🤖 AIチャットボット開発レクチャー

🎯 ゴール

ラシキアゼミのWebサイト（https://lanet.sist.chukyo-u.ac.jp/）の情報をもとに、
質問に自動で答える**AIチャットボット**を構築する。

🛠 使用技術一覧

技術	内容
Crawl4AI	Webサイトをクロールしてテキスト抽出
Chroma	テキストをベクトル形式で保存・検索
LangChain	ベクトル検索・RAG構成の支援
OpenAI API	ChatGPTモデルで回答生成
Streamlit	簡単なWebチャットUIを作成

🔧 準備

1. Python環境を整える

Python 3.10 以上

```
python -m venv venv
venv\Scripts\activate  # Mac の場合: source venv/bin/activate
```

2. 必要なパッケージをインストール

requirements.txt

```
crawl4ai
langchain
langchain-community
langchain-openai
chromadb
openai
tiktoken
streamlit
python-dotenv
unstructured
```

```
pip install -r requirements.txt
```

3. OpenAI APIキーの準備

.envファイルの作成

```
OPENAI_API_KEY=sk-xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

☑ ステップ1：サイトをクロール（データ収集）

ここでは、crawl4ai を用いて ラシキアゼミWebサイトの複数ページをクロールし、Markdown形式の本文を .txt ファイルとして保存します。

Crawl4AI の役割と利点

Crawl4AI は、従来のスクレイピングツールとは異なり、AI時代に最適化されたデータ取得ツールです。 以下の特徴を活かして、RAG 構成に必要な高品質なテキストデータ収集を実現します。

☑ 特徴と強み

特徴	内容
JSレンダリング対応	`Playwright` を使用し、JavaScriptで動的に生成されるページ内容も正確に取得可能
Markdown形式で出力	AIにとって扱いやすい階層構造・見出し構造を維持したデータが得られる
セッション再利用対応	同一ブラウザセッションで複数ページを高速にクロール可能
高速かつ非同期	`asyncio` を活用した非同期処理で高速なクロールが可能
構造化された結果	Markdown 形式により、見出し（`#`）・リスト（`-`）・リンクなどが明確に区分けされ、ベクトル化に適した構造が得られる

使用コード：crawl_lanet_session_save.py

- コードの役割
  クローリングした情報をMarkDown形式に変換してテキストファイルに保存する。

ファイル:crawl_lanet_session_save.py

```
import asyncio, os, re
from typing import List
from crawl4ai import AsyncWebCrawler, BrowserConfig, CrawlerRunConfig
from crawl4ai.markdown_generation_strategy import DefaultMarkdownGenerator

# クロール対象URL（必要に応じて追加）
TARGET_URLS = [
    "https://lanet.sist.chukyo-u.ac.jp/",
    "https://lanet.sist.chukyo-u.ac.jp/activities",
    "https://lanet.sist.chukyo-u.ac.jp/societies",
    "https://lanet.sist.chukyo-u.ac.jp/researches",
    "https://lanet.sist.chukyo-u.ac.jp/jobs",
    "https://lanet.sist.chukyo-u.ac.jp/members",
    "https://lanet.sist.chukyo-u.ac.jp/links"
]

# 保存先ディレクトリ
OUTPUT_DIR = "lanet_data"
os.makedirs(OUTPUT_DIR, exist_ok=True)

# ファイル名のサニタイズ関数
def sanitize_filename(url: str) -> str:
    return re.sub(r'[^\w\-_.]', '_', url.strip("/"))[:100]

# クローリングとMarkdown形式で保存する関数
async def crawl_sequential_and_save(urls: List[str]):
    print("\n=== MarkDown形式で保存中 ===")

    # ブラウザ設定
    browser_config = BrowserConfig(
        headless=True,
        extra_args=["--disable-gpu", "--disable-dev-shm-usage", "--no-sandbox"],
    )

    # クローリング設定
    crawl_config = CrawlerRunConfig(
        markdown_generator=DefaultMarkdownGenerator()
    )

    # 非同期クローラーの初期化
    crawler = AsyncWebCrawler(config=browser_config)
    await crawler.start()

    try:
        session_id = "lanet_session"
```

```
        for url in urls:
            # クローリング
            result = await crawler.arun(
                url=url,
                config=crawl_config,
                session_id=session_id
            )

