

# Advanced Generative AI: Models, Tools and Applications



# LangChain for LLM Application Development Part-1



# Quick Recap



- How do the main parts and design of advanced prompt engineering help in making smart applications? Using the best methods for designing prompts and applying techniques such as zero-shot prompting, few-shot prompting, and more.
- What are the core components and architecture of large language models (LLMs), and how do they contribute to the development of context-aware applications in LangChain?

# Engage and Think



XYZ Corp is reviewing its annual sustainability report. With a focus on renewable energy, waste reduction, employee well-being, and ethical governance, XYZ aims to understand its environmental and social impact.

How might the company leverage innovative approaches to measure and communicate its environmental and social impact beyond traditional metrics?

# Learning Objectives

By the end of this lesson, you will be able to:

- 🔗 Apply LangChain's input or output mechanisms, including prompts and models for improved language processing in your applications
- 🔗 Illustrate document loading and splitting procedures in LangChain effectively to streamline data handling
- 🔗 Utilize embeddings in generative AI for enhanced model performance

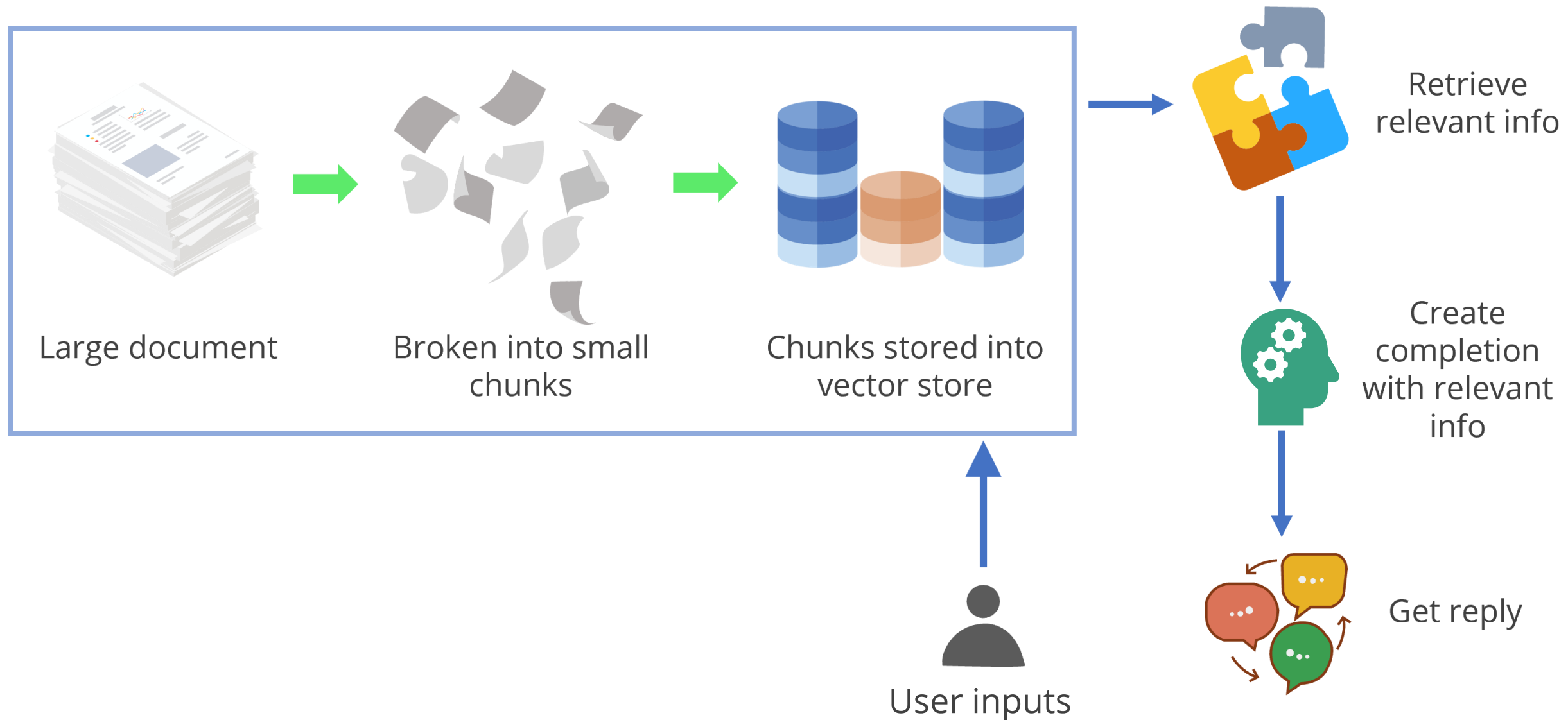




## **Model I/O: Prompts, Language Models, and Parsers**

# Flow of Chatbot Application

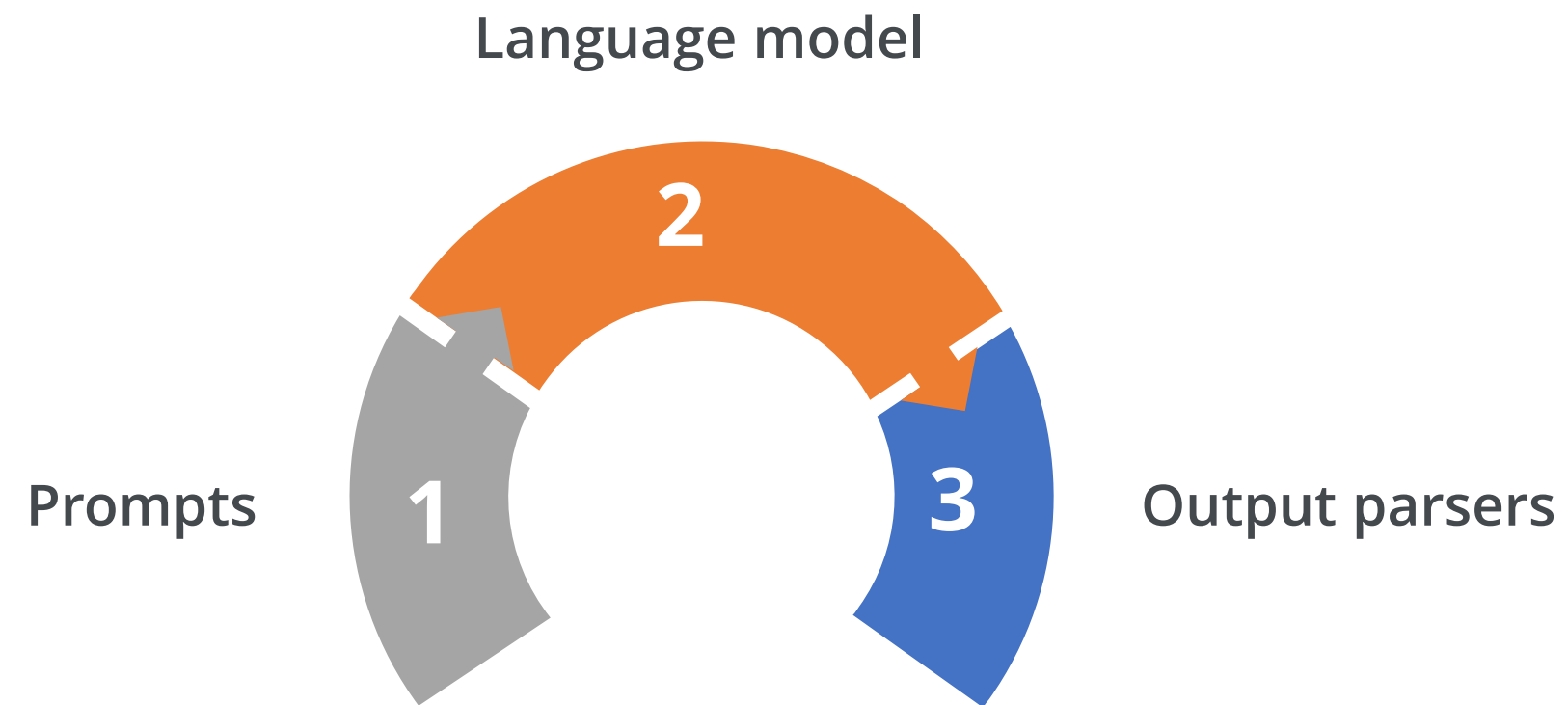
Now, let's explore the process of developing a chatbot application in detail and observe its functionality in action.



