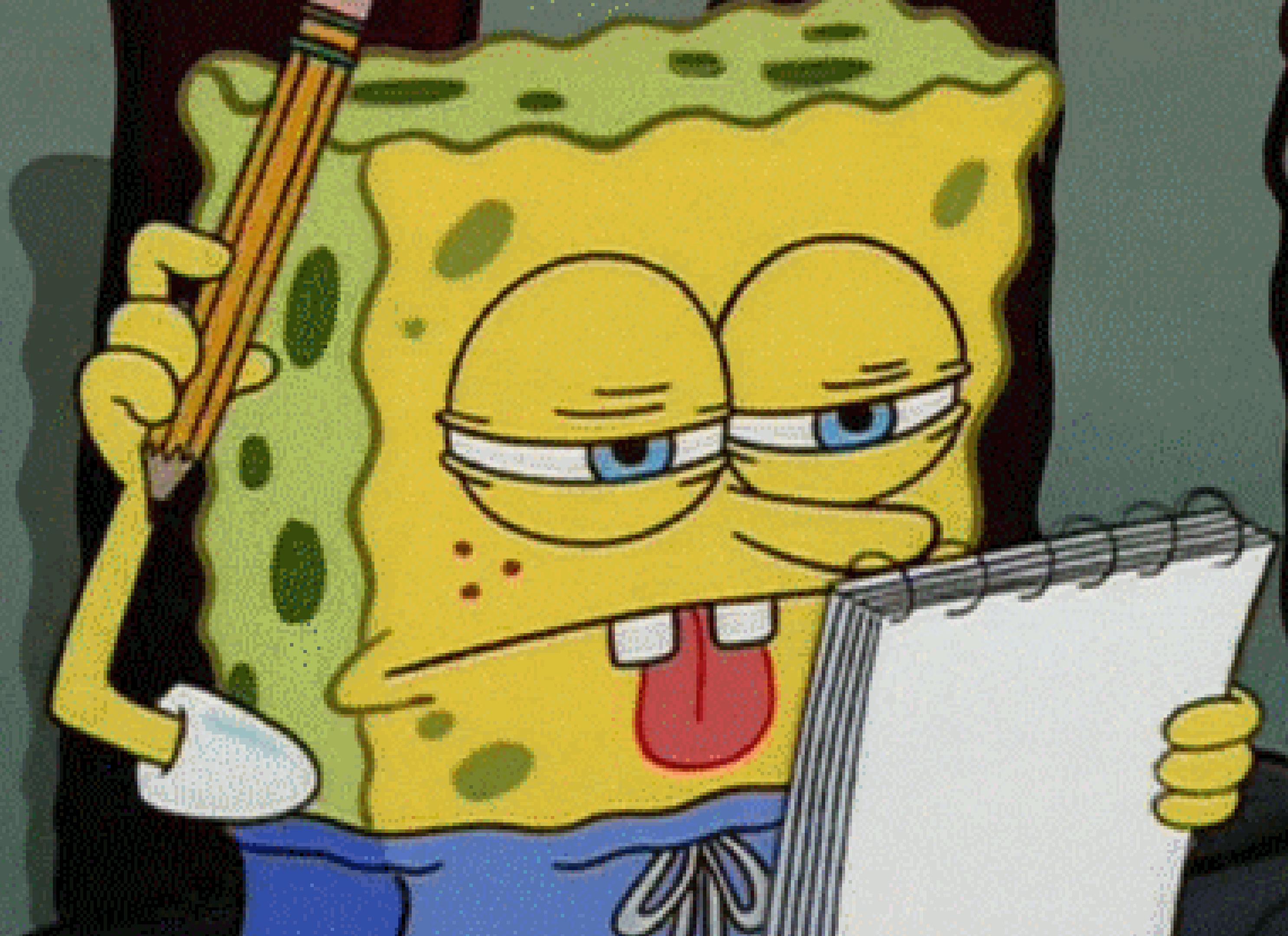


Getting started with LlamaIndex







Forster, E.M.

letter

Reed, 12 December

MUSCIFR

Reed, Henry
Ken Russ

published

1885

MS.

