https://python.langchain.com/docs/get_started/introduction/

- framework to develop apps powered by LLMs

→ Libraries:

- langchain-core: base abstractions and LC Expression language
- langchain-community: 3rd party integrations (langchain-openai, langchain-anthropic, etc.)
- langchain: chains, agents and retrieval strategies that make up app's cognitive architect.

→ Quickstart

https://python.langchain.com/docs/get_started/quickstart/

- Retrieval used when have too much data to pass to LLM directly, it will only fetch the most relevant data
  - ↳ can be backed by anything (SQL table, internet, vector store, etc.)

First, we need to load the data that we want to index. To do this, we will use the WebBaseLoader. This requires installing BeautifulSoup:

```
pip install beautifulsoup4
```

After that, we can import and use WebBaseLoader.

```
from langchain_community.document_loaders import WebBaseLoader
loader = WebBaseLoader("https://docs.smith.langchain.com/user_guide")

docs = loader.load()
```

Next, we need to index it into a vectorstore. This requires a few components, namely an embedding model and a vectorstore.

For embedding models, we once again provide examples for accessing via API or by running local models.

OpenAI (API) | Local (using Ollama) | Cohere (API)

Make sure you have Ollama running (same set up as with the LLM).

```
from langchain_community.embeddings import OllamaEmbeddings

embeddings = OllamaEmbeddings()
```

- we use embedding model to ingest docs into a vectorstore (FAISS)
- the chain will take incoming question, look up relevant docs and pass them along w the original question into the LLM

First, let's set up the chain that takes a question and the retrieved documents and generates an answer.

```
from langchain.chains.combine_documents import create_stuff_documents_chain

prompt = ChatPromptTemplate.from_template("""Answer the following question based only on the prov

<context>
{context}
</context>

Question: {input}""")

document_chain = create_stuff_documents_chain(llm, prompt)
```

If we wanted to, we could run this ourselves by passing in documents directly:

```
from langchain_core.documents import Document

document_chain.invoke({
    "input": "how can langsmith help with testing?",
    "context": [Document(page_content="langsmith can let you visualize test results")]
})
```

API Reference:

- Document

However, we want the documents to first come from the retriever we just set up. That way, we can use the retriever to dynamically select the most relevant documents and pass those in for a given question.

```
from langchain.chains import create_retrieval_chain

retriever = vector.as_retriever()
retrieval_chain = create_retrieval_chain(retriever, document_chain)
```

API Reference:

- create_retrieval_chain

We can now invoke this chain. This returns a dictionary - the response from the LLM is in the `answer` key

```
response = retrieval_chain.invoke({"input": "how can langsmith help with testing?"})
print(response["answer"])

# LangSmith offers several features that can help with testing:...
```

This answer should be much more accurate!

→ Conversation Retrieval Chain