# Model I/O

In the world of language models, the interaction between the user, the model, and the output is crucial.

This interaction is facilitated through three key components, as given below:





# Model I/O: Prompts

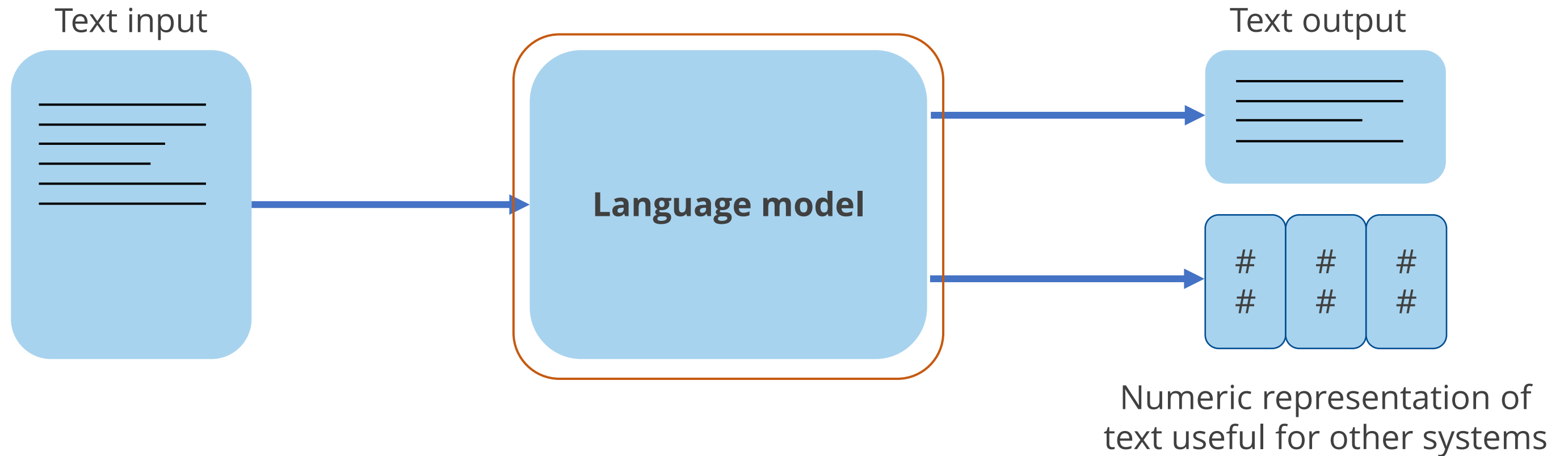
Prompts, as learned in the last lesson, serve as inputs for a language model and guide it on what to generate.



For instance, to make a story about a brave king, the prompt could be  
**Once upon a time, there was a brave king...**

# Model I/O: Language Models

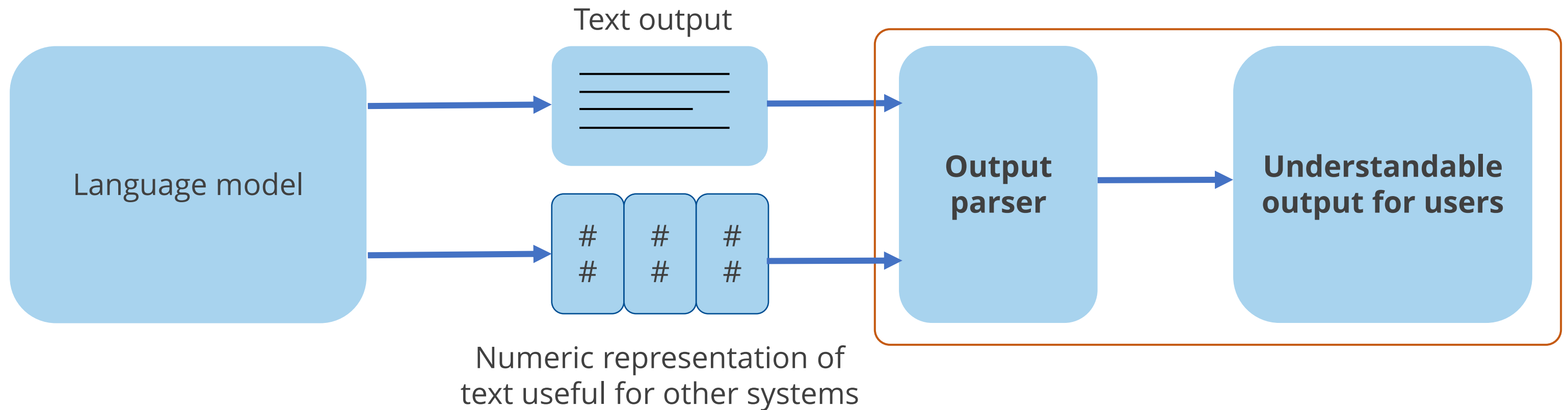
Language models are the heart of the system, and they take the prompts and generate text based on their training.



given the previous prompt, a language model might continue the story:  
**...who lived in a grand castle.**

# Model I/O: Output Parsers

Output parsers transform the raw output from the language model into a format that is useful and accessible.



They might correct grammar, ensure the text makes sense, or extract specific information.

# Demo: LangChain-Models, Prompts, and Output Parsers



**Duration: 25 minutes**

- This demo guides you through the process of using the LangChain library to interact with OpenAI's GPT-3.5-turbo model.
- This example helps learners understand how to create prompts, generate responses in different styles, and parse the output using the StructuredOutputParser class.

**Context:** The example used throughout the tutorial is a customer who bought a new gaming console, which arrived in 2 days, and the customer believes it was worth the price.

## Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

## Quick Check

Which of the following statements is NOT true about the interaction between the user, the model, and the output in the world of language models?

- A. Prompts are the inputs that you provide to a language model.
- B. Language models generate text based on their training.
- C. Output parsers take the raw output from the language model and format it in a way that's useful for us.
- D. Language models correct grammar, ensure the text makes sense, and extract specific information.

