
- 출처: LangChain 공식 문서 또는 해당 교재명
- 원본 URL: https://smith.langchain.com/hub/teddynote/summary-stuff-documents

---

코드 분할 (Python, Markdown, JAVA, C++, C#, GO, JS, Latex 등)

Split code

- **CodeTextSplitter 사용 → 다양한 프로그래밍 언어로 작성된 코드 분할 가능**
- **Language enum → import → 해당하는 프로그래밍 언어 지정**

- 사전에 VS Code 터미널에 설치할 것

```
pip install -qU langchain-text-splitters
```

---

1) RecursiveCharacterTextSplitter

- **langchain_text_splitters 모듈: Language와 RecursiveCharacterTextSplitter 클래스 임포트**
- **RecursiveCharacterTextSplitter = 텍스트를 문자 단위로 재귀적으로 분할하는 텍스트 분할기**

```
from langchain_text_splitters import (
    Language,
    RecursiveCharacterTextSplitter,
)
```

- **지원되는 언어의 전체 목록** 가져오기

```
# 지원되는 언어의 전체 목록 가져오기

[e.value for e in Language]
```

- 셀 출력

```
['cpp', 'go', 'java', 'kotlin', 'js', 'ts', 'php', 'proto', 'python', 'rst', 'ruby', 'rust', 'scala', 'swift', 'm
```

- **RecursiveCharacterTextSplitter 클래스의 get_separators_for_language 메서드 사용 → 특정 언어에 사용되는 구분자 (separators)를 확인 가능**
  - 예시: Langauge.PYTHON 열거형 값을 인자로 전달 → Python 언어에 사용되는 구분자 확인

```
# 주어진 언어에 대해 사용되는 구분자 확인 가능

RecursiveCharacterTextSplitter.get_separators_for_language(Language.PYTHON)
```

- 셀 출력

```
['\nclass ', '\ndef ', '\n\tdef ', '\n\n', '\n', ' ', '']
```

---

2) Python

- **`RecursiveCharacterTextSplitter` 사용한 예제**
  - `RecursiveCharacterTextSplitter` → `Python 코드`를 `문서` 단위로 `분할`
  - `language` 매개변수에 `Language.PYTHON`을 지정 → `Python 언어`를 `사용`
  - `chunk_size` = `50` → `각 문서의 최대 크기 제한`
  - `chunk_overlap` = `0` → `문서 간의 중복 허용 X`

```
PYTHON_CODE = """
def hello_world():
    print("Hello, World!")

hello_world()
"""

# RecursiveCharacterTextSplitter.from_language() 사용하기
python_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.PYTHON,               # 언어 = Python
    chunk_size=50,                          # 각 문서 최대 크기 = 50
    chunk_overlap=0                         # 문서 간 중복 X
)
```

- `Document` 생성: `Document` = `리스트 형태`로 반환

```
python_docs = python_splitter.create_documents([PYTHON_CODE])

python_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='def hello_world():\n    print("Hello, World!")'),
Document(metadata={}, page_content='hello_world()')]
```

```
print(type(python_docs))                # <class 'list'>
```

```
# 출력해보기

for doc in python_docs:
    print(doc.page_content, end="\n==================\n")
```

- 셀 출력

```
def hello_world():
    print("Hello, World!")
==================
hello_world()
==================
```

---

3) `JS`

- **`JS` 텍스트 분할기를 사용한 예시**

```
JS_CODE = """
function helloWorld() {
    console.log("Hello, World!");
}

helloWorld();
"""

# RecursiveCharacterTextSplitter.from_language() 사용하기
js_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.JS,                   # 언어 = JS
    chunk_size=60,                          # 최대 문서 크기 = 60
    chunk_overlap=0                         # 문서 간 중복 허용 X
)

# 문서로 만들어 출력해보기
js_docs = js_splitter.create_documents([JS_CODE])
js_docs
```

- 셀 출력 (참고: js_docs의 type = <class `list`>)

```
[Document(metadata={}, page_content='function helloWorld() {\n    console.log("Hello, World!");\n}'),
Document(metadata={}, page_content='helloWorld();')]
```

4) `TS`

- **`TS` 텍스트 분할기를 사용한 예시**

- `TS` = `TypeScript`
  - `JavaScript`의 `상위 집합` 언어: `JavaScript`와 호환, 더 많은 기능 제공
  - `JavaScript`에 **정적 타입(`static typing`) 추가한 언어 → `웹 개발`에서 주로 사용**
  - 주요 사용 용도: **`웹 애플리케이션`, `모바일 애플리케이션`, `데스크톱 애플리케이션` 등 개발**

```
TS_CODE = """
function helloWorld(): void {
    console.log("Hello, World!");
}

helloWorld();
"""

ts_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.TS,
    chunk_size=60,
    chunk_overlap=0
)

# 문서로 만들어 출력하기
ts_docs = ts_splitter.create_documents([TS_CODE])
ts_docs
```

