2190101 Computer Programming

Academ Chatbot

6838204021 Pattadon Khamdee

6838234921 Pattrachonm Vejchakul

6838310921 Supolthit Rochanakeekajon

6838318021 Anapat Leemakdej

Information and Communication Engineering (ICE)

International School of Engineering, Chulalongkorn University

Academic Year 2025

Contents

Objective 3
Description of Input and Output 4
Details of Libraries Used 7
Code Walkthrough 8
  Overall Architecture 8
  Streamlit setup & layout 8
  User Login ("Join" System) 9
  Chat History Storage 10
  Chat Interface Rendering 11
  OpenAI Response Generation 12
  Output Handling 13
Results / Output Examples 14
Reference 14
    Official Documentation 15
    Tutorials & Learning Resources 15
Declaration 16
Requirements 17

Objective

This project focuses on creating and developing a chatbot, ‘Academ,’ that provides assistance for students in Chulalongkorn University. Our goal is to help them access vital information related to academic or campus matters. By offering instant responses, the chatbot serves to reduce time for students while giving them the right information about the university.

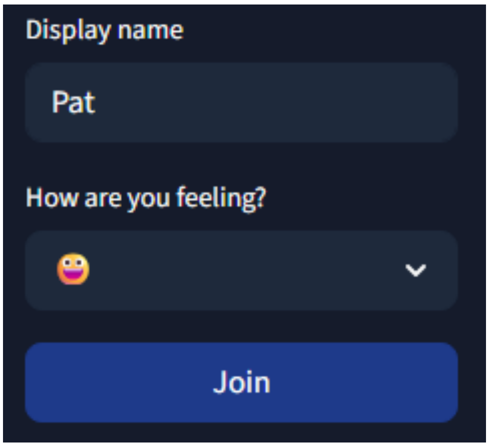
Several subjects relevant to student life are covered by the chatbot's capability. It will provide details about academic programs, including courses, majors, and degree requirements, as well as advice on a range of campus facilities, including housing, counseling, and libraries. It will also assist users in remaining in touch with the school community by offering updates on clubs, events, and student activities. For both present and future pupils, the chatbot acts as a single source for information by answering general university questions. In addition to improving ease, this design promotes participation and independence among people in need of support.

In addition to helping students, the project illustrates important programming concepts and approaches to problem-solving. It places a heavy focus on the practical application of fundamental Python ideas like functions, loops, and data structures, which are the cornerstones of organized and effective coding. These features are used by the chatbot to process user input, store data, and continuously provide relevant responses. The developers exhibit their ability to create logical, modular, and reusable code that enhances the chatbot's reliability and overall performance by putting these components into practice.

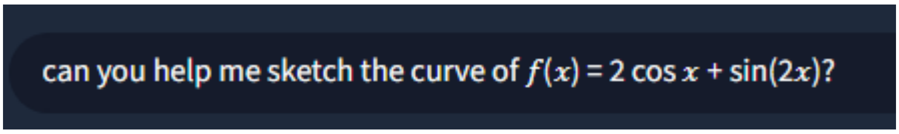
This project shows how technology can help improve access to academic resources and support for students. It combines technical education with real-world application to demonstrate how software can enhance university services. By merging programming skills with user-focused design, the project takes a thoughtful approach to innovation and communication in education. The chatbot is an example of how digital tools can link information and accessibility, benefiting the entire university community.