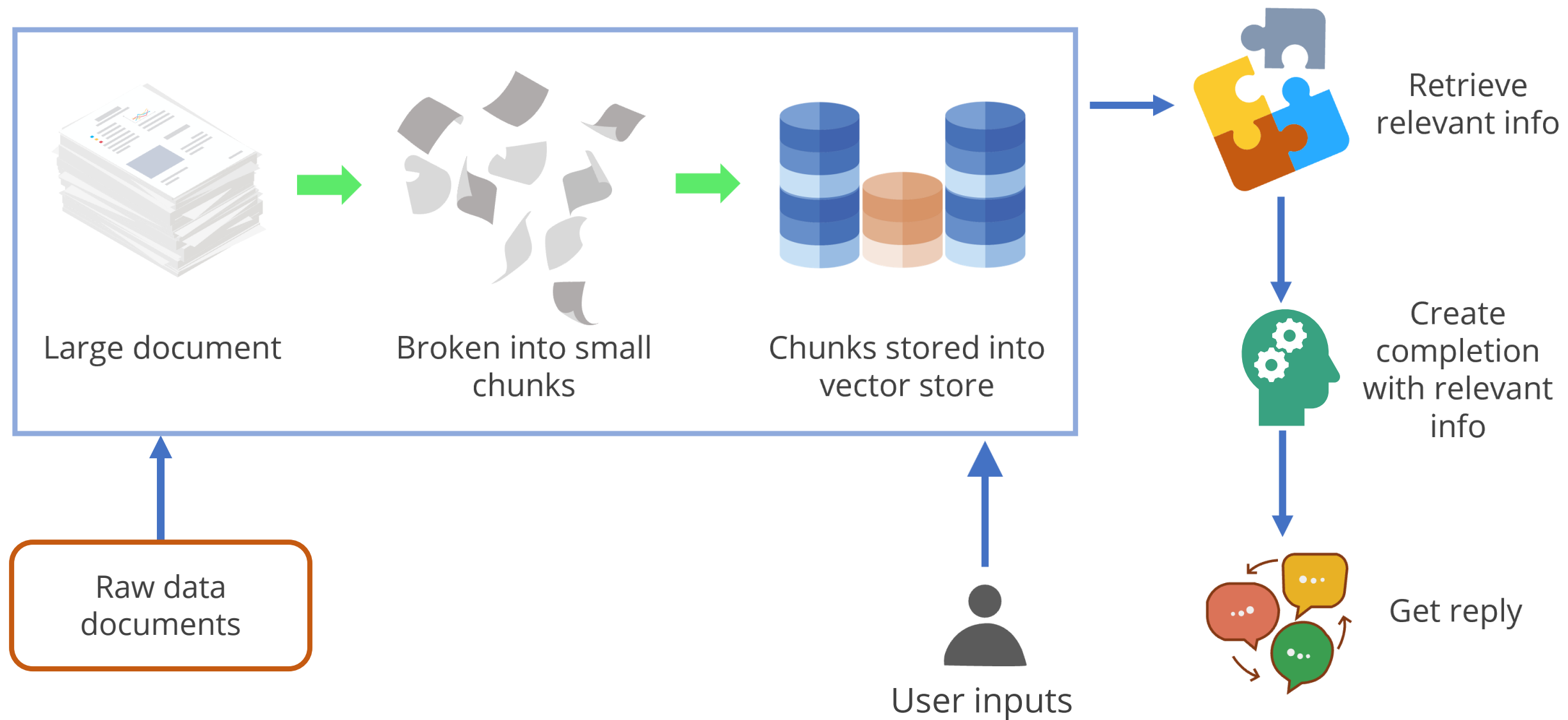




## Document Loaders

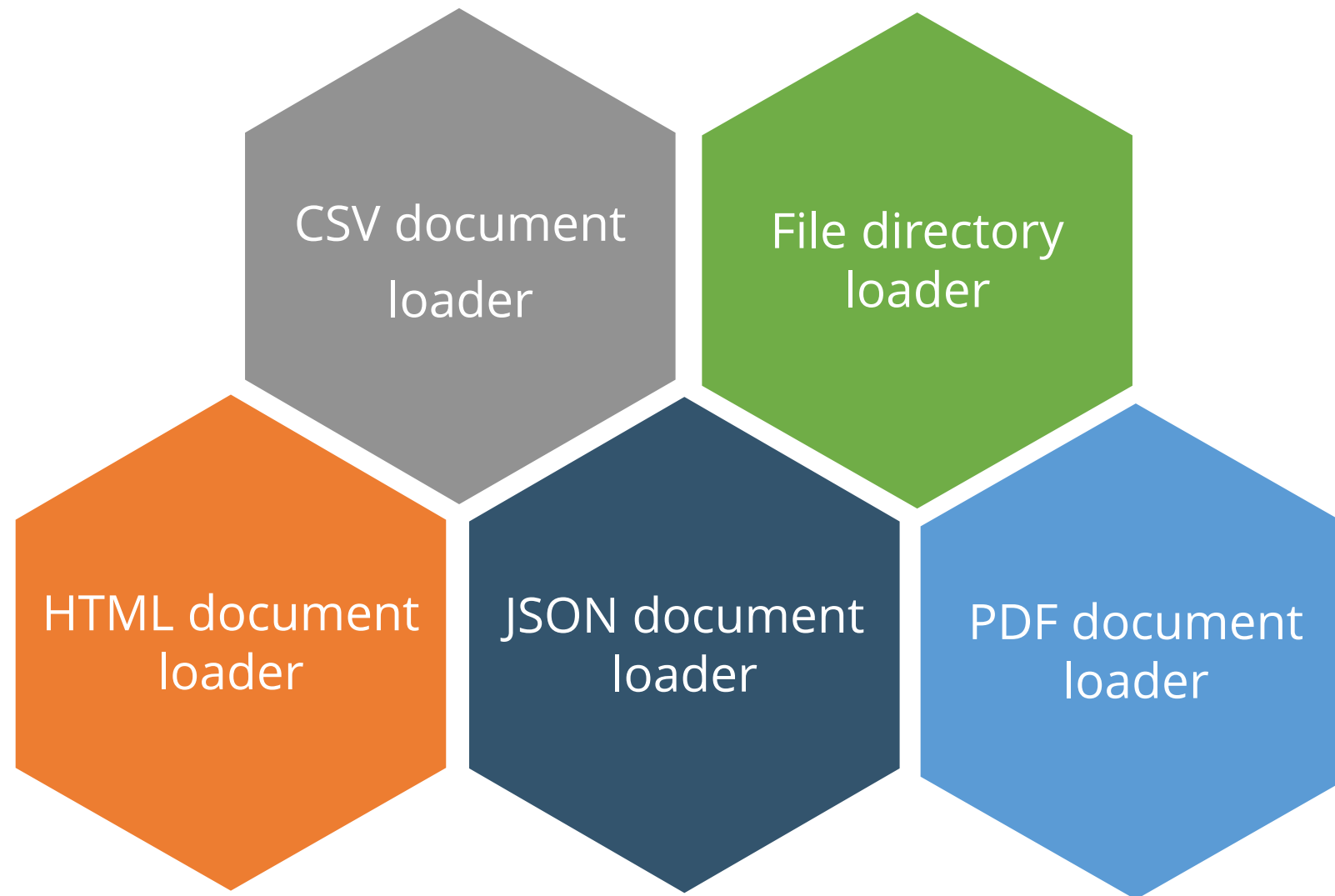
# Chatbot Application Flow

The initial step is to import the raw document and explore the process of importing documents with examples.