C

F

C

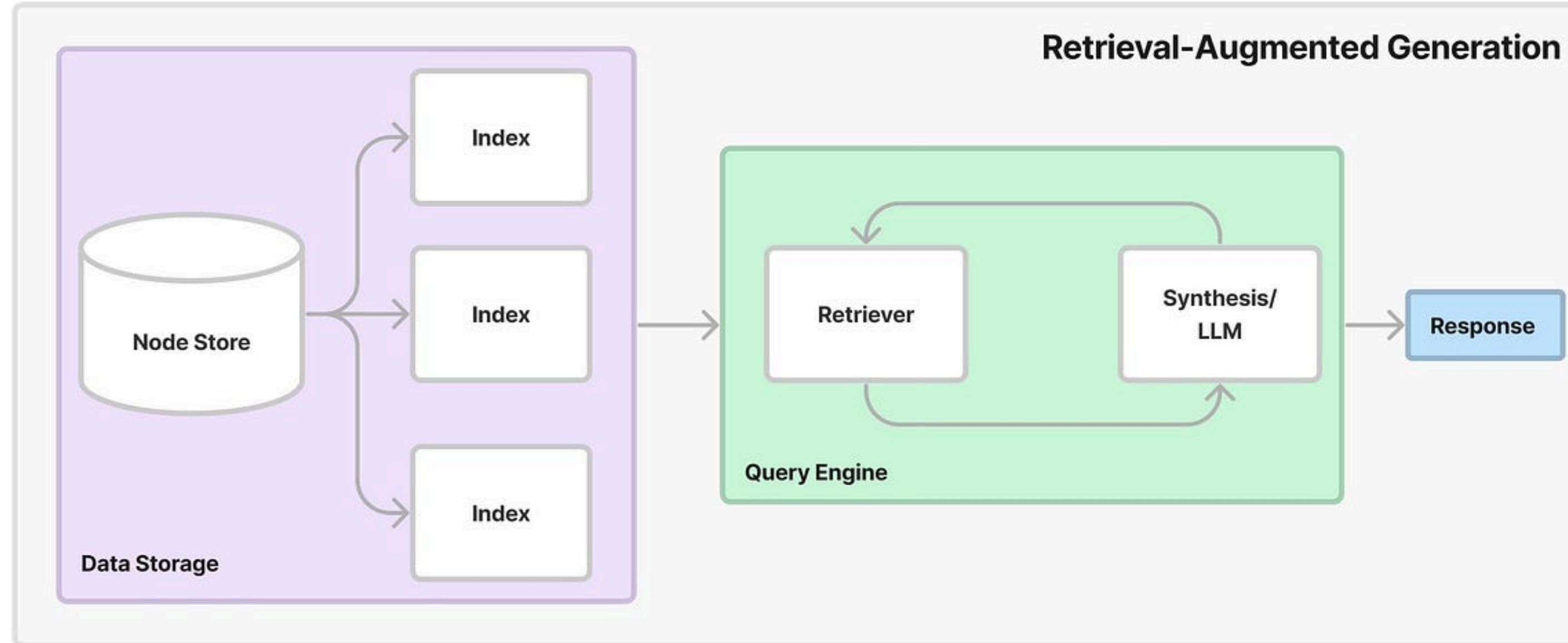
F

C

F



Retrieval-Augmented Generation



AC

You

who is spongebob



ChatGPT

SpongeBob SquarePants is a fictional character and the main protagonist of the animated television series created by marine science educator and animator Stephen Hillenburg for Nickelodeon. SpongeBob is depicted as a good-natured, optimistic, and enthusiastic sea sponge who physically resembles a rectangular kitchen sponge, living in a pineapple house in the underwater city of Bikini Bottom. He works as a fry cook at the Krusty Krab, a fast-food restaurant famous for its Krabby Patty burgers. The show features his various adventures with his friends and neighbors, including his best friend Patrick Star, his boss Mr. Krabs, and his neighbor Squidward Tentacles. SpongeBob SquarePants has become a popular and influential part of children's entertainment since it first aired in 1999.



