Text-Splitters

▼ Split By HTMLheader:

Description :

- **Purpose:** To split HTML content at the header element level(h1 to h6)
- **Functionality:** It groups text chunks based on headers, aiming to maintain semantic context and preserve document structure.
- **Metadata:** Each chunk retains metadata about the headers relevant to it, helping to organize and understand the content hierarchy.

Usage Examples:

1. **With HTML String:**
   - You install the `langchain-text-splitters` library.
   - Define an `HTMLHeaderTextSplitter` with specific headers (`h1`, `h2`, `h3`).
   - Split an HTML string and retrieve chunks with associated metadata.
2. **Pipelined with Another Splitter:**
   - Load HTML from a web URL.
   - Use `HTMLHeaderTextSplitter` to split based on headers.
   - Further process the resulting chunks using a different splitter (`RecursiveCharacterTextSplitter` in this case) to handle large chunks of text.

API Reference:

- **HTMLHeaderTextSplitter:** Responsible for splitting HTML content based on specified header tags. Returns structured chunks with metadata.

Limitations:

- **Structural Variations:** It may miss headers if the document structure doesn't align with assumed hierarchical relationships (like parent-child

relationships of headers).

- **Example:** In news articles where headers aren't directly hierarchical

DEMO:

```
# HTML content from your file (replace this with your actual HTML content)
html_content = """
<!DOCTYPE html>
<html>
<body>
   <div>
    <h1>Hi</h1>
   </div>
</body>
</html>
"""
# Define headers to split on
headers_to_split_on = [
    ("h1", "Header 1"),
    ("h2", "Header 2"),
    ("h3", "Header 3"),
]


html_splitter = HTMLHeaderTextSplitter(headers_to_split_on=headers_to_split_on)
html_header_splits = html_splitter.split_text(html_content)
html_header_splits
```

```
[28]: [Document(page_content='Hi')]
```

if we remove the parent div :

```
from langchain_text_splitters import HTMLHeaderTextSplitter

html_string = """
<!DOCTYPE html>
<html>
<body>
        <h1>Hi</h1>
</body>
</html>
"""

headers_to_split_on = [
    ("h1", "Header 1"),
    ("h2", "Header 2"),
    ("h3", "Header 3"),
]

html_splitter = HTMLHeaderTextSplitter(headers_to_split_on=headers_to_split_on)
html_header_splits = html_splitter.split_text(html_string)
html_header_splits
```

```
[]
```

▼ Split by HTML section

Purpose:

The primary purpose of the `HTMLSectionSplitter` is to break down HTML content into semantically meaningful sections. Unlike the `HTMLHeaderTextSplitter`, which focuses mainly on headers (`h1`, `h2`, `h3`, etc.), the `HTMLSectionSplitter` can consider multiple structural cues beyond just headers to detect and segment sections of HTML content.

Functionality:

1. **Structure-Aware Chunking:**
    - The splitter operates at the element level within the HTML document, analyzing structural elements to determine section boundaries.
    - It can detect and segment content based on header tags (`h1`, `h2`, `h3`, etc.) as well as other structural elements like divs, spans, or sections.
2. **Metadata and Context Preservation:**
    - Each segmented section retains metadata that captures relevant structural information associated with that section.

- This metadata helps preserve context-rich information encoded within the document structure, aiding in further analysis or processing.

3. **XSLT Transformation Support:**
   - Optionally, the `HTMLSectionSplitter` can utilize XSLT transformations (`xslt_path`) to preprocess the HTML content.
   - This preprocessing step can enhance the detection of sections based on specific tags, attributes, or structural transformations applied to the HTML document.