# Document Loaders

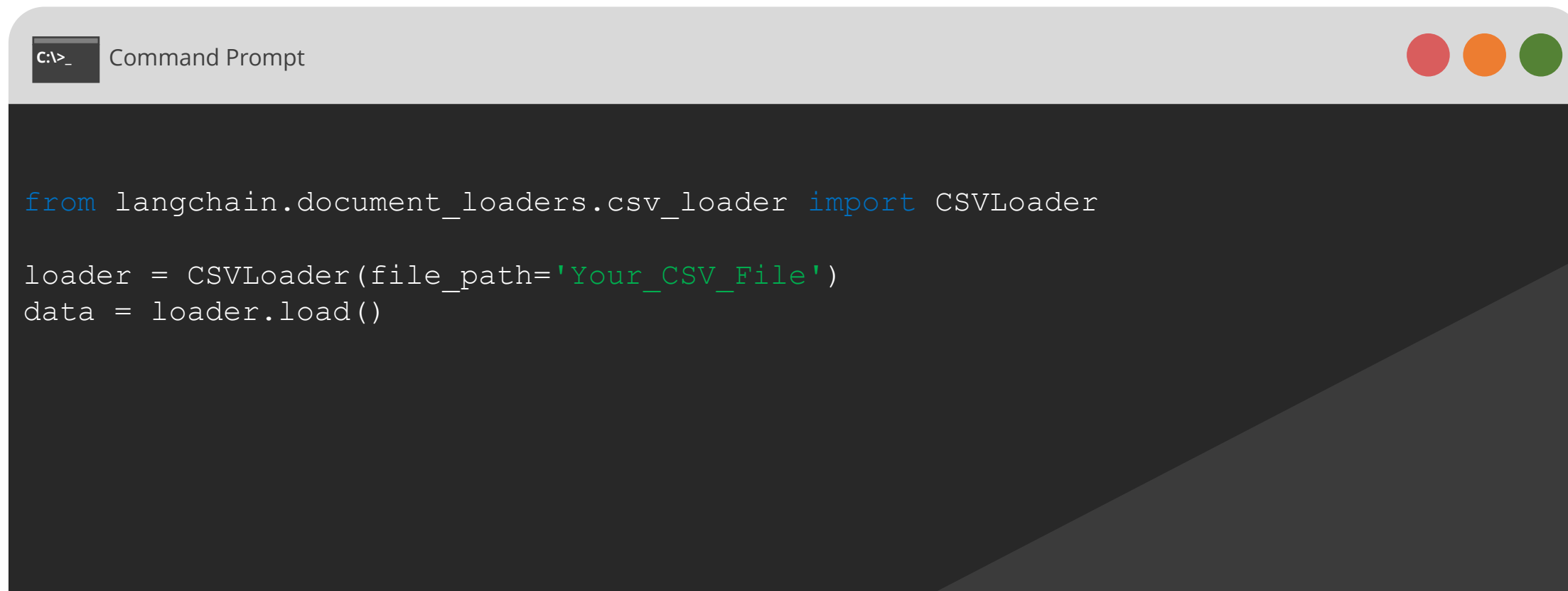
The following are major document loaders in LangChain:





# CSV Document Loader

A CSV (comma-separated values) file is a type of text file that uses a comma as a delimiter to separate values.

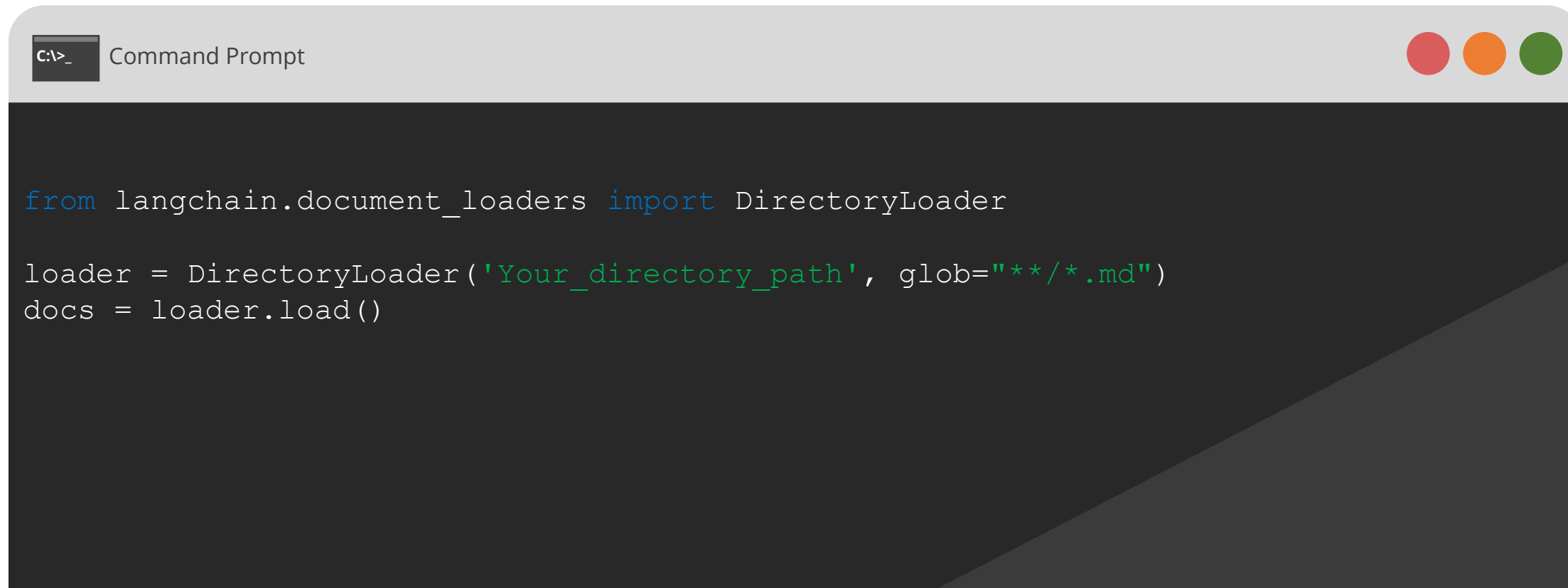


```
from langchain.document_loaders.csv_loader import CSVLoader

loader = CSVLoader(file_path='Your_CSV_File')
data = loader.load()
```

# File Directory Loader

A file directory is like a digital filing cabinet where you organize and store files. It is a structure to manage data.

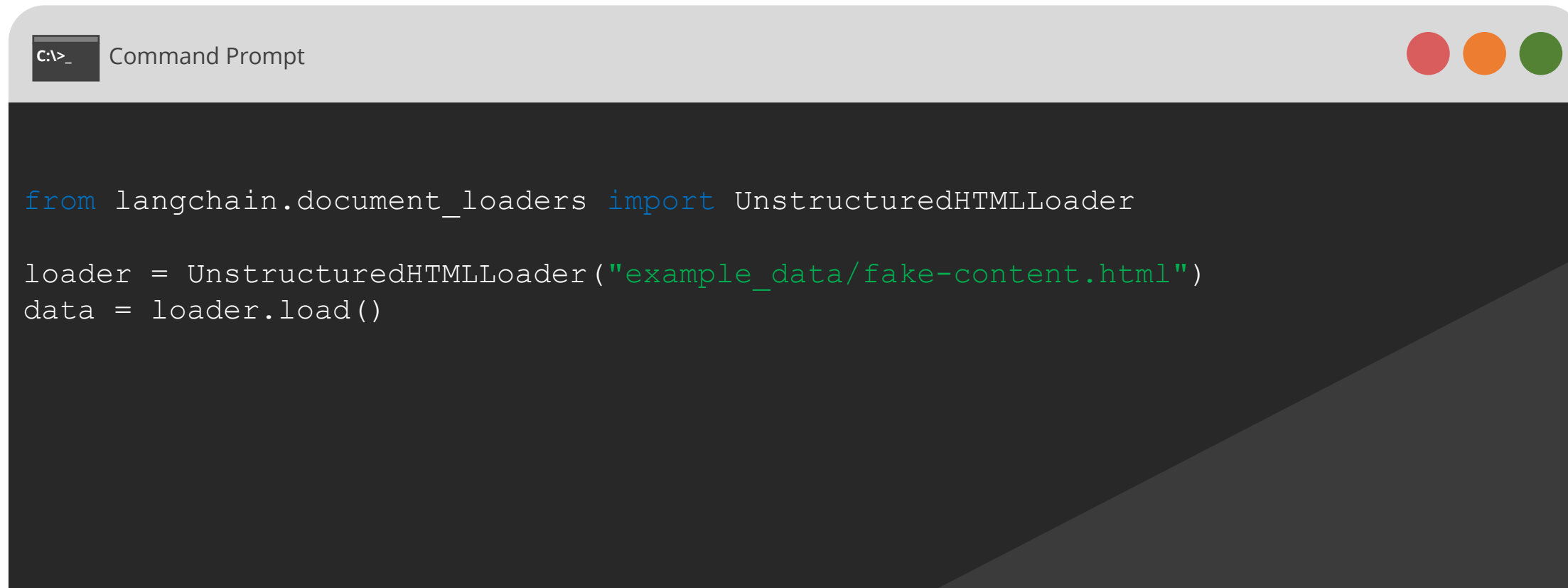


```
from langchain.document_loaders import DirectoryLoader

loader = DirectoryLoader('Your_directory_path', glob="**/*.md")
docs = loader.load()
```

# HTML Document Loader

HTML (hypertext markup language) is the standard markup language used for creating documents intended for web browser display.