- 셀 출력 (참고: ts_docs의 type = <class `list`>)

```
[Document(metadata={}, page_content='function helloWorld(): void {'),
Document(metadata={}, page_content='console.log("Hello, World!");\n}'),
Document(metadata={}, page_content='helloWorld();')]
```

---

5) `Markdown`

- `Markdown` 텍스트 분할기를 사용한 예시

```
markdown_text = """
# 🦜🔗 LangChain

⚡ LLM을 활용한 초스피드 애플리케이션 구축 ⚡

## 빠른 설치
```

```bash
pip install langchain

# 빠르게 발전하는 분야의 오픈 소스 프로젝트 입니다. 많관부 🙏
"""
```

```
print(type(markdown_text))              # <class 'str'>
```

- 분할 및 결과 출력하기

```
# 마크다운 언어를 사용하여 텍스트 분할기 생성
md_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.MARKDOWN,                 # 언어 = md
    chunk_size=60,
    chunk_overlap=0,
)

# 마크다운 텍스트를 분할하여 문서 생성
md_docs = md_splitter.create_documents([markdown_text])

# 생성된 문서 출력
md_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='# 🦜🔗 LangChain\n\n⚡ LLM을 활용한 초스피드 애플리케이션 구축 ⚡'),
Document(metadata={}, page_content='## 빠른 설치\n\n```bash\npip install langchain'),
Document(metadata={}, page_content='# 빠르게 발전하는 분야의 오픈 소스 프로젝트 입니다. 많관부 🙏')]
```

---

6) `Latex`

- `LaTeX`
  - **`문서 작성을 위한 마크업 언어`**
  - `수학 기호`와 `수식`을 표현하는 데 널리 사용

- 주의! - **`raw string`**(`r""`) **사용하기**
  - 목적: `이스케이프 시퀀스`를 `일반 문자`로 `처리`하기 위해 사용
  - 특징
    - `\` = 일반 문자로 취급 ≠ 이스케이프 문자
    - 예시: **`r"\n"`** → **`문자열 \n`** **(줄바꿈 문자 아님)**
    - 경고 메시지

```
<>:2: SyntaxWarning: invalid escape sequence '\d'
<>:2: SyntaxWarning: invalid escape sequence '\d'
/var/folders/h3/l7wnkv352kqftv0t8ctl2ld40000gn/T/ipykernel_48710/1540502022.py:2: SyntaxWarning: invalid escape se
\documentclass{article}
```

* * 경고 메시지 해석 * `SyntaxWarning: invalid escape sequence '\d'` = `LaTeX` 문서에서 `\d`를 `이스케이프 시퀀스`로 인식하지 못해서 발생하는 경고 메시지 * `\d` = `LaTeX`에서 숫자를 나타내는 명령어 = `Python 문자열`에서는 **`이스케이프 시퀀스`로 인식**

```
* 해결 방법: `\d`를 `이스케이프 시퀀스`로 인식하지 않도록 처리 → **`r 접두사` 추가 = `raw string`로 처리**
```

- 동작: `문자열`을 그대로 해석
- 사용하는 이유
  - **`LaTeX` 문서에서 `\`는 특별한 명령어를 나타내는 데 사용하기 때문에 구분이 필요**
    - 예시: **`\section` = 섹션을 만드는 명령어**

```
# LaTeX 텍스트 예시
latex_text = r"""
\documentclass{article}

\begin{document}
```

```
\maketitle

\section{Introduction}
% LLM은 방대한 양의 텍스트 데이터로 학습하여 사람과 유사한 언어를 생성할 수 있는 기계 학습 모델의 한 유형입니다.
% 최근 몇 년 동안 LLM은 언어 번역, 텍스트 생성, 감성 분석 등 다양한 자연어 처리 작업에서 상당한 발전을 이루었습니다.

\subsection{History of LLMs}
% 초기 LLM은 1980년대와 1990년대에 개발되었지만, 처리할 수 있는 데이터 양과 당시 사용 가능한 컴퓨팅 능력으로 인해 제한되었습니다.
% 그러나 지난 10년 동안 하드웨어와 소프트웨어의 발전으로 대규모 데이터 세트에 대해 LLM을 학습시킬 수 있게 되었고, 이는 성능의 큰 향상으로 이어졌습니다.

\subsection{Applications of LLMs}
% LLM은 챗봇, 콘텐츠 생성, 가상 어시스턴트 등 산업 분야에서 많은 응용 분야를 가지고 있습니다.
% 또한 언어학, 심리학, 컴퓨터 언어학 연구를 위해 학계에서도 사용될 수 있습니다.

\end{document}
"""
```

```
print(type(latex_text))                # <class 'str'>
```