AC

You

who is spongebob

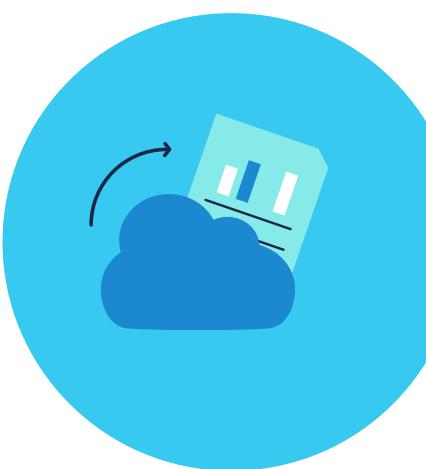


SpongeBob Seeker

SpongeBob SquarePants is an energetic and optimistic sea sponge who physically resembles a rectangular kitchen sponge. He lives in a pineapple house in the underwater city of Bikini Bottom. SpongeBob works as a fry cook at the Krusty Krab, a fast food restaurant known for its Krabby Patty burgers. He has a pet snail named Gary who meows like a cat. Despite his job's simplicity, SpongeBob is incredibly enthusiastic about it. He's known for his trademark square pants and his ability to get into all sorts of misadventures with his dimwitted yet endearing starfish friend, Patrick Star, and other colorful characters around Bikini Bottom. SpongeBob's ever-cheerful demeanor and his knack for getting into sticky situations are central themes throughout the series.



LOADING



Data is ingested into the RAG system

INDEXING



Data is structured and formatted for easier access

STORING



Indexed data (embeddings) are stored properly for later access

QUERYING



Specific data is retrieved based on user's queries

Can you help me write better?



Courses Search Khan Academy Welcome Catalina

This conversation is recorded and viewable by your parent and teacher.

Activities

Let's learn together through any of these activities that interest you!

Welcome! Writing with the AI > Getting feedback Academic coaching Talking to historical characters

Welcome! I'm Khanmigo. I use Artificial Intelligence (AI) to help you learn with Khan Academy.

You can choose an activity from the list on the left, or ask me to help you choose an activity.

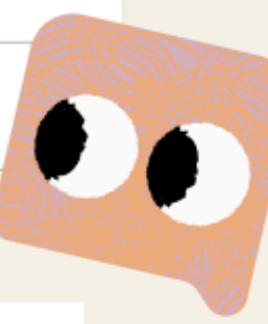
I can also figure out what you should learn next, just ask me!

What should I learn next?