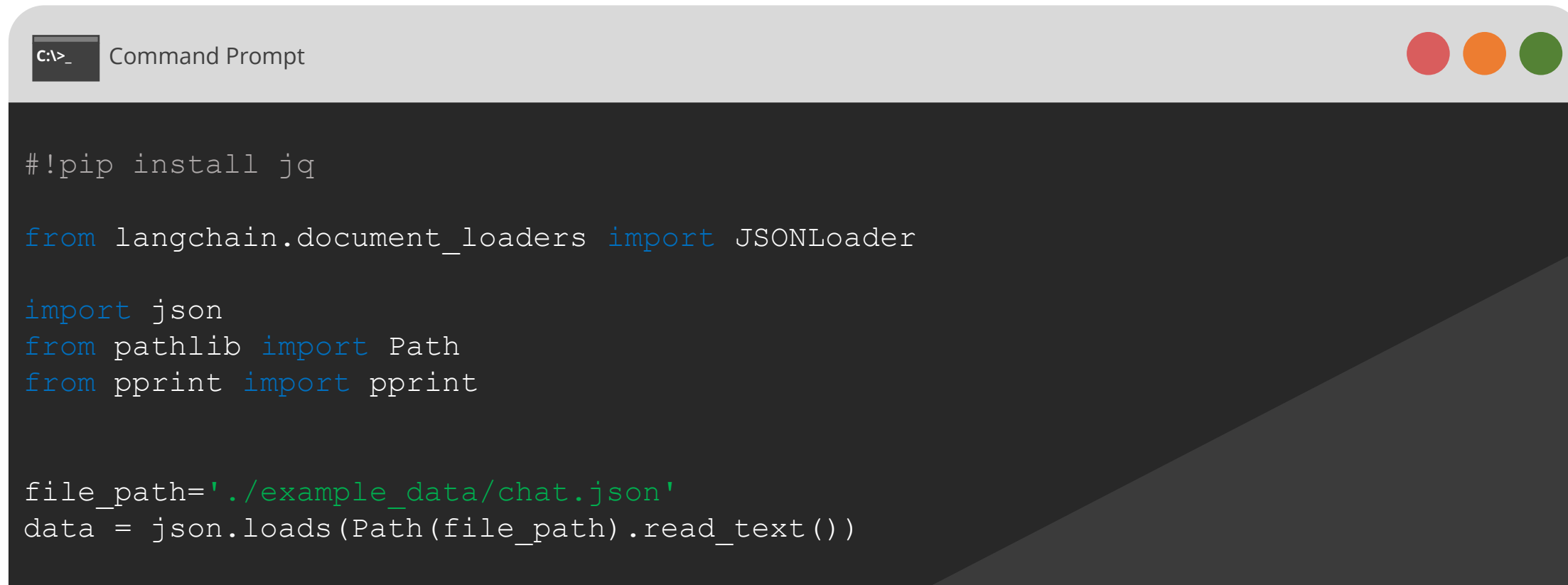


```
from langchain.document_loaders import UnstructuredHTMLLoader

loader = UnstructuredHTMLLoader("example_data/fake-content.html")
data = loader.load()
```

# JSON Document Loader

JSON (JavaScript object notation) is an open standard file format and data interchange format that employs human-readable text to store and transmit data objects.



```
#!pip install jq

from langchain.document_loaders import JSONLoader

import json
from pathlib import Path
from pprint import pprint

file_path='./example_data/chat.json'
data = json.loads(Path(file_path).read_text())
```

## Note

The libraries and dependencies are already preinstalled in the Simplilearn lab, so refrain from using the **pip install** command.

# Markdown Loader

Markdown is a simple markup language used to create formatted text using a plain-text editor.

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

from langchain.document_loaders import UnstructuredMarkdownLoader

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

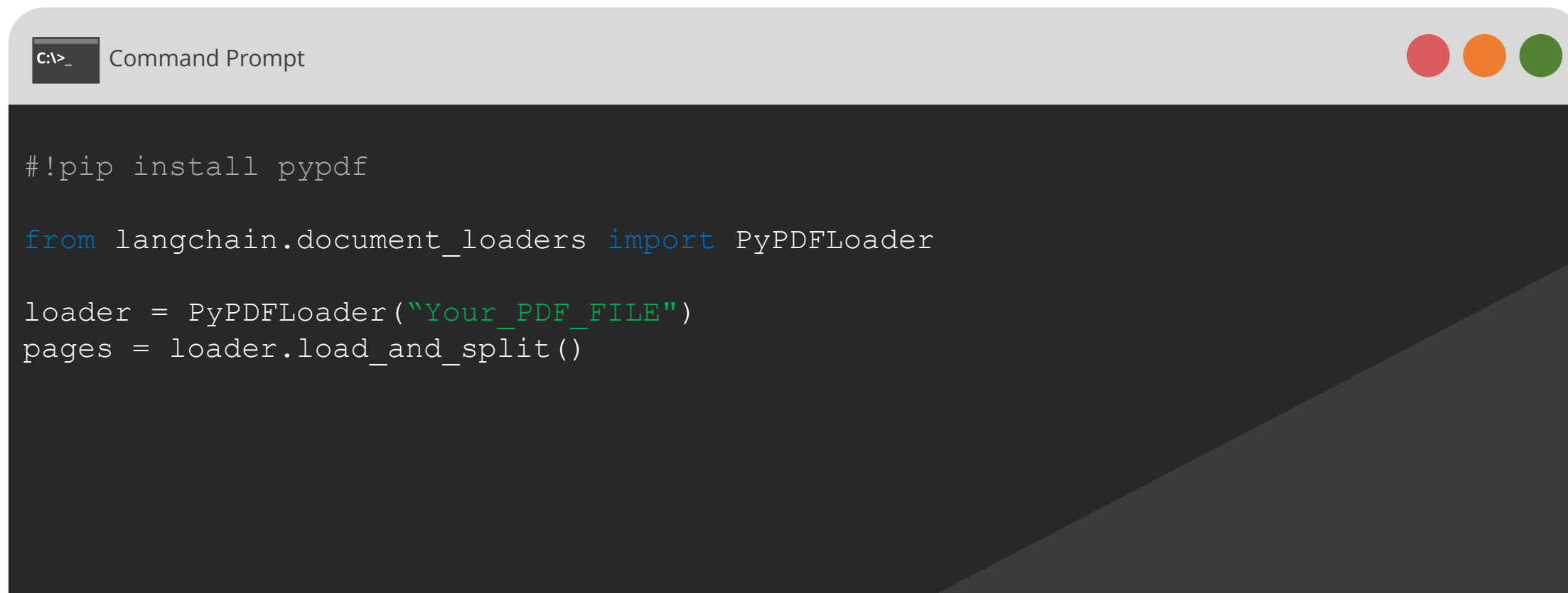
data = loader.load()
```

## Note

The libraries and dependencies are already preinstalled in the Simplilearn lab, so refrain from using the **pip install** command.

# PDF Document Loader

The PDF (portable document format) is a file format designed to display documents, including text formatting and images.