- 분할 및 결과 출력하기

```
latex_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.LATEX,
    chunk_size=60,

    chunk_overlap=0,
)

# latex_text를 분할하여 문서 목록 생성하기
latex_docs = latex_splitter.create_documents([latex_text])

# 생성된 문서 목록 출력하기
latex_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='\\documentclass{article}\n\n\\begin{document}\n\n\\maketitle'),
    Document(metadata={}, page_content='\\section{Introduction}\n% LLM은 방대한 양의 텍스트 데이터로 학습하여 사람과 유사한'),
    Document(metadata={}, page_content='언어를 생성할 수 있는 기계 학습 모델의 한 유형입니다.\n% 최근 몇 년 동안 LLM은 언어 번역, 텍
    Document(metadata={}, page_content='생성, 감성 분석 등 다양한 자연어 처리 작업에서 상당한 발전을 이루었습니다.'),
    Document(metadata={}, page_content='\\subsection{History of LLMs}\n% 초기 LLM은 1980년대와 1990년대에'),
    Document(metadata={}, page_content='개발되었지만, 처리할 수 있는 데이터 양과 당시 사용 가능한 컴퓨팅 능력으로 인해 제한되었습니다.\
    Document(metadata={}, page_content='그러나 지난 10년 동안 하드웨어와 소프트웨어의 발전으로 대규모 데이터 세트에 대해 LLM을 학습시킬
    Document(metadata={}, page_content='있게 되었고, 이는 성능의 큰 향상으로 이어졌습니다.'),
    Document(metadata={}, page_content='\\subsection{Applications of LLMs}\n% LLM은 챗봇, 콘텐츠 생성, 가상'),
    Document(metadata={}, page_content='어시스턴트 등 산업 분야에서 많은 응용 분야를 가지고 있습니다.\n% 또한 언어학, 심리학, 컴퓨터 언
    Document(metadata={}, page_content='연구를 위해 학계에서도 사용될 수 있습니다.\n\n\\end{document}')]
```

---

7) `HTML`

- **`HTML` 텍스트 분할기를 사용한 예제**

```
html_text = """
<!DOCTYPE html>
<html>
    <head>
        <title>🦜🔗 LangChain</title>
        <style>
            body {
                font-family: Arial, sans-serif;
            }
            h1 {
                color: darkblue;
            }
        </style>
    </head>
    <body>
        <div>
            <h1>🦜🔗 LangChain</h1>
            <p>⚡ Building applications with LLMs through composability ⚡</p>
        </div>
```

```
        <div>
            As an open-source project in a rapidly developing field, we are extremely open to contributions.
        </div>
    </body>
</html>
"""
```

- 분할 및 결과 출력하기

```
# HTML 언어를 사용하여 텍스트 분할기 생성
html_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.HTML,
    chunk_size=60,
    chunk_overlap=0,
)

# 주어진 HTML 텍스트를 분할하여 문서 생성
html_docs = html_splitter.create_documents([html_text])

# 생성된 문서 출력
html_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='<!DOCTYPE html>\n<html>'),
Document(metadata={}, page_content='<head>\n        <title>🦜🔗 LangChain</title>'),
Document(metadata={}, page_content='<style>\n            body {\n                font-family: Aria'),
Document(metadata={}, page_content='l, sans-serif;  \n            }\n            h1 {'),
Document(metadata={}, page_content='color: darkblue;\n            }\n        </style>\n    </he'),
Document(metadata={}, page_content='ad>'),
Document(metadata={}, page_content='<body>'),
Document(metadata={}, page_content='<div>\n            <h1>🦜🔗 LangChain</h1>'),
Document(metadata={}, page_content='<p>⚡ Building applications with LLMs through composability ⚡'),
Document(metadata={}, page_content='</p>  \n        </div>'),
Document(metadata={}, page_content='<div>\n            As an open-source project in a rapidly dev'),
Document(metadata={}, page_content='eloping field, we are extremely open to contributions.'),
Document(metadata={}, page_content='</div>\n    </body>\n</html>')]
```

---

8) `Solidity`

- `Solidity`
  - **`이더리움 블록체인`에서 `스마트 계약`을 작성하기 위해 `사용`되는 `프로그래밍 언어`**
  - **`튜링 완전한 언어`**
    - 이론상 `모든 계산 가능한 문제를 해결할 수 있는 능력`을 가진 프로그래밍 언어
    - = `복잡한 알고리즘`이나 `프로그램`을 `작성`할 수 있는 `강력한 기능`을 가지고 있음
    - 이더리움의 스마트 계약은 튜링 완전한 언어로 작성 → **`다양한 금융`, `게임`, `투표` 등 `복잡한 시스템` 구현 가능**
  - **`이더리움 가상 머신`(`EVM`)에서 실행**