Description of Input and Output

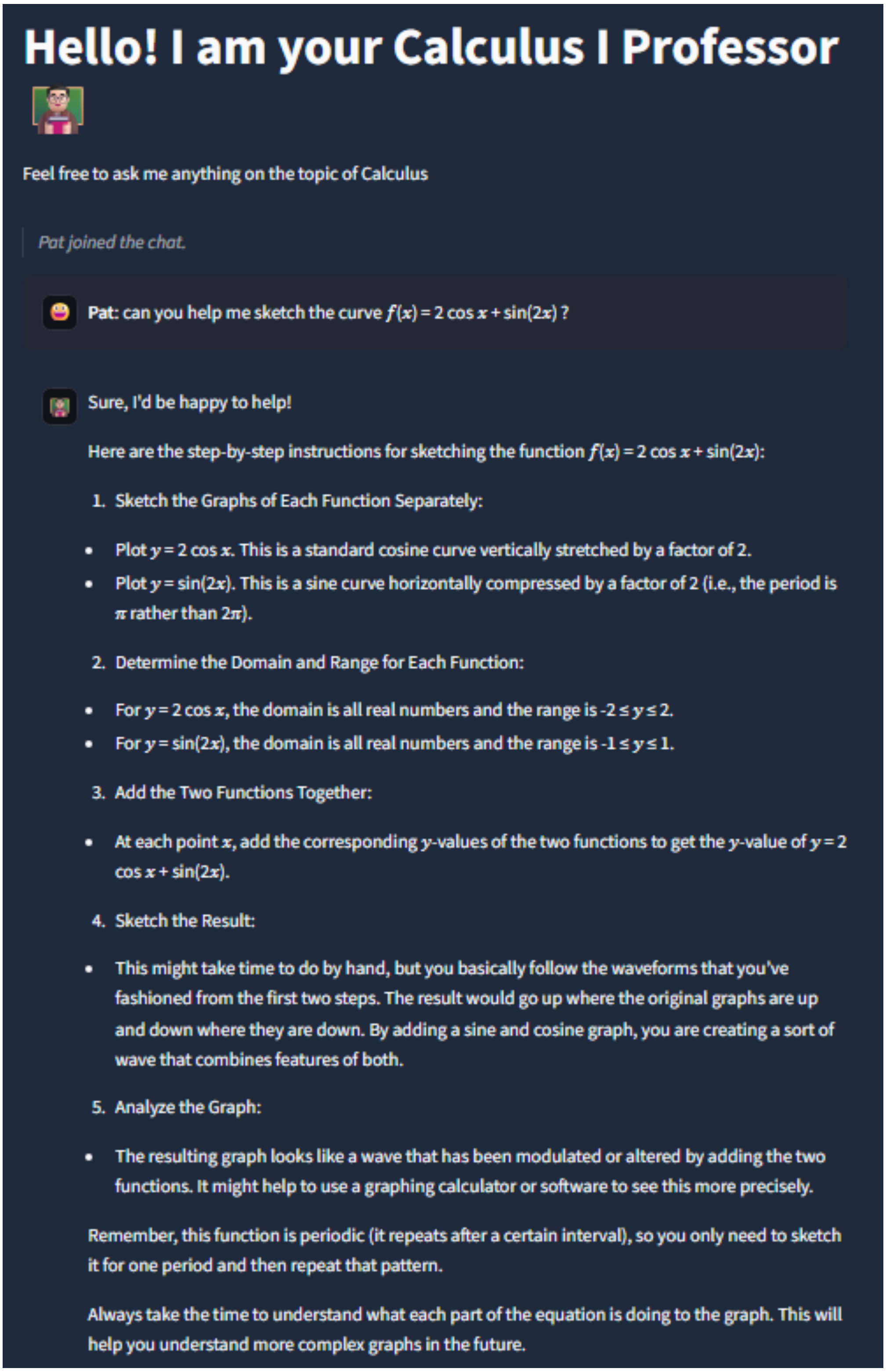
Firstly, Academ requires the user to input their name into the sidebar after the user has clicked any of the options — Calculus I Professor, Com Prog Professor, Physics I Professor, and Project Manager — then, the user can select emojis representing their mood for the day. Once that is finished, the user will be able to type in their desired inputs corresponding to the professors/manager they choose to seek advice from. After that, the user has to press the 'join' button below the emoji section in order to let their consultation to begin. The output will respond to the user's inputs using the OpenAI chatbot to provide clear, concise answers and assistance; however, the chatbot only answers questions related to the topic: Calculus I Professor can not teach the user python lessons, and vice versa.


Display name
Pat
How are you feeling?
😛

Display name, mood selection, and join button


can you help me sketch the curve of $f(x) = 2\cos x + \sin(2x)$?

Inputted question



Hello! I am your Calculus I Professor

Feel free to ask me anything on the topic of Calculus

> *Pat joined the chat.*

Pat: can you help me sketch the curve $f(x) = 2\cos x + \sin(2x)$?