```
#!pip install pypdf

from langchain.document_loaders import PyPDFLoader

loader = PyPDFLoader("Your_PDF_FILE")
pages = loader.load_and_split()
```

## Note

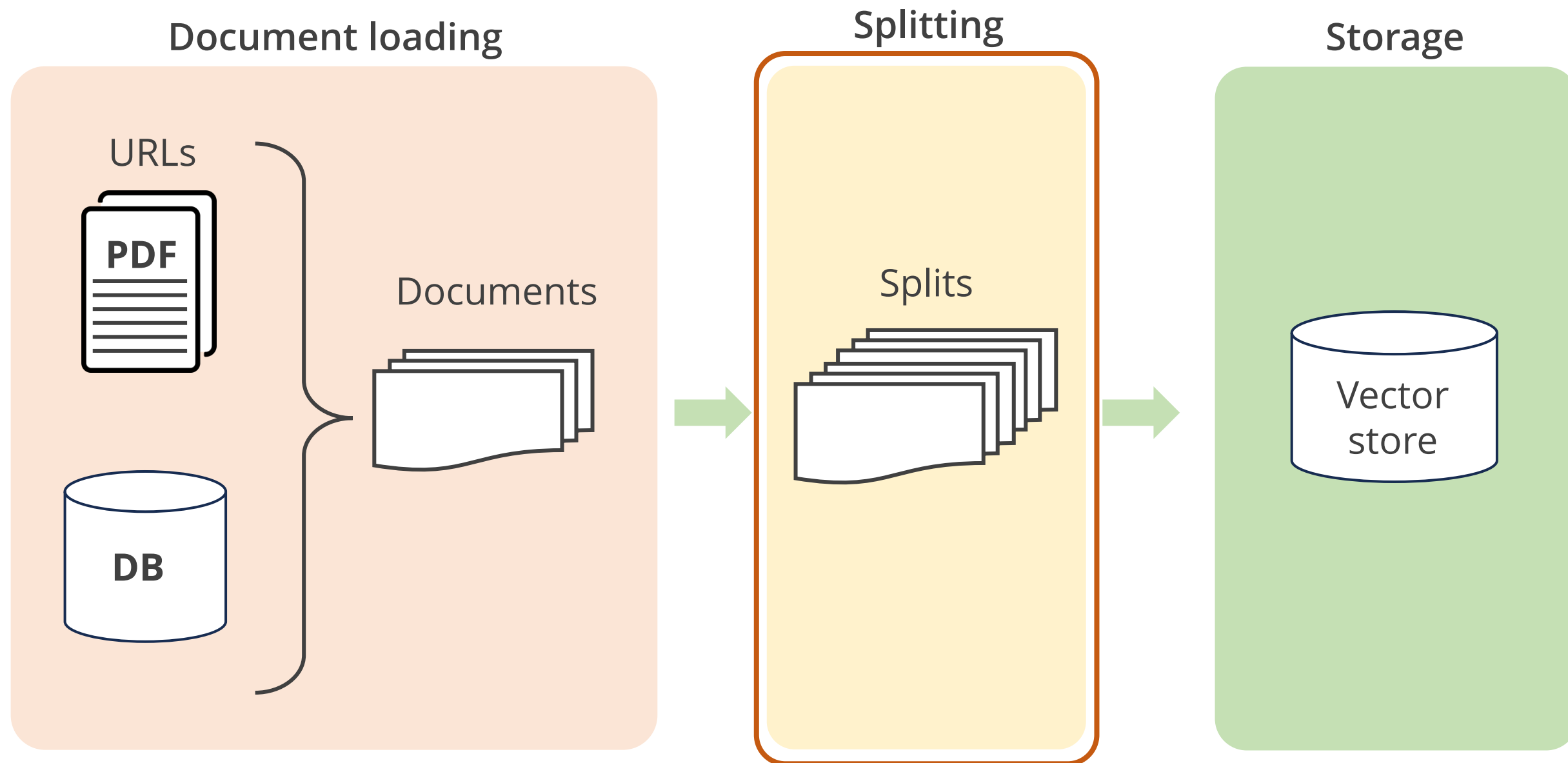
The libraries and dependencies are already preinstalled in the Simplilearn lab, so refrain from using the **pip install** command.



## Text Splitters

# Text Splitters

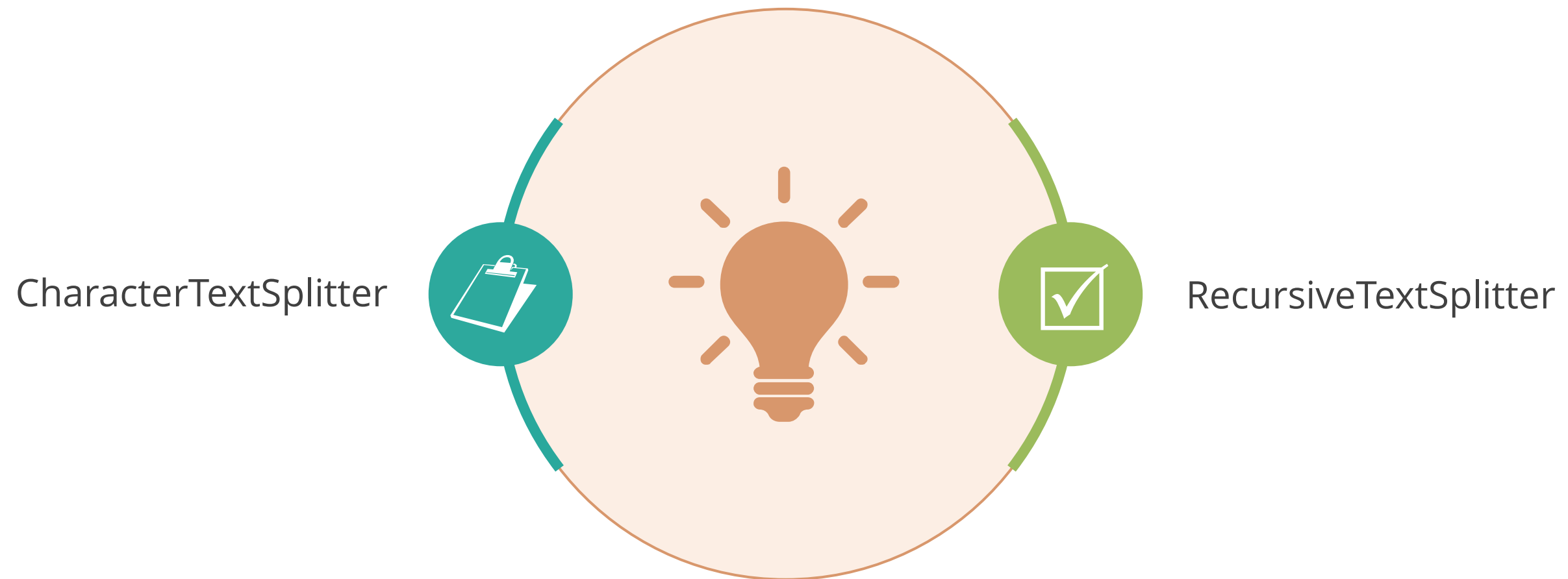
Initially, the documents must be loaded into LangChain. Then, it is essential to learn how to segment the text and store it in segments.





# Text Splitters

There are two major text splitters in LangChain:



# CharacterTextSplitter

The CharacterTextSplitter in LangChain divides large text into smaller sections using a defined set of separator characters to determine the chunk size.

The chunk size of the CharacterTextSplitter determines the maximum number of characters in each chunk of text.

# CharacterTextSplitter: Example

This code creates a CharacterTextSplitter object with a separator of `\n`.

C:\> Command Prompt

