```
= Chains
:order: 3
:type: lesson

In this lesson, you will learn about chains and how to use them to create
reusable components.

Chains allows you to combine language models with different data sources
and third-party APIs.

== LCEL

The simplest chain combines a prompt template with an LLM and returns a
response.

You can create a chain using LangChain Expression Language (LCEL).
LCEL is a declarative way to chain Langchain components together.

Components are chained together using the `|` operator.

[source, python, role=nocopy noplay]
----
chain = prompt | llm
----

Previously, you created a program that used a prompt template and an LLM
to generate a response about fruit.

[%collapsible]
.Click to reveal the code for the program.
====
[source,python]
----
include::../2-initialising-the-llm/code/llm_prompt.py[tag=**]
----
====

You can combine this program into a chain and create a reusable component.

[source,python]
----
include::code/llm_chain.py[tag=**]
----

Note how the `llm_chain` is created by chaining the `template` and the
`llm`.

[source,python]
----
include::code/llm_chain.py[tag=llm_chain]
----

You `invoke` the `llm_chain` passing the template parameters as a
```

```
dictionary.

[source,python]
----
include::code/llm_chain.py[tag=invoke]
----

== Output Parsers

The output from the chain is typically a string, and you can specify an
link:https://python.langchain.com/docs/modules/model_io/output_parsers/[ou
tput parser^] to parse the output.

Adding a `StrOutputParser` to the chain would ensure a string.

[source,python]
----
include::code/llm_chain_output.py[tag=import]

include::code/llm_chain_output.py[tag=llm_chain]
----

You can change the prompt to instruct the LLM to return a specific output
type.
For example, return JSON by specifying `Output JSON` and give a format in
the prompt:

[source,python]
----
template = PromptTemplate.from_template("""
You are a cockney fruit and vegetable seller.
Your role is to assist your customer with their fruit and vegetable needs.
Respond using cockney rhyming slang.

Output JSON as {{"description": "your response here"}}

Tell me about the following fruit: {fruit}
""")
----

You can ensure Langchain parses the response as JSON by specifying
`SimpleJsonOutputParser` as the `output_parser`:

[source,python]
----
include::code/llm_chain_output_json.py[tag=import]

include::code/llm_chain_output_json.py[tag=llm_chain]
----

The benefits of using chains are:

* **Modularity**: LangChain provides many modules that can be used to
```

build language model applications. These modules can be used as
stand-alones in simple applications and they can be combined for more
complex use cases.

* **Customizability**: Most LangChain applications allow you to configure
the LLM and/or the prompt used, so knowing how to take advantage of this
will be a big enabler.

* **Ease** of Use: The components are designed to be easy to use,
regardless of whether you are using the rest of the LangChain framework or
not.

* **Standard** Interface: LangChain provides a standard interface for
chains, enabling developers to create sequences of calls that go beyond a
single LLM call.

== Check Your Understanding

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

[.summary]
== Lesson Summary

In this lesson, you learned about LLM chains and how they can group a
prompt, LLM, and output parser.

In the next lesson, you will learn about chat models.