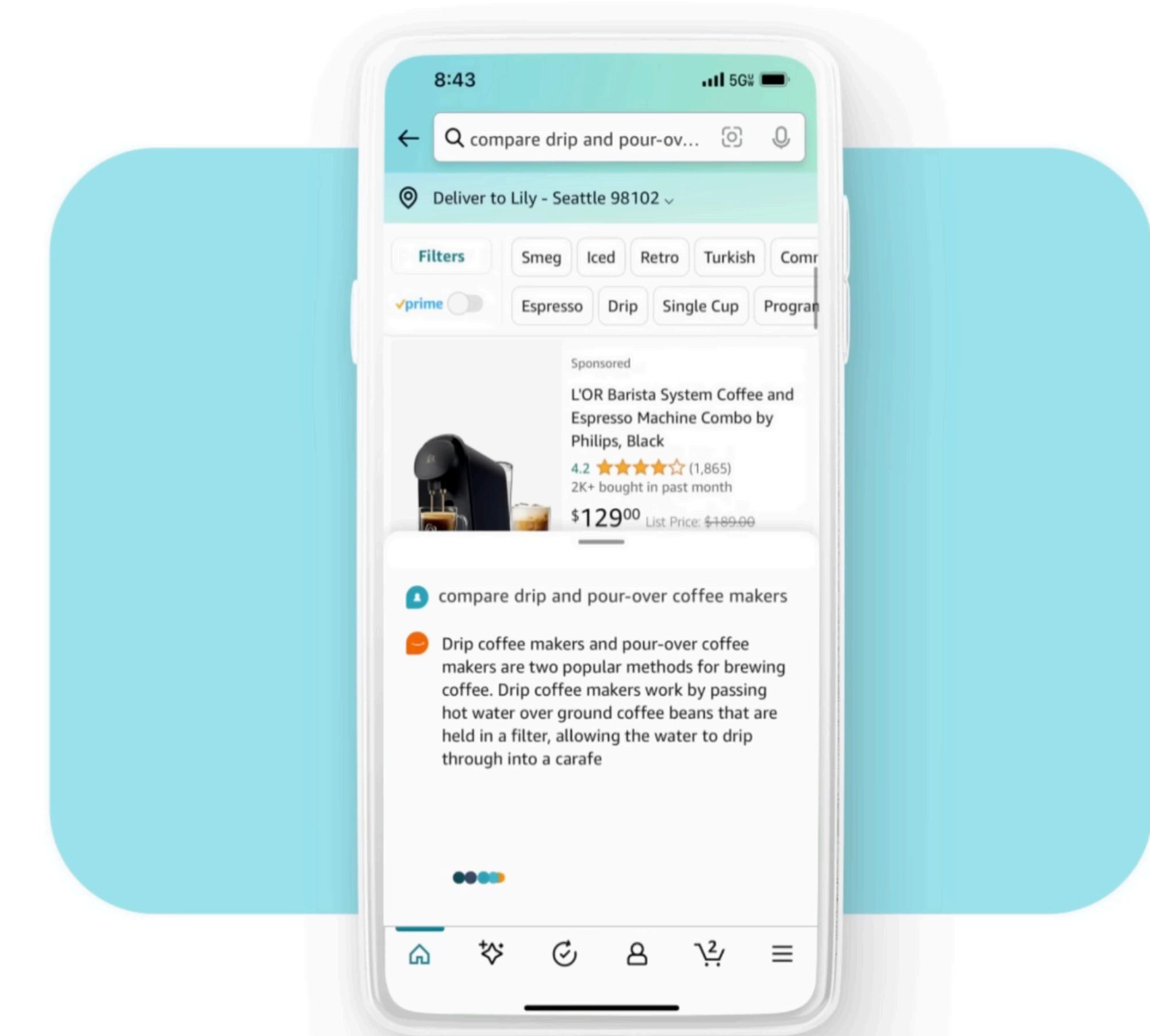
Let's focus on the Variables & expressions unit within 6th grade math. I recommend working on the following lessons:

Type message (do NOT share any personal data)

Khanmigo makes mistakes sometimes. [Here's why.](#)

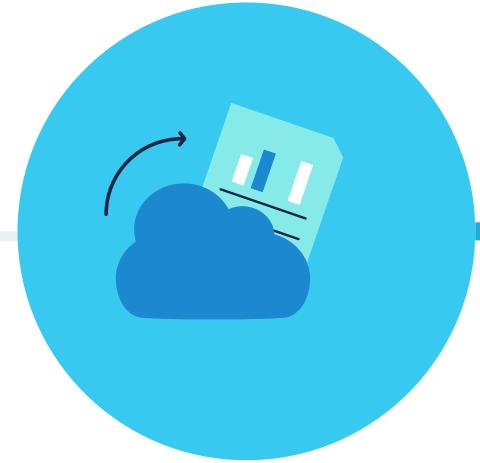


Yes! What kind of writing do you want to improve?





Llamaindex



LOADING

LlamaHub offers a variety of data loaders from different APIs and document formats, providing a natural integration point with the ChatGPT Retrieval Plugin.



INDEXING

The platform supports the easy construction of multiple index types, such as the Vector Store Index, which creates vector embeddings of text from nodes for efficient querying.



STORING

LlamaIndex supports various vector stores for data storage. You can specify a storage method using the StorageContext and specify the vector store, as demonstrated with Pinecone in the example.



QUERYING

LlamaIndex supports complex querying mechanisms. For instance, it allows the use similarity search when querying the documents.



notebook

```
%pip install llama-index-multi-modal-llms-gemini --quiet  
%pip install llama-index-vector-stores-chroma --quiet  
%pip install llama-index-embeddings-gemini --quiet  
%pip install llama-index-llms-gemini --quiet  
%pip install llama-index-readers-file --quiet
```