- `Solidity` 텍스트 분할기 사용한 예제
  - **`Solidity` 코드** = 문자열 형태 → **`SOL_CODE` 변수에 저장**
  - **`RecursiveCharacterTextSplitter`** 사용 → `Solidity` 코드를 `청크 단위`로 `분할`하는 **`sol_splitter` 생성하기**
    - `language` 매개변수 = **`Language.SOL`로 설정 → `Solidity` 언어 지정**
    - **`chunk_size = 128`** → 각 청크의 최대 크기 지정
    - **`chunk_overlap = 0`** → 청크 간의 중복 X
  - `sol_splitter.create_documents()` 메서드 사용 → `SOL_CODE`를 `청크 단위`로 `분할` → 분할된 청크를 `sol_docs` 변수에 `저장`
  - `sol_docs`를 `출력`하여 분할된 `Solidity` 코드 청크를 확인

```
SOL_CODE = """
pragma solidity ^0.8.20;
contract HelloWorld {
    function add(uint a, uint b) pure public returns(uint) {
        return a + b;
```

```
    }
}
"""
```

- 분할 및 결과 출력하기

```
sol_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.SOL,
    chunk_size=128,
    chunk_overlap=0
)

sol_docs = sol_splitter.create_documents([SOL_CODE])
sol_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='pragma solidity ^0.8.20;'),
Document(metadata={}, page_content='contract HelloWorld {  \n    function add(uint a, uint b) pure public returns
```

---

8) `C#`

- **`C#`**
  - 2000년 7월 MS에서 개발한 프로그래밍 언어
  - **`.NET` 프레임워크의 핵심 언어로 설계 → 객체 지향 프로그래밍, 자동 메모리 관리를 지원하는 현대적인 언어**
- 특징: **`OOP(객체 지향 프로그래밍)` 지원**, 강력한 타입 시스템 및 메모리 관리 제공
- 주요 사용 분야
  - **`MS`의 `.NET` 플랫폼에서 주로 사용 → `Windows` 애플리케이션 개발에 최적화**
  - 게임 개발(`Unity 엔진`), 클라우드 서비스(`Azure`), 엔터프라이즈 소프트웨어 등에 활용
- 만들어진 이유
  - **간결성** : `C/C++`의 복잡한 문법 단순화 → 개발 생산성 ↑
  - **객체 지향** : 클래스, 상속, 다형성 등 `OOP`를 기본으로 지원 → 복잡한 시스템 구현 용이성 ↑
  - **통합성**
    - `.NET` 프레임워크와 완벽하게 통합 → `Windows` 애플리케이션 개발 혁신 ↑ (**`MS 생태계와의 호환성 ↑`**)
    - **`크로스 플랫폼 개발 가능`** → 모바일(`Android, iOS`), 웹, 데스크톱 애플리케이션 모두 지원
- **`C#` vs `C/C++` 차이점**
  - 문법 : `C#`은 `C/C++`보다 **`간결`하고 `직관적`인 `문법`을 제공**
  - 메모리 관리 : `C#`은 **`자동 가비지 컬렉션 지원` → `메모리 관리`가 더 쉬움**
  - 플랫폼 : `C#`은 `.NET 프레임워크` 위에서 실행 → **`크로스 플랫폼 개발 가능`**

```
C_CODE = """
using System;
class Program
{
    static void Main()
    {
        Console.WriteLine("Enter a number (1-5):");
        int input = Convert.ToInt32(Console.ReadLine());
        for (int i = 1; i <= input; i++)
        {
            if (i % 2 == 0)
            {
                Console.WriteLine($"{i} is even.");
            }
            else
            {
                Console.WriteLine($"{i} is odd.");
            }
        }
        Console.WriteLine("Goodbye!");
    }
}
"""
```

- 분할 및 결과 출력하기

```
c_splitter = RecursiveCharacterTextSplitter.from_language(
    language=Language.CSHARP, chunk_size=128, chunk_overlap=0
)
c_docs = c_splitter.create_documents([C_CODE])
c_docs
```

- 셀 출력

```
[Document(metadata={}, page_content='using System;'),
Document(metadata={}, page_content='class Program\n{\n    static void Main()\n    {\n        Console.WriteLine("E
Document(metadata={}, page_content='int input = Convert.ToInt32(Console.ReadLine());\n        for (int i = 1; i <
Document(metadata={}, page_content='if (i % 2 == 0)\n            {\n                Console.WriteLine($"{i} is ev
Document(metadata={}, page_content='{\n                Console.WriteLine($"{i} is odd.");\n            }\n
Document(metadata={}, page_content='}\n}')]
```

- *next:* ***`마크다운 헤더 텍스트 분할 (MarkdownHeaderTextSplitter)`***