```
from langchain.text_splitter import CharacterTextSplitter

# Create a CharacterTextSplitter object
c_splitter = CharacterTextSplitter(separator="\n")

# It then splits the text into smaller chunks based on the
specified separator.
file_path = 'state_of_union.txt'
with open(file_path, 'r') as file:
    text = file.read()
chunks = c_splitter.split_text(text)

print(chunks)
```

- The CharacterTextSplitter exclusively splits based on the chosen separator.
- The chunk\_size parameter establishes an upper limit on the size of chunks generated through the split\_text method.

# CharacterTextSplitter: Example

Output of the CharacterTextSplitter example:

[ 'Madam Speaker, Madam Vice President, our First Lady and Second Gentleman. Members of Congress and the Cabinet. Justices of the Supreme Court. My fellow Americans. \nLast year COVID-19 kept us apart. This year we are finally together again. \nTonight, we meet as Democrats Republicans and Independents. But most importantly as Americans. \nWith a duty to one another to the American people to the Constitution. \nAnd with an unwavering resolve that freedom will always triumph over tyranny. \nSix days ago, Russia's Vladimir Putin sought to shake the foundations of the free world thinking he could make it bend to his menacing ways. But he badly miscalculated. \nHe thought he could roll into Ukraine and the world would roll over. Instead he met a wall of strength he never imagined. \nHe met the Ukrainian people. \nFrom President Zelenskyy to every Ukrainian, their fearlessness, their courage, their determination, inspires the world. \nGroups of citizens blocking tanks with their bodies. Everyone from students to retirees teachers turned soldiers defending their homeland. \nIn this struggle as President Zelenskyy said in his speech to the European Parliament Light will win over

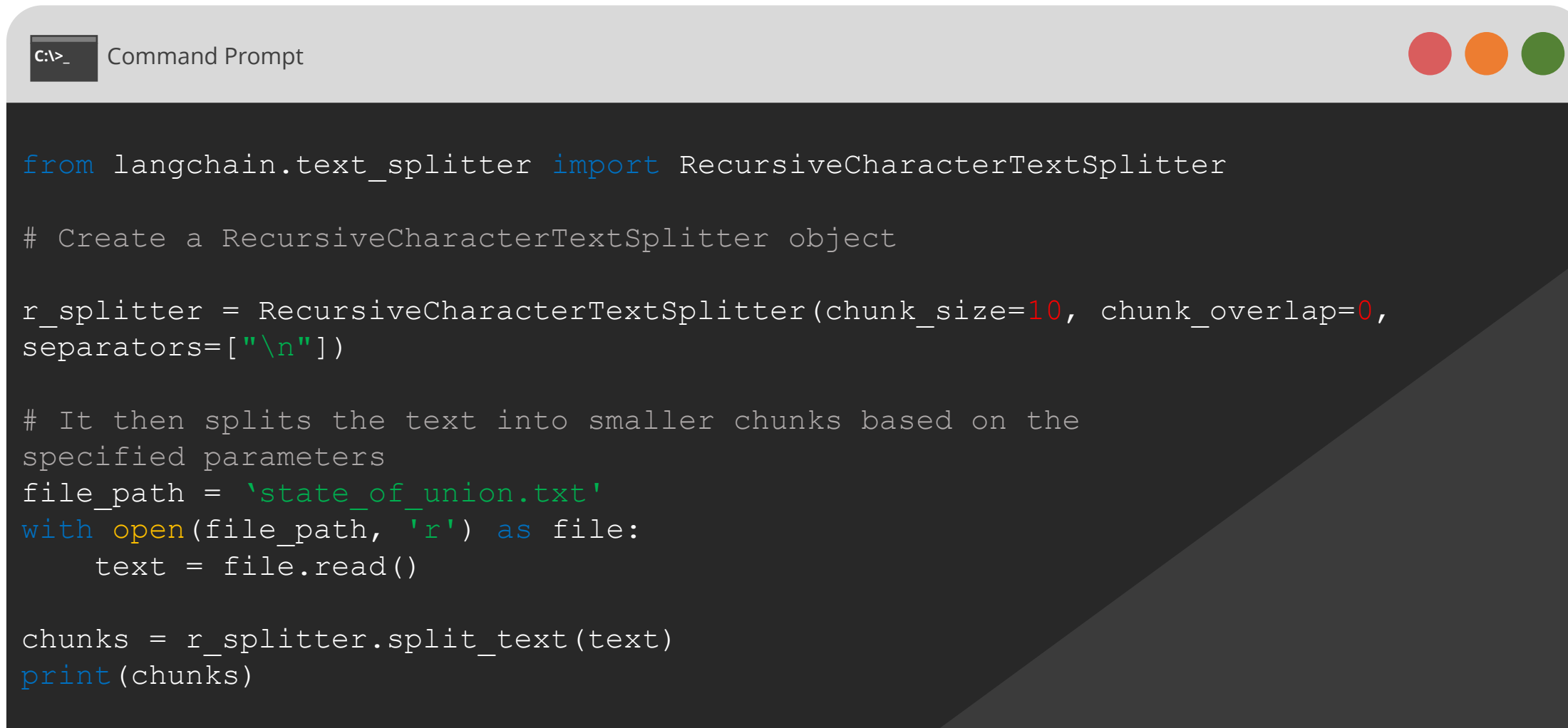
# RecursiveCharacterTextSplitter

RecursiveCharacterTextSplitter in LangChain recursively splits text into chunks for better handling, unlike CharacterTextSplitter, which has a fixed splitting approach.

There are different types of text splitters available in LangChain, and you can specify parameters such as chunk size and chunk overlap.

# RecursiveCharacterTextSplitter: Example

This code creates a RecursiveCharacterTextSplitter object with a chunk size of 10 and a chunk overlap of 0.



```
from langchain.text_splitter import RecursiveCharacterTextSplitter

# Create a RecursiveCharacterTextSplitter object

r_splitter = RecursiveCharacterTextSplitter(chunk_size=10, chunk_overlap=0,
separators=["\n"])

# It then splits the text into smaller chunks based on the
specified parameters
file_path = 'state_of_union.txt'
with open(file_path, 'r') as file:
    text = file.read()

chunks = r_splitter.split_text(text)
print(chunks)
```

# RecursiveCharacterTextSplitter: Example

Output of the RecursiveCharacterTextSplitter example:

