



Markdown

Markdown is a lightweight markup language for creating formatted text using a plain-text editor.

This covers how to load `Markdown` documents into a document format that we can use downstream.

```
# !pip install unstructured > /dev/null
```

```
from langchain.document_loaders import UnstructuredMarkdownLoader
```

```
markdown_path = "../../../../../README.md"
loader = UnstructuredMarkdownLoader(markdown_path)
```

```
data = loader.load()
```



```
data
```

```
[Document(page_content="ð\x9f|\x9cï,\x8fð\x9f"\x97
LangChain\n\nâ\x9ai Building applications with LLMs through
composability â\x9ai\n\nLooking for the JS/TS version? Check out
LangChain.js.\n\nProduction Support: As you move your LangChains into
production, we'd love to offer more comprehensive support.\nPlease fill
out this form and we'll set up a dedicated support Slack
channel.\n\nQuick Install\n\npip install langchain\nor\nconda install
langchain -c conda-forge\n\nð\x9f" What is this?\n\nLarge language
models (LLMs) are emerging as a transformative technology, enabling
developers to build applications that they previously could not.
However, using these LLMs in isolation is often insufficient for
creating a truly powerful app – the real power comes when you can
combine them with other sources of computation or knowledge.\n\nThis
library aims to assist in the development of those types of
applications. Common examples of these applications include:\n\nâ\x9d"
Question Answering over specific documents\n\nDocumentation\n\nEnd-to-
```

```

end Example: Question Answering over Notion Database\n\nð\x9f'-
Chatbots\n\nDocumentation\n\nEnd-to-end Example: Chat-
LangChain\n\nð\x9f\x96 Agents\n\nDocumentation\n\nEnd-to-end Example:
GPT+WolframAlpha\n\nð\x9f"\x96 Documentation\n\nPlease see here for
full documentation on:\n\nGetting started (installation, setting up the
environment, simple examples)\n\nHow-To examples (demos, integrations,
helper functions)\n\nReference (full API docs)\n\nResources (high-level
explanation of core concepts)\n\nð\x9f\x9a\x80 What can this help with?
\n\nThere are six main areas that LangChain is designed to help
with.\n\nThese are, in increasing order of complexity:\n\nð\x9f"\x83 LLMs
and Prompts:\n\nThis includes prompt management, prompt optimization, a
generic interface for all LLMs, and common utilities for working with
LLMs.\n\nð\x9f"\x97 Chains:\n\nChains go beyond a single LLM call and
involve sequences of calls (whether to an LLM or a different utility).
LangChain provides a standard interface for chains, lots of
integrations with other tools, and end-to-end chains for common
applications.\n\nð\x9f"\x9a Data Augmented Generation:\n\nData
Augmented Generation involves specific types of chains that first
interact with an external data source to fetch data for use in the
generation step. Examples include summarization of long pieces of text
and question/answering over specific data sources.\n\nð\x9f\x96
Agents:\n\nAgents involve an LLM making decisions about which Actions
to take, taking that Action, seeing an Observation, and repeating that
until done. LangChain provides a standard interface for agents, a
selection of agents to choose from, and examples of end-to-end
agents.\n\nð\x9f\x9a Memory:\n\nMemory refers to persisting state
between calls of a chain/agent. LangChain provides a standard interface
for memory, a collection of memory implementations, and examples of
chains/agents that use memory.\n\nð\x9f\x9a Evaluation:\n\n[BETA]
Generative models are notoriously hard to evaluate with traditional
metrics. One new way of evaluating them is using language models
themselves to do the evaluation. LangChain provides some prompts/chains
for assisting in this.\n\nFor more information on these concepts,
please see our full documentation.\n\nð\x9f'\x81 Contributing\n\nAs an
open-source project in a rapidly developing field, we are extremely
open to contributions, whether it be in the form of a new feature,
improved infrastructure, or better documentation.\n\nFor detailed
information on how to contribute, see here.", metadata={'source':
'../../../../../README.md'}}]

```

Retain Elements

Under the hood, Unstructured creates different "elements" for different chunks of text. By default we combine those together, but you can easily keep that separation by specifying

```
mode="elements".
```

```
loader = UnstructuredMarkdownLoader(markdown_path, mode="elements")
```

```
data = loader.load()
```

```
data[0]
```

```
Document(page_content='ð\x9f|\x9cï.\x8fð\x9f"\x97 LangChain',  
metadata={'source': '../.../README.md', 'page_number': 1,  
'category': 'Title'})
```