What makes documentation good

Author: Ted Sanders

Article: https://cookbook.openai.com/articles/what_makes_documentation_good

Documentation puts useful information inside other people's heads. Follow these tips to write better documentation.

Make docs easy to skim

- Few readers read linearly from top to bottom. They'll jump around, trying to assess which bit solves their problem, if any. To reduce their search time and increase their odds of success, make docs easy to skim.
- **Split content into sections with titles.** Section titles act as signposts, telling readers whether to focus in or move on.
- **Prefer titles with informative sentences over abstract nouns.** For example, if you use a title like "Results", a reader will need to hop into the following text to learn what the results actually are. In contrast, if you use the title "Streaming reduced time to first token by 50%", it gives the reader the information immediately, without the burden of an extra hop.

- **Include a table of contents.** Tables of contents help readers find information faster, akin to how hash maps have faster lookups than linked lists. Tables of contents also have a second, oft overlooked benefit: they give readers clues about the doc, which helps them understand if it's worth reading.
- **Keep paragraphs short.** Shorter paragraphs are easier to skim. If you have an essential point, consider putting it in its own one-sentence paragraph to reduce the odds it's missed. Long paragraphs can bury information.
- **Begin paragraphs and sections with short topic sentences that give a standalone preview.** When people skim, they look disproportionately at the first word, first line, and first sentence of a section. Write these sentences in a way that don't depend on prior text. For example, consider the first sentence "Building on top of this, let's now talk about a faster way." This sentence will be meaningless to someone who hasn't read the prior paragraph. Instead, write it in a way that can understood standalone: e.g., "Vector databases can speed up embeddings search."
- **Put topic words at the beginning of topic sentences.** Readers skim most efficiently when they only need to read a word or two to know what a paragraph is about. Therefore, when writing topic sentences, prefer putting the topic at the beginning of the sentence rather than the end. For example, imagine you're writing a paragraph on vector databases in the middle of a long article on embeddings search. Instead of writing "Embeddings search can be sped up by vector databases" prefer "Vector databases speed up embeddings search." The second sentence is better for skimming, because it puts the paragraph topic at the beginning of the paragraph.
- **Put the takeaways up front.** Put the most important information at the tops of documents and sections. Don't write a Socratic big build up. Don't introduce your procedure before your results.
- **Use bullets and tables.** Bulleted lists and tables make docs easier to skim. Use them frequently.
- **Bold important text.** Don't be afraid to bold important text to help readers find it.

Write Well

Badly written text is taxing to read. Minimize the tax on readers by writing well.

- **Keep sentences simple.** Split long sentences into two. Cut adverbs. Cut unnecessary words and phrases. Use the imperative mood, if applicable. Do what writing books tell you.
- **Write sentences that can be parsed unambiguously.** For example, consider the sentence "Title sections with sentences." When a reader reads the word "Title", their brain doesn't yet know whether "Title" is going to be a noun or verb or adjective. It takes a bit of brainpower to keep track as they parse the rest of the sentence, and can cause a hitch if their brain mispredicted the meaning. Prefer sentences that can be parsed more easily (e.g., "Write section titles as sentences") even if longer. Similarly, avoid noun phrases like "Bicycle clearance exercise notice" which can take extra effort to parse.
- **Avoid left-branching sentences.** Linguistic trees show how words relate to each other in sentences. Left-branching trees require readers to hold more things in memory than right-branching sentences, akin to breadth-first search vs depth-first search. An example of a left-branching sentence is "You need flour, eggs, milk, butter and a dash of salt to make pancakes." In this sentence you don't find out what 'you need' connects to until you reach the end of the sentence. An easier-to-read right-branching version is "To make pancakes, you need flour, eggs, milk, butter, and a dash of salt." Watch out for sentences in which the reader must hold onto a word for a while, and see if you can rephrase them.
- **Avoid demonstrative pronouns (e.g., "this"), especially across sentences.** For example, instead of saying "Building on our discussion of the previous topic, now let's discuss function calling" try "Building on message formatting, now let's discuss function calling." The second sentence is easier to understand because it doesn't burden the reader with recalling the previous topic. Look for opportunities to cut demonstrative pronouns altogether: e.g., "Now let's discuss function calling."
- **Be consistent.** Human brains are amazing pattern matchers. Inconsistencies will annoy or distract readers. If we use Title Case everywhere, use Title Case. If we use terminal commas everywhere, use terminal commas. If all of the Cookbook notebooks are named with underscores and sentence case, use

underscores and sentence case. Don't do anything that will cause a reader to go 'huh, that's weird.' Help them focus on the content, not its inconsistencies.

- **Don't tell readers what they think or what to do.** Avoid sentences like "Now you probably want to understand how to call a function" or "Next, you'll need to learn to call a function." Both examples presume a reader's state of mind, which may annoy them or burn our credibility. Use phrases that avoid presuming the reader's state. E.g., "To call a function, ..."

Be broadly helpful

People come to documentation with varying levels of knowledge, language proficiency, and patience. Even if we target experienced developers, we should try to write docs helpful to everyone.

- **Write simply.** Explain things more simply than you think you need to. Many readers might not speak English as a first language. Many readers might be really confused about technical terminology and have little excess brainpower to spend on parsing English sentences. Write simply. (But don't oversimplify.)
- **Avoid abbreviations.** Write things out. The cost to experts is low and the benefit to beginners is high. Instead of IF, write instruction following. Instead of RAG, write retrieval-augmented generation (or my preferred term: the search-ask procedure).
- **Offer solutions to potential problems.** Even if 95% of our readers know how to install a Python package or save environment variables, it can still be worth proactively explaining it. Including explanations is not costly to experts—they can skim right past them. But excluding explanations is costly to beginners—they might get stuck or even abandon us. Remember that even an expert JavaScript engineer or C++ engineer might be a beginner at Python. Err on explaining too much, rather than too little.
- **Prefer terminology that is specific and accurate.** Jargon is bad. Optimize the docs for people new to the field, instead of ourselves. For example, instead of writing "prompt", write "input." Or instead of writing "context limit" write "max token limit." The latter terms are more self-evident, and are probably better than the jargon developed in base model days.

- **Keep code examples general and exportable.** In code demonstrations, try to minimize dependencies. Don't make users install extra libraries. Don't make them have to refer back and forth between different pages or sections. Try to make examples simple and self-contained.
- **Prioritize topics by value.** Documentation that covers common problems—e.g., how to count tokens—is magnitudes more valuable than documentation that covers rare problems—e.g., how to optimize an emoji database. Prioritize accordingly.
- **Don't teach bad habits.** If API keys should not be stored in code, never share an example that stores an API key in code.
- **Introduce topics with a broad opening.** For example, if explaining how to program a good recommender, consider opening by briefly mentioning that recommendations are widespread across the web, from YouTube videos to Amazon items to Wikipedia. Grounding a narrow topic with a broad opening can help people feel more secure before jumping into uncertain territory. And if the text is well-written, those who already know it may still enjoy it.

Break these rules when you have a good reason

Ultimately, do what you think is best. Documentation is an exercise in empathy. Put yourself in the reader's position, and do what you think will help them the most.