Example :

```
from lxml import etree
from langchain_text_splitters import HTMLSectionSplitter

# Define headers and other structural elements to split on
headers_to_split_on = [
    ("h1", "Header 1"),
    ("h2", "Header 2"),
    ("h3", "Header 3"),
    ("h4", "Header 4"),
]

# Path to the XSLT file
xslt_path = "Downloads/convert_to_header.xslt"

# Example HTML content with <span> elements to be transforme
html_content = """
<!DOCTYPE html>
<html>
<head>
    <title>Example HTML Document</title>
</head>
<body>
    <span class="section-title">Introduction</span>
    <p>This section introduces the topic.</p>
```

```
    <span class="section-title">Methods</span>
    <p>Details about the experimental methods used.</p>

    <span class="section-title">Results</span>
    <p>Summary of the key results obtained.</p>

    <span class="section-title">Discussion</span>
    <p>Interpretation of results and implications.</p>
</body>
</html>
"""

# Apply XSLT transformation to preprocess HTML content
xslt = etree.parse(xslt_path)
transform = etree.XSLT(xslt)
transformed_html = transform(etree.fromstring(html_content))

# Initialize HTMLSectionSplitter with the XSLT path and heade
html_splitter = HTMLSectionSplitter(headers_to_split_on=heade

# Convert the transformed HTML back to a string
transformed_html_string = etree.tostring(transformed_html, pr

# Split transformed HTML content into sections
html_sections = html_splitter.split_text(transformed_html_str

# Print out each section and its metadata
for section in html_sections:
    print(f"Section: {section.page_content}")
    print(f"Metadata: {section.metadata}")
    print("--------------------")
```

The Output:

```
Section: Introduction
 This section introduces the topic.
Metadata: {'Header 1': 'Introduction'}
--------------------
Section: Methods
 Details about the experimental methods used.
Metadata: {'Header 1': 'Methods'}
--------------------
Section: Results
 Summary of the key results obtained.
Metadata: {'Header 1': 'Results'}
--------------------
Section: Discussion
 Interpretation of results and implications.
Metadata: {'Header 1': 'Discussion'}
--------------------
```

Explanation

1. **XSLT Transformation**:
   - The XSLT file converts `<span>` elements with the class `section-title` into `<h1>` headers.
   - This preprocessing step ensures that the HTML content has a consistent structure with headers that can be detected by the `HTMLSectionSplitter`.
2. **HTMLSectionSplitter**:
   - The splitter is initialized with the headers to split on and the path to the XSLT file.
   - The transformed HTML content is split into sections based on the specified headers.
   - Each section is printed with its content and metadata.

This example demonstrates how to use XSLT to preprocess HTML content and then use `HTMLSectionSplitter` to segment the content into meaningful sections.

▼ CharacterTextSplitter

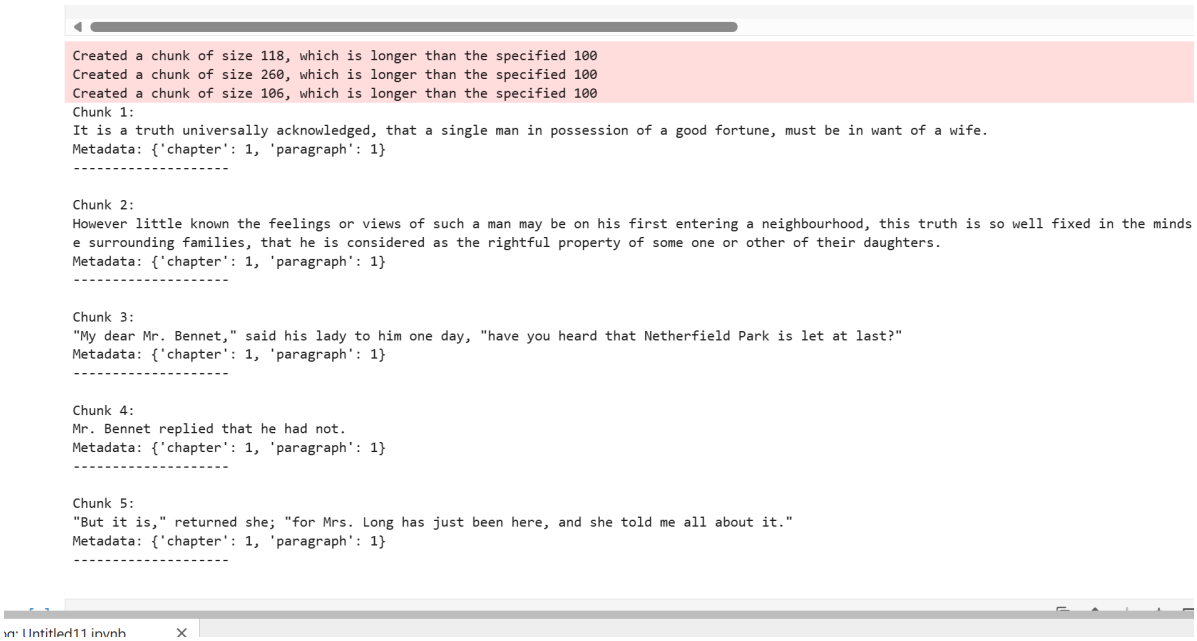
The `CharacterTextSplitter` is a simple yet effective text splitter that divides text based on a specified character separator. It measures chunk sizes by the number of characters, allowing for easy control over the size and overlap of chunks.

