# Split code

CodeTextSplitter allows you to split your code with multiple languages supported. Import enum `Language` and specify the language.

```
from langchain.text_splitter import (
    Language,
    RecursiveCharacterTextSplitter,
)
```

```
# Full list of supported languages
[e.value for e in Language]
```

```
['cpp',
 'go',
 'java',
 'kotlin',
 'js',
 'ts',
 'php',
 'proto',
```

```
 'python',
 'rst',
 'ruby',
 'rust',
 'scala',
 'swift',
 'markdown',
 'latex',
 'html',
 'sol',
 'csharp',
 'cobol']
```

```
# You can also see the separators used for a gi
RecursiveCharacterTextSplitter.get_separators_f
```

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

# Python

Here’s an example using the PythonTextSplitter:

```
PYTHON_CODE = """
def hello_world():
```

```
    print("Hello, World!")

# Call the function
hello_world()
"""
python_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.PYTHON, chunk_size=50,
chunk_overlap=0
)
python_docs =
python_splitter.create_documents([PYTHON_CODE])
python_docs
```

```
[Document(page_content='def hello_world():\n
print("Hello, World!")'),
 Document(page_content='# Call the
function\nhello_world()')]
```

# JS

Here's an example using the JS text splitter:

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

```
// Call the function
helloWorld();
"""

js_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.JS, chunk_size=60,
chunk_overlap=0
)
js_docs =
js_splitter.create_documents([JS_CODE])
js_docs
```

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

# TS

Here’s an example using the TS text splitter:

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

```
// Call the function
helloWorld();
"""

ts_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.TS, chunk_size=60,
chunk_overlap=0
)
ts_docs =
ts_splitter.create_documents([TS_CODE])
ts_docs
```

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

# Markdown

Here’s an example using the Markdown text splitter:

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

```
⚡ Building applications with LLMs through
composability ⚡

## Quick Install

```bash
# Hopefully this code block isn't split
pip install langchain
```

As an open-source project in a rapidly
developing field, we are extremely open to
contributions.
"""
```

```
md_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.MARKDOWN,
chunk_size=60, chunk_overlap=0
)
md_docs =
md_splitter.create_documents([markdown_text])
md_docs
```

```
[Document(page_content='# 🦜🔗 LangChain'),
 Document(page_content='⚡ Building
applications with LLMs through composability
```

```
⚡'),
 Document(page_content='## Quick
Install\n\n```bash'),
 Document(page_content="# Hopefully this code
block isn't split"),
 Document(page_content='pip install
langchain'),
 Document(page_content='```'),
 Document(page_content='As an open-source
project in a rapidly developing field, we'),
 Document(page_content='are extremely open to
contributions.')]
```

## Latex

Here's an example on Latex text:

```
latex_text = """
\documentclass{article}

\begin{document}

\maketitle

\section{Introduction}
Large language models (LLMs) are a type of
machine learning model that can be trained on
vast amounts of text data to generate human-
```

```
like language. In recent years, LLMs have
made significant advances in a variety of
natural language processing tasks, including
language translation, text generation, and
sentiment analysis.

\subsection{History of LLMs}
The earliest LLMs were developed in the 1980s
and 1990s, but they were limited by the
amount of data that could be processed and
the computational power available at the
time. In the past decade, however, advances
in hardware and software have made it
possible to train LLMs on massive datasets,
leading to significant improvements in
performance.

\subsection{Applications of LLMs}
LLMs have many applications in industry,
including chatbots, content creation, and
virtual assistants. They can also be used in
academia for research in linguistics,
psychology, and computational linguistics.

\end{document}
"""
```

```
latex_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.MARKDOWN,
```

```
chunk_size=60, chunk_overlap=0
)
latex_docs =
latex_splitter.create_documents([latex_text])
latex_docs
```

```
[Document(page_content='\\documentclass{article
 Document(page_content='\\section{Introduction}
 Document(page_content='Large language models (
 Document(page_content='model that can be train
 Document(page_content='generate human-like lan
 Document(page_content='made significant advanc
 Document(page_content='processing tasks, inclu
 Document(page_content='generation, and sentime
 Document(page_content='\\subsection{History of
 Document(page_content='The earliest LLMs were
 Document(page_content='but they were limited b
 Document(page_content='processed and the compu
 Document(page_content='time. In the past decad
 Document(page_content='software have made it p
 Document(page_content='datasets, leading to si
 Document(page_content='performance.'),
 Document(page_content='\\subsection{Applicatio
 Document(page_content='LLMs have many applicat
 Document(page_content='chatbots, content creat
 Document(page_content='can also be used in aca
 Document(page_content='psychology, and computa
 Document(page_content='\\end{document}')]
```

# HTML

Here's an example using an HTML text splitter:

```
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>
"""
```

```
html_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.HTML, chunk_size=60,
chunk_overlap=0
)
html_docs =
html_splitter.create_documents([html_text])
html_docs
```

```
[Document(page_content='<!DOCTYPE
html>\n<html>'),
 Document(page_content='<head>\n
<title>🦜🔗 LangChain</title>'),
 Document(page_content='<style>\n
body {\n                font-family: Aria'),
 Document(page_content='l, sans-serif;\n
}\n            h1 {'),
 Document(page_content='color: darkblue;\n
}\n        </style>\n    </head'),
 Document(page_content='>'),
 Document(page_content='<body>'),
 Document(page_content='<div>\n
<h1>🦜🔗 LangChain</h1>'),
```

```
 Document(page_content='<p>⚡ Building
applications with LLMs through composability
⚡'),
 Document(page_content='</p>\n
</div>'),
 Document(page_content='<div>\n            As
an open-source project in a rapidly dev'),
 Document(page_content='eloping field, we are
extremely open to contributions.'),
 Document(page_content='</div>\n
</body>\n</html>')]
```

## Solidity

Here’s an example using the Solidity text splitter:

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

sol_splitter =
RecursiveCharacterTextSplitter.from_language(
```

```
    language=Language.SOL, chunk_size=128,
chunk_overlap=0
)
sol_docs =
sol_splitter.create_documents([SOL_CODE])
sol_docs
```

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

# C

Here's an example using the C# text splitter:

```
C_CODE = """
using System;
class Program
{
    static void Main()
    {
        int age = 30; // Change the age value
as needed
```

```
        // Categorize the age without any
console output
        if (age < 18)
        {
            // Age is under 18
        }
        else if (age >= 18 && age < 65)
        {
            // Age is an adult
        }
        else
        {
            // Age is a senior citizen
        }
    }
}
"""
c_splitter =
RecursiveCharacterTextSplitter.from_language(
    language=Language.CSHARP, chunk_size=128,
chunk_overlap=0
)
c_docs =
c_splitter.create_documents([C_CODE])
c_docs
```

```
[Document(page_content='using System;'),
 Document(page_content='class Program\n{\n
static void Main()\n    {\n        int age =
30; // Change the age value as needed'),
```

```
 Document(page_content='// Categorize the age
without any console output\n        if (age <
18)\n        {\n            // Age is under
18'),
 Document(page_content='}\n        else if
(age >= 18 && age < 65)\n        {\n
// Age is an adult\n        }\n        else\n
{'),
 Document(page_content='// Age is a senior
citizen\n        }\n    }\n}')]
```