== Initializing an agent

The following code creates the agent:

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

There are different
link:https://python.langchain.com/docs/modules/agents/agent_types/[types of agents^] that you can create. This example creates a `_ReAct_` - Reasoning and Acting) agent type.

An agent requires a prompt. You could create a prompt, but in this example, the program pulls a pre-existing prompt from the
link:<https://smith.langchain.com/hub/>[Langsmith Hub^].

The
link:<https://smith.langchain.com/hub/hwchase17/react-chat?organizationId=d9a804f5-9c91-5073-8980-3d7112f1cbd3>[`hwcase17/react-chat`^] prompt instructs the model to provide an answer using the tools available in a specific format.

The ``create_react_agent`` function creates the agent and expects the following parameters:

- * The ``llm`` that will manage the interactions and decide which tool to use
- * The ``tools`` that the agent can use
- * The ``prompt`` that the agent will use

The ``AgentExecutor`` class runs the agent. It expects the following parameters:

- * The ``agent`` to run
- * The ``tools`` that the agent can use
- * The ``memory`` which will store the conversation history

[TIP]

.AgentExecutor parameters
====

You may find the following additional parameters useful when initializing an agent:

- * ``max_iterations`` - the maximum number of iterations to run the LLM for.

This is useful in preventing the LLM from running for too long or entering an infinite loop.

* `verbose` - if `True` the agent will print out the LLM output and the tool output.

* `handle_parsing_errors` - if `True` the agent will handle parsing errors and return a message to the user.

```
[source, python]
```

```
----
```

```
agent_executor = AgentExecutor(  
    agent=agent,  
    tools=tools,  
    max_iterations=3,  
    verbose=True,  
    handle_parse_errors=True  
)
```

```
----
```

```
=====
```

```
=== Multiple tools
```

A key advantage of using an agent is that they can use multiple tools. Access to multiple tools allows you to create agents that can perform several specific tasks.

You can extend this example to allow it to search YouTube for movie trailers by adding the [link:https://python.langchain.com/v0.2/docs/integrations/tools/youtube/](https://python.langchain.com/v0.2/docs/integrations/tools/youtube/)[YouTube Tool^] to the `tools` list.

Firstly, you will need to install the [link:https://pypi.org/project/youtube-search/](https://pypi.org/project/youtube-search/)[`youtube-search`^] package.

```
[source, bash]
```

```
----
```

```
pip install youtube-search
```

```
----
```

Import the `YouTubeSearchTool` and create a new tool.

```
[source, python]
```

```
----
```

```
include::code/chat-agent-trailer.py[tag=import-youtube]
```

```
include::code/chat-agent-trailer.py[tag=youtube]
```

```
----
```

The `YouTubeSearchTool` tool expects a search term and the number of results passed as a comma-separated string.

The agent may pass queries containing commas, so create a function to strip the commas from the query and pass the query to the `YouTubeSearchTool`.

```
[source, python]
```

```
----
```

```
include::code/chat-agent-trailer.py[tag=trailer-search]
```

```
----
```

Finally, add the ``call_trailer_search`` function to the ``tools`` list.

```
[source, python]
```

```
----
```

```
include::code/chat-agent-trailer.py[tag=tools]
```

```
----
```

```
[%collapsible]
```

```
.Click here to reveal the complete program
```

```
====
```

```
[source, python]
```

```
----
```

```
include::code/chat-agent-trailer.py[tag=**]
```

```
----
```

```
====
```

The model will use the ``name`` and ``description`` for each tool to decide which tool to use.

When prompted to find a movie trailer, the model should use the ``YouTubeSearchTool`` tool.

```
[user] Find the movie trailer for the Matrix.
```

```
[agent] Here are the movie trailers for "The Matrix":
```

```
The Matrix - Official Trailer #1 -
```

```
https://www.youtube.com/watch?v=vKQi3bBA1y8&pp=ygUKVGhlIE1hdHJpeA%3D%3D
```

```
The Matrix - Official Trailer #2 -
```

```
https://www.youtube.com/watch?v=xrYg\_qKX-aI&pp=ygUKVGhlIE1hdHJpeA%3D%3D
```

However, when asked about movies, genres or plots, the model will use the ``chat_chain`` tool.

```
[user] Find a movie about the meaning of life
```

```
[agent] Certainly! One movie that explores the meaning of life is "The Tree of Life" directed by Terrence Malick. It follows the journey of a young boy as he grows up in the 1950s and reflects on his experiences and the meaning of existence. It's a visually stunning and thought-provoking film that delves into existential questions.
```

As the agent also uses the conversation memory, you can refer back to the previous questions, such as finding a trailer for a movie it has

recommended:

[user] Can you find the trailer

[agent] Here are two links to the trailer for "The Tree of Life":

Link 1 -

<https://www.youtube.com/watch?v=RrAz1YLh8nY&pp=ygUQVGhlIFRyZWUgb2YgTGlmZQ%3D%3D>

Link 2 -

<https://www.youtube.com/watch?v=cv-dH5gHi1c&pp=ygUQVGhlIFRyZWUgb2YgTGlmZQ%3D%3D>

Agents and tools allow you to create more adaptable and flexible models to perform multiple tasks.

== Check Your Understanding

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

[.summary]

== Summary

In this lesson, you learned how to create an agent to use multiple tools.

In the next lesson, you will learn how to use Neo4j as a vector store using Langchain ****Receivers****.