            # 結果の書き込み
            if result.success:
                print(f"☑ Success: {url}")
                filename = sanitize_filename(url)
                path = os.path.join(OUTPUT_DIR, f"{filename}.txt")
                with open(path, "w", encoding="utf-8") as f:
                    f.write(result.markdown.raw_markdown or "")
                print(f"📄 保存: {path}")
            else:
                print(f"❌ Failed: {url} - {result.error_message}")
    finally:
        await crawler.close()
        print("☑ クロール完了（すべてのセッションを閉じました）")

async def main():
    await crawl_sequential_and_save(TARGET_URLS)

if __name__ == "__main__":
    asyncio.run(main())
```

- 結果：lanet_data/ フォルダに .txt ファイル群が保存されます。

☑ Markdown（MD形式）にする主な理由

対象	MD形式での利点
検索精度	見出し単位・段落単位の関連性が明確になる
LLM回答	「この〜について」の問いに一貫した文脈で返せる
将来の再利用	PDF・HTML化・表示にも応用しやすい

☑ 他との比較まとめ（表）

ツール	特徴	向いている用途
Crawl4AI	Markdown出力、非同期、Playwright、RAG最適化	AIチャットボット、RAG前処理、LLM学習素材
BeautifulSoup	軽量、シンプル、カスタム解析容易	小規模スクレイピング、特定要素の抽出
Selenium	JSレンダリング、操作再現性あり	自動化・ブラウザ操作が必要な検証系

☑ ステップ2：ベクトルDBを構築（LangChain + Chroma）

📌 なぜベクトルデータベースを使うのか？

- 従来のキーワード検索では、単語の「意味」までは理解できません。
- ベクトルデータベースは、テキスト（文章）を意味ベースの数値（ベクトル）に変換して保存し、クエリに近い意味の文書を効率的に検索可能にします。
- そのため、自然言語の質問に対して意味的に関連する情報を正確に取り出せるという利点があります。

🔍 RAG（Retrieval-Augmented Generation）とは？

- RAG（検索拡張生成）は、以下の2つを組み合わせた仕組みです：

1. Retrieval（検索）：質問に対してベクトルDBから関連文書を取り出す。
2. Generation（生成）：取り出した文書を元に、ChatGPTなどの大規模言語モデルが自然な文章で回答を生成。

🗀 使用コード：build_vector_db.py

- コードの役割
  このスクリプトは、事前に収集したラシキア研究室のMarkdownまたはテキストファイルを読み込み、チャンク分割 → 埋め込み変換 → Chroma形式で保存する処理を行います。

これにより、事前に学習されていない内容でも回答が可能になります。
ファイル:build_vector_db.py

```
from langchain_community.document_loaders import DirectoryLoader, TextLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_openai import OpenAIEmbeddings
from langchain_community.vectorstores import Chroma
from dotenv import load_dotenv
import os

# .env から OPENAI_API_KEY を読み込み
load_dotenv()

# ① ディレクトリ内のMarkdown or テキストファイルを読み込む
loader = DirectoryLoader(
    "lanet_data",
    glob="**/*.txt",
    loader_cls=lambda path: TextLoader(path, encoding="utf-8")
)
docs = loader.load()
print(f"📄 ドキュメント数: {len(docs)}")

# ② チャンク（分割）処理：LLMで扱いやすいサイズに分割
splitter = RecursiveCharacterTextSplitter(chunk_size=500, chunk_overlap=50)
split_docs = splitter.split_documents(docs)
print(f"✂️ 分割後ドキュメント数: {len(split_docs)}")
```

```
# ③ テキストをベクトル化 → Chromaで保存
embedding = OpenAIEmbeddings(openai_api_key=os.getenv("OPENAI_API_KEY"))
db = Chroma.from_documents(split_docs, embedding,
persist_directory="lanet_chroma_md")