Key Features:

- **Splitting Method**: Splits text based on a single character or sequence of characters (e.g., "\n\n").
- **Chunk Size Measurement**: Measures the chunk size by the number of characters.
- **Customizable Parameters**: Allows customization of the separator, chunk size, chunk overlap, and length measurement function.

Example:

```
novel_excerpt = """
It is a truth universally acknowledged, that a single man in possession of a good fortune, must be in want of a wife.

However little known the feelings or views of such a man may be on his first entering a neighbourhood, this truth is so well fixed in the

"My dear Mr. Bennet," said his lady to him one day, "have you heard that Netherfield Park is let at last?"

Mr. Bennet replied that he had not.

"But it is," returned she; "for Mrs. Long has just been here, and she told me all about it."
"""

# Step 3: Import and Configure CharacterTextSplitter
from langchain_text_splitters import CharacterTextSplitter

text_splitter = CharacterTextSplitter(
    separator="\n\n",  # Split on double newline
    chunk_size=100,    # Desired chunk size in characters
    chunk_overlap=20,  # Number of overlapping characters between chunks
    length_function=len,  # Function to measure chunk length (default is len)
    is_separator_regex=False  # Whether the separator is a regex pattern
)

# Step 4: Split the Document and Add Metadata
metadata = [{"chapter": 1, "paragraph": i + 1} for i in range(len(novel_excerpt.split("\n\n")))]
documents = text_splitter.create_documents([novel_excerpt], metadatas=metadata)

# Step 5: Print the Split Chunks with Metadata
for i, doc in enumerate(documents):
    print(f"Chunk {i + 1}:\n{doc.page_content}\nMetadata: {doc.metadata}\n{'-' * 20}\n")
```



▼ SplitCode

The `SplitCode` functionality in LangChain's `RecursiveCharacterTextSplitter` allows you to effectively split code blocks based on the syntax rules of various programming languages. Here's how it works and what you can expect:

How it Works

1. **Language Support**: LangChain supports a variety of programming languages such as Python, JavaScript, TypeScript, Markdown, LaTeX, HTML, Solidity, C#, Haskell, PHP, and more. Each language has specific rules and separators that define where code blocks can logically be split.
2. **API Usage**:
   - Import the necessary components from `langchain_text_splitters`.
   - Specify the language using the `Language` enum.
   - Use `RecursiveCharacterTextSplitter.from_language()` to initialize a text splitter for a specific language.
   - Specify parameters like `chunk_size` and `chunk_overlap` to control how the code is split into documents.
3. **Example**:
   - For Python:

```
python_splitter = RecursiveCharacterTextSplitter.fro
m_language(
    language=Language.PYTHON, chunk_size=50, chunk_o
verlap=0
)
python_docs = python_splitter.create_documents([PYTH
ON_CODE])
```

This would split Python code (`PYTHON_CODE`) into documents based on Python-specific separators.