```
[\"Madam Speaker, Madam Vice President, our First Lady and Second Gentleman. Members of Congress and the Cabinet. Justices of the Supreme Court. My fellow Americans. '\nLast year COVID-19 kept us apart. This year we are finally together again. '\nTonight, we meet as Democrats Republicans and Independents. But most importantly as Americans. '\nWith a duty to one another to the American people to the Constitution. '\nAnd with an unwavering resolve that freedom will always triumph over tyranny. '\nSix days ago, Russia's Vladimir Putin sought to shake the foundations of the free world thinking he could make it bend to his menacing ways. But he badly miscalculated. '\nHe thought he could roll into Ukraine and the world would roll over. Instead he met a wall of strength he never imagined. '\nHe met the Ukrainian people. '\nFrom President Zelenskyy to every Ukrainian, their fearlessness, their courage, their determination, inspires the world. '\nGroups of citizens blocking tanks with their bodies. Everyone from students to retirees teachers turned soldiers defending their homeland. '\nIn this struggle as President Zelenskyy said in his speech to the European
```

## Quick Check



What does the **CharacterTextSplitter** in LangChain do?

- A. It splits text based on a specified separator character.
- B. It loads all documents present in a directory.
- C. It loads HTML documents into a format that can be utilized in subsequent processes.
- D. It loads PDF documents into the Document format that you use downstream.

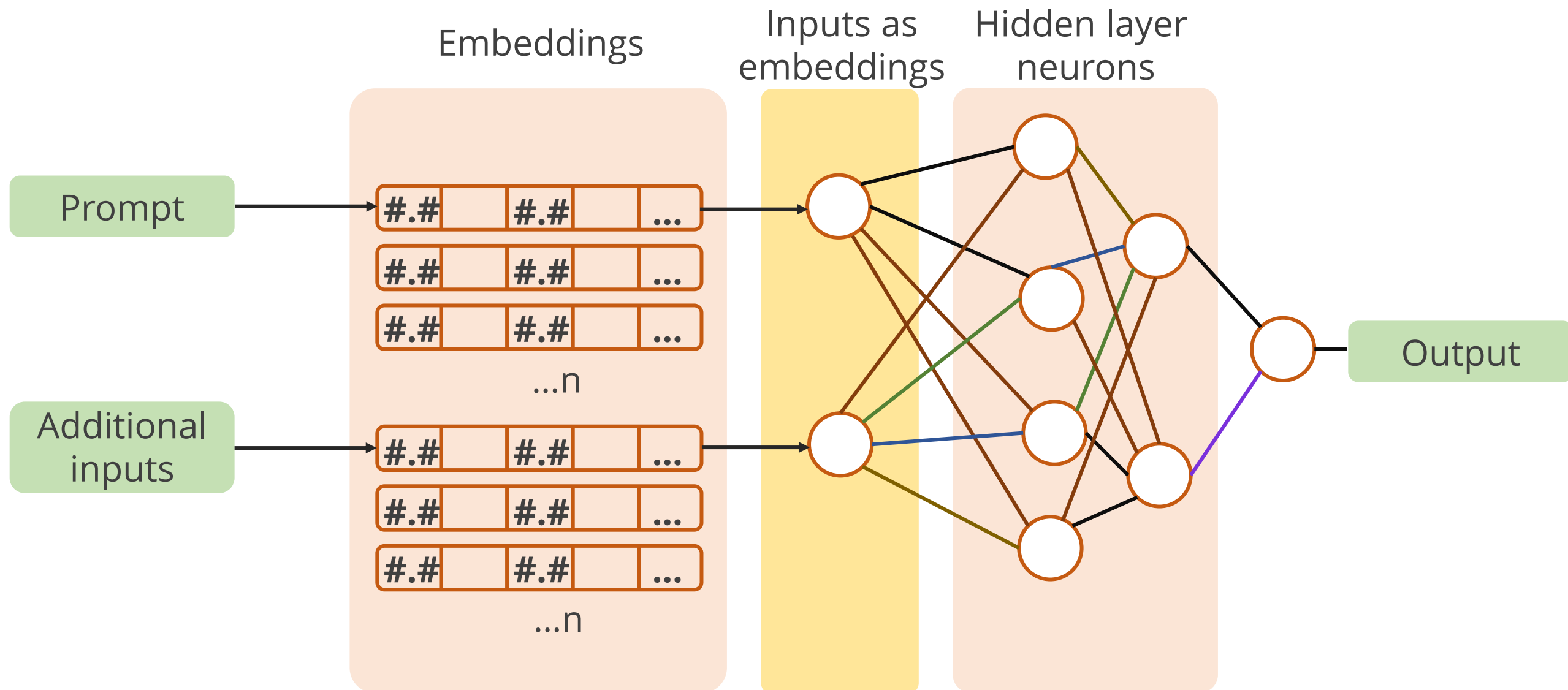




# Embeddings in GenAI

# Embeddings in GenAI

Before delving into converting these chunks into embeddings, first grasp the concept of embeddings in Generative AI.



# Embeddings in GenAI

In generative AI, embeddings transform complex data into compact vector representations. These vectors, typically floating-point numbers, encode the characteristics of the data.

Distances between vectors, calculated using metrics like cosine similarity or Euclidean distance, indicate their relatedness; shorter distances signify stronger correlations.

**For example:** In a text embedding, the words dog and puppy might have vectors with a small cosine similarity angle or Euclidean distance because they are closely related in meaning. Conversely, dog and car would have a larger distance, reflecting their lack of similarity.

Embeddings are a method of representing text, images, and other formats in a structured way that enables categorization and analysis. They allow AI systems to understand and organize data effectively, powering applications like text generation, language translation, and image recognition.

# Text Embedding Models in LangChain

Here are the key points about text embedding models in LangChain:

LangChain provides a standardized interface for various embedding model providers like OpenAI, Cohere, and Hugging Face.

These models transform text into vector representations, enabling operations like semantic search through text similarity in vector space.

To begin text embedding models, install the required packages and set up API keys. For OpenAI in LangChain, these steps are already complete.

The `embed_documents` method embeds multiple texts, providing a list of vector representations.

# Text Embedding Models Intuition

## Text embedding

For embedding a single text, such as a search query, the `embed_query` method is used. This is useful for comparing a query to a set of document embeddings.

## Text vectorization

Each piece of text is converted into a vector, the dimension of which depends on the model used.

## Model outputs

For instance, OpenAI models typically produce 1536-dimensional vectors. These embeddings are then used for retrieving relevant information.

# Text Embedding Models Intuition

## Flexible compatibility

LangChain's embedding functionality is not limited to OpenAI but is designed to work with various providers.

## Consistent concept

The setup and usage might slightly differ depending on the provider, but the core concept of embedding texts into vector space remains the same.

# Text Embeddings Model: Example

c:\>\_ Command Prompt

```
from langchain.embeddings import OpenAIEmbeddings

# Initialize the model
embeddings_model = OpenAIEmbeddings()

# Embed a list of texts
embeddings = embeddings_model.embed_documents(
    ["Hi there!", "Oh, hello!", "What's your name?",
     "My friends call me World", "Hello World!"]
)
print("Number of documents embedded:", len(embeddings))
print("Dimension of each embedding:", len(embeddings[0]))
```

Output:

Number of documents embedded: 5  
Dimension of each embedding: 1536

# Text Embeddings Model: Example

In this example, you first import the OpenAIEmbeddings class from the LangChain.embeddings module.

Initialize an instance of this class. Using the embed\_documents method of this instance, you embed a list of texts.

The method returns a list of vector representations (embeddings) for these texts.

In the end, you print the count of embedded documents and the dimension of each embedding, providing processing insights.



## Quick Check



What is the purpose of the `embed_documents` method in LangChain's text embedding models?

- A. It is used to split the text into smaller chunks.
- B. It is used to load all documents present in a directory.
- C. It is used to embed multiple texts, providing a list of vector representations.
- D. It is used to install specific packages and set up API keys.

# Guided Practice



## Overview

**Duration: 25 minutes**

In this exercise, you will gain hands-on experience loading documents, splitting them into manageable chunks, and embedding them into a numerical space. These are key steps in many natural language processing pipelines, including document classification, and information retrieval.

**Data:** state\_of\_union.txt, michael\_resume.pdf

### Note

Please download the solution document from the Reference Material Section and follow the Jupyter Notebook for step-by-step execution.

# Key Takeaways

- The interaction between the user, the model, and the output is crucial.
- The CharacterTextSplitter in LangChain divides large text into smaller sections using a defined set of separator characters to determine the chunk size.
- Embeddings convert complex data into compact vector forms. These vectors, usually floating-point numbers, encapsulate the characteristics of the dataset.



# Q&A