- Installation: Before using the library, you must install it in your environment:
 - Use the command !pip install llama-index to install.
- Importing Components: Once installed, import the necessary components of the library:
 - Example: from llama_index.prompts import PromptTemplate



```
from llama_index.core import VectorStoreIndex, StorageContext, SimpleDirectoryReader,  
Settings, StorageContext  
from llama_index.embeddings.gemini import GeminiEmbedding  
from llama_index.llms.gemini import Gemini  
from llama_index.vector_stores.chroma import ChromaVectorStore  
import llama_index  
import chromadb  
import os
```

- Installation: Before using the library, you must install it in your environment:
 - Use the command !pip install llama-index to install.
- Importing Components: Once installed, import the necessary components of the library:
 - Example: from llama_index.prompts import PromptTemplate



notebook

```
GOOGLE_API_KEY = "AIzaSyBKNj89SQXJsEit9VAhWMTEEitt61n2VHk" # add your GOOGLE API key here  
os.environ["GOOGLE_API_KEY"] = GOOGLE_API_KEY
```

- Gemini AI Model: For our project, we utilize Google's Gemini, a versatile generative AI model. Gemini supports various applications such as chatting, embedding generation, image, and audio processing.
- API Access:
 - API Key: Access to Gemini requires a Google API key. This key enables interaction with Gemini's functionalities.



notebook

```
documents = SimpleDirectoryReader('/content/data').load_data()
```

- Data Loading:
 - LlamaIndex supports various data formats including CSV, DOCX, EPUBs, HTML, Jupyter Notebooks, and more, facilitating versatile data handling capabilities.
 - For simplicity, you can use a directory reader to load data directly from a folder, ideal for processing multiple files efficiently, such as PDFs.



notebook

```
chroma_client = chromadb.EphemeralClient()  
chroma_collection = chroma_client.get_or_create_collection("genai_ph")  
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
```

- Storage Essentials:
 - Transforming data into embeddings requires them to be stored in a structured container. This setup enables easy access, modification, and deletion of the embeddings.
- Chroma Client:
 - The Chroma Client acts as a mediator, facilitating communication between user commands and the system.



notebook

```
chroma_client = chromadb.EphemeralClient()  
chroma_collection = chroma_client.get_or_create_collection("genai_ph")  
vector_store = ChromaVectorStore(chroma_collection=chroma_collection)
```

- Chroma Collections:
 - Collections in Chroma function similarly to folders on a computer, organizing your embeddings efficiently. These collections can be created or accessed using identifiers, which means you can retrieve or initialize them as needed.
- Vector Store Integration:
 - Once embeddings are placed in a collection, they are managed through a vector store. This system supports operations like manipulation and retrieval of embeddings, with various vector stores available that differ in features and performance advantages.



notebook

```
Settings.embed_model = GeminiEmbedding(  
    model_name="models/embedding-001"  
)  
Settings.llm = Gemini(api_key=GOOGLE_API_KEY)
```

- Setting Up the Embedding Model:
 - Through LlamaIndex settings, you can specify which AI model to utilize for embeddings. For instance, selecting the 'GeminiEmbedding' model allows you to define the model details and integrate the necessary Google API key for operation.
- Configuring API Settings:
 - Further customization of settings includes specifying the API key for the Gemini model, ensuring secure and effective communication with Google's AI services.



notebook

```
storage_context = StorageContext.from_defaults(vector_store=vector_store)

index = VectorStoreIndex.from_documents(
    documents,
    storage_context=storage_context,
)
```

- Storage Configuration:
 - The 'StorageContext' is set up, typically with default parameters, to determine how embeddings are stored. This setting ties directly into how the data is managed and retrieved.
- Creating and Managing the Index:
 - This function reads from the specified documents (PDF files in this case) and applies the predefined storage settings to organize, generate, and store the generated embeddings .



notebook

```
query_engine = index.as_query_engine(  
    similarity_top_k=1,  
)  
  
response = query_engine.query(  
    "What is the difference between the valleys and the hills"  
)  
print(response)
```

- Setting Up the Query Engine:
 - To interact with the indexed data, set up a query engine using the `index.as_query_engine` function. This allows you to specify parameters such as `similarity_top_k`, which helps determine the most relevant data points based on your query.
- Executing Queries:
 - With the query engine in place, you can start querying the data. Simply place your specific questions within the `query` function to retrieve relevant information from your data.