- Similar approaches apply to other languages supported by LangChain.

Benefits

- **Accurate Splitting**: LangChain leverages language-specific rules to accurately split code, ensuring that each resulting document contains coherent code segments.
- **Integration**: It seamlessly integrates with various programming languages commonly used in software development, making it versatile for projects with multi-language codebases.

▼ MarkdownHeaderTextSplitter

The MarkdownHeaderTextSplitter is a tool used to split a markdown file into chunks based on specified headers. This tool is useful when you want to preserve the structure of the document while splitting it into smaller chunks. For example, if you have a markdown file with multiple sections, you can use MarkdownHeaderTextSplitter to split the file into chunks based on the section headers.

key features and functionality:

1. **Header-Based Splitting**: Markdown documents are structured with headers denoted by prefixes like `#`, `##`, `###`, etc. The splitter allows you to specify which headers to use for splitting. For instance, you might choose

to split on all levels of headers (`#`, `##`, `###`), or only on specific levels depending on your document's structure.

2. **Metadata Preservation**: Each split section retains metadata that indicates which headers were used to define that section. This metadata is useful for understanding the hierarchical structure of the original Markdown document.
3. **Example Usage**:
    - You initialize the `MarkdownHeaderTextSplitter` with a list of tuples defining the headers to split on (e.g., `[("#", "Header 1"), ("##", "Header 2"), ("###", "Header 3")]`).
    - The splitter then processes a Markdown document, splitting it into segments wherever the specified headers appear.
    - Each segment is encapsulated in a `Document` object that includes:
        - `page_content`: The actual content of the section.
        - `metadata`: A dictionary that maps header levels to their respective titles as defined in the Markdown document.
4. **Customization**:
    - By default, the splitter removes the headers being split on from the content of each section (`strip_headers=True`). This behavior can be adjusted if you prefer to retain headers (`strip_headers=False`).
5. **Integration**:
    - The splitter is integrated into the LangChain framework, making it suitable for use alongside other text processing tools and workflows.
6. **Application**:
    - Use cases include processing Markdown files for content extraction, analysis, or transformation into other formats where preserving the hierarchical structure defined by headers is essential

Example:

```
from langchain_text_splitters import MarkdownHeaderTextSplit
from langchain.schema.document import Document

# Example Markdown document
markdown_document = """
# Introduction

## Purpose

The purpose of this document is to demonstrate the usage of M

## Features

- Splits Markdown based on headers.
- Retains header information in metadata.

### Example Section

This is an example section within the Markdown document.

## Conclusion

In conclusion, MarkdownHeaderTextSplitter is useful for organ
"""

# Headers to split on
headers_to_split_on = [("#", "Header 1"), ("##", "Header 2"),

# Initialize MarkdownHeaderTextSplitter
markdown_splitter = MarkdownHeaderTextSplitter(headers_to_spl

# Split Markdown document into chunks with headers
md_header_splits = markdown_splitter.split_text(markdown_docu

# Display the results
```

```
for idx, split in enumerate(md_header_splits):
    print(f"Split {idx + 1}:")
    print(f"Content:\n{split.page_content}\n")
    print(f"Metadata:\n{split.metadata}\n")
```

The Output:

```
Split 1:
Content:
The purpose of this document is to demonstrate the usage of M

Metadata:
{'Header 1': 'Introduction', 'Header 2': 'Purpose'}

Split 2:
Content:
- Splits Markdown based on headers.
- Retains header information in metadata.

Metadata:
{'Header 1': 'Introduction', 'Header 2': 'Features'}

Split 3:
Content:
This is an example section within the Markdown document.

Metadata:
{'Header 1': 'Introduction', 'Header 2': 'Features', 'Header

Split 4:
Content:
In conclusion, MarkdownHeaderTextSplitter is useful for organ

Metadata:
{'Header 1': 'Introduction', 'Header 2': 'Conclusion'}
```