print("✅ ベクトルDB作成完了（Markdown対応）")
```

⚙️ コマンド実行

```
python build_vector_db.py
```

🔄 処理の流れ図（簡略）

```
Markdown/TXT 読込
     ↓
チャンク分割（意味単位のテキスト）
     ↓
OpenAI Embedding によりベクトル化
     ↓
Chroma DB に保存（RAGに活用）
```

📁 LangChainの仕組みの作用

LangChainは、LLMを使ったアプリを効率的に構築するためのライブラリです。今回のRAG構成においてLangChainが担う主な仕事は次のとおりです：

- 文書読み込み：DirectoryLoader などを使って文書を解析
- 分割：RecursiveCharacterTextSplitter でテキストを適切なチャンクに分割
- 基盤処理：OpenAIEmbeddingsやChromaベクトルDBとの連携
- RAGチェーンの作成：RetrievalQA.from_chain_type などを用いて、検索とLLMを組み合わせる
  つまりLangChainは、「データをうまく分割し、検索し、回答まで連携させる」ための奥絡系の中心部分です。

🧠 OpenAIEmbeddings の役割

項目	内容
機能	OpenAIの埋め込みAPIを使って、テキストをベクトルに変換
用途	ベクトルDBに登録したり、類似検索に使ったりするための前処理
使い方	`embedding = OpenAIEmbeddings(openai_api_key=...)`
背景モデル	`text-embedding-3-small` などが裏で使用される（2024年現在）

🧠 Chromaとは？（ベクトルデータベース）

Chroma は、文章の「意味」に基づいて検索できるベクトルデータベース（Vector Database）です。

従来のキーワード検索とは異なり、意味的に類似した情報を探すのに適しています。

💡 なぜ Chroma を使うのか？

従来の検索	Chromaによる検索
「単語の一致」が中心	「意味の近さ」でマッチングする
類語や言い換えに弱い	表現の違いを吸収できる
例：「就職」≠「進路」	例：「就職」≒「進路」「キャリア」など

🔍 Chroma の特徴

特徴	説明
🧭 意味ベース検索	テキストをベクトル（数値）に変換して保存・検索
🛠 ローカルで使える	SQLiteベースで、サーバー構築不要
🔁 LangChain対応	`.from_documents()` や `.as_retriever()` など便利な連携関数が豊富
⚡ 高速な検索	数千〜数百万件でも高速な意味検索が可能
💰 無料・OSS	オープンソースで商用利用もOK

🧩 RAG構成での役割（重要！）

Chromaは、RAG（検索拡張生成）構成における**「検索部分（Retrieval）」**を担います。

```
ユーザーの質問
    ↓（意味ベクトルに変換）
ChromaベクトルDBが意味的に近い文書を検索
    ↓
ChatGPTなどのLLMが回答を生成（RAG）
```

☑ ステップ3：チャットボットアプリを構築（Streamlit）

このステップの目的

- ベクトルDBとChatGPTを使って、ユーザーの質問に自動で応答する「対話型チャットボット」を構築します。
- ユーザーが入力した質問文に対して、関連する情報を検索し、その情報をもとにAIが自然な応答を返します。

使用技術とその役割

技術	役割
Streamlit	WebアプリのUI作成。ブラウザ上でチャットできる画面を提供
LangChain	ベクトルDB（Chroma）との接続や、RAGの構築を簡素化するライブラリ
OpenAI API	ChatGPTモデル（GPT-3.5-turboなど）で自然な回答文を生成
Chroma（langchain-chroma）	検索のためのベクトルデータベースを保持・操作

コード全体の流れ

1. 環境変数読み込み（OpenAIのAPIキー）
2. StreamlitのUI表示
3. ChromaベクトルDBのロード
4. ChatGPTモデルの設定
5. ユーザー入力 → 関連情報の検索 → 回答生成
6. Streamlit上に表示<br ファイル:chat_app.py

```
import streamlit as st
from dotenv import load_dotenv
import os

from langchain_community.document_loaders import DirectoryLoader, TextLoader
from langchain_openai import ChatOpenAI, OpenAIEmbeddings
from langchain_chroma import Chroma
from langchain.chains import RetrievalQA

# .envからAPIキーを読み込み
load_dotenv()

st.title("ラシキア研究室チャットボット 🤖")
query = st.text_input("質問を入力してください")

# ベクトルDBロード
embedding = OpenAIEmbeddings(openai_api_key=os.getenv("OPENAI_API_KEY"))
db = Chroma(persist_directory="lanet_chroma_md", embedding_function=embedding)
```

```
# チャットモデルとRAGチェーンの設定
llm = ChatOpenAI(temperature=0, model_name="gpt-3.5-turbo",
openai_api_key=os.getenv("OPENAI_API_KEY"))
qa = RetrievalQA.from_chain_type(llm=llm, retriever=db.as_retriever())

# 応答処理
if query:
    with st.spinner("考え中..."):
        result = qa.invoke({"query": query})
        st.success(result["result"])
```

☑ ステップ4：Streamlitアプリを起動

```
streamlit run chat_app.py
```

☑ 動作例（質問例集）

🔬 研究・活動に関する質問

- 「ラシキアゼミの研究テーマは何ですか？」
- 「ラシキアゼミではどのような卒業研究が行われていますか？」

📅 ゼミ活動・行事

- 「ゼミではどんな活動がありますか？」
- 「ラシキアゼミの最近のイベントを教えてください」

👤 メンバーに関する質問

- 「学部3年のメンバーを教えてください」
- 「ゼミのメンバーである片倉悠紀さんについて教えてください。」

💼 就職・進路について

- 「ラシキアゼミの卒業生の就職先はどこですか？」
- 「ラシキアゼミの卒業生はどんな企業に就職していますか？」

🔧 その他

- 「ラシキアゼミの公式サイトURLを教えてください」
- 「ラシキアゼミという名前の由来は何ですか？（※答えがなければ '情報がありません' と返答されます）」