📘 Notes on LangChain Text Splitters

---

1. CharacterTextSplitter

- **How it works**: Splits text into chunks based on a **fixed character count**.
- **Default separator**: `"\n\n"` (double newline = paragraph break).
- **Parameters**:
  - `separator` → single string (default: `"\n\n"`).
  - `chunk_size` → max number of characters per chunk (default `1000`).
  - `chunk_overlap` → overlap between chunks (default `200`).
  - `length_function` → measures chunk length (default: `len` = counts characters).
  - `keep_separator` → include separator at the end of chunks (`False` by default).
- **Pros**:
  - Simple, predictable splitting.
- **Cons**:
  - May cut sentences or words in the middle.
- **Use case**:
  - When you just want **raw character-based chunks** (e.g., testing).

---

2. RecursiveCharacterTextSplitter

- **How it works**: Splits text recursively, trying larger separators first, then smaller ones, until chunks fit the size.
- **Default separators**: `["\n\n", "\n", " ", ""]`
  - Tries: paragraphs → lines → words → characters.
- **Parameters**:
  - `separators` → list of strings (default above).
  - `chunk_size`, `chunk_overlap`, `length_function`, `keep_separator` (same as CharacterTextSplitter).
- **Pros**:

  - Produces **natural, meaningful chunks** (doesn't cut words/sentences unnecessarily).
- **Cons**:
  - Slightly more complex.
- **Use case**:
  - **RAG pipelines**, where context quality matters.

---

3. TokenTextSplitter

- **How it works**: Splits text into chunks of **tokens** using a tokenizer (like OpenAI's `tiktoken`).
- **Parameters**:
  - `chunk_size` → max tokens per chunk (default `1000`).
  - `chunk_overlap` → overlap tokens (default `200`).
  - `encoding_name` → tokenizer (default `"gpt2"`, but `"cl100k_base"` is better for OpenAI embeddings).
  - `add_start_index` → if `True`, adds metadata with character index of each chunk.
- **Pros**:
  - **Accurate for embeddings and LLMs**, since they work on tokens.
- **Cons**:
  - Requires tokenizer; slightly slower.
- **Use case**:
  - Before sending text to **LLMs or embedding models** (avoids exceeding token limits).

🔹 Comparison Table

Splitter	Splitting unit	Default split rule	Best for
CharacterTextSplitter	Characters	Paragraphs (`\n\n`)	Simple, raw splitting

Splitter	Splitting unit	Default split rule	Best for
RecursiveCharacterTextSplitter	Paragraph → Line → Word → Char	Recursive fallback (`["\n\n", "\n", " ", ""]`)	Semantic chunks (RAG)
TokenTextSplitter	Tokens	Model tokenizer (e.g., GPT)	Embeddings + LLM context control

---

🔹 Other Splitters in LangChain

LangChain provides several other specialized splitters:

1. **MarkdownHeaderTextSplitter**
   - Splits Markdown documents based on **headers** (`#` , `##` , `###` , etc.).
   - Useful for structured knowledge bases, wikis, or technical docs.
2. **HTMLHeaderTextSplitter**
   - Similar to Markdown, but works on **HTML tags** (`<h1>` , `<h2>` , etc.).
3. **Language-aware Splitters (Code-specific)**
   - **PythonCodeTextSplitter**, **CppTextSplitter**, **JavaScriptSplitter**, etc.
   - Split code files by functions, classes, or logical blocks instead of plain characters.
4. **SpacyTextSplitter**
   - Uses **spaCy NLP library** to split text by sentences.
   - More linguistically accurate.
5. **NLTKTextSplitter**
   - Uses **NLTK** to split by sentences/words.
   - Alternative to spaCy.
6. **SentenceTransformersTokenTextSplitter**
   - Token-aware splitter built for **sentence-transformers embeddings**.

✅ Final Takeaways

- Use `CharacterTextSplitter` → if you just want fast, simple, character-based chunks.
- Use `RecursiveCharacterTextSplitter` → for **RAG pipelines** → keeps chunks semantically clean.
- Use `TokenTextSplitter` → when working with **LLMs or embeddings** → ensures chunks fit within token limits.
- Use **Markdown/HTML/Code splitters** → when your data has a **clear structure** (docs, web pages, codebases).