Sure, I'd be happy to help!

Here are the step-by-step instructions for sketching the function $f(x) = 2\cos x + \sin(2x)$:

1. Sketch the Graphs of Each Function Separately:

- Plot $y = 2\cos x$. This is a standard cosine curve vertically stretched by a factor of 2.
- Plot $y = \sin(2x)$. This is a sine curve horizontally compressed by a factor of 2 (i.e., the period is π rather than 2π).

2. Determine the Domain and Range for Each Function:

- For $y = 2\cos x$, the domain is all real numbers and the range is $-2 \leq y \leq 2$.
- For $y = \sin(2x)$, the domain is all real numbers and the range is $-1 \leq y \leq 1$.

3. Add the Two Functions Together:

- At each point x, add the corresponding y-values of the two functions to get the y-value of $y = 2\cos x + \sin(2x)$.

4. Sketch the Result:

- This might take time to do by hand, but you basically follow the waveforms that you've fashioned from the first two steps. The result would go up where the original graphs are up and down where they are down. By adding a sine and cosine graph, you are creating a sort of wave that combines features of both.

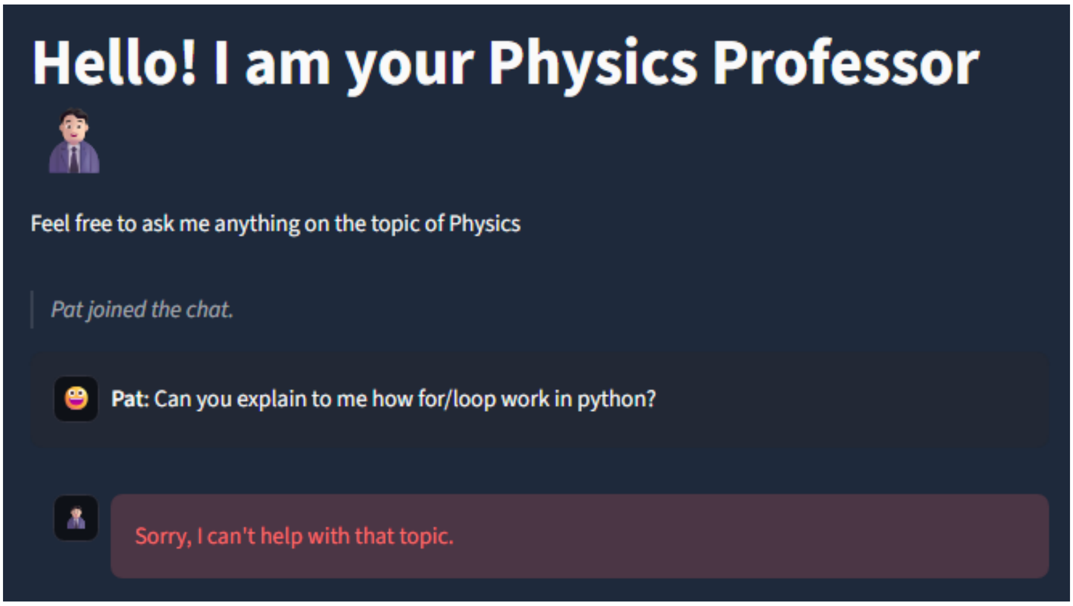
5. Analyze the Graph:

- The resulting graph looks like a wave that has been modulated or altered by adding the two functions. It might help to use a graphing calculator or software to see this more precisely.

Remember, this function is periodic (it repeats after a certain interval), so you only need to sketch it for one period and then repeat that pattern.

Always take the time to understand what each part of the equation is doing to the graph. This will help you understand more complex graphs in the future.

Output: questions related to the professor's lesson



Hello! I am your Physics Professor

Feel free to ask me anything on the topic of Physics

Pat joined the chat.

Pat: Can you explain to me how for/loop work in python?

Sorry, I can't help with that topic.

Output: questions not related to the professor's lesson

Details of Libraries Used

Table 1 Libraries and Their functionalities

Library	**Purpose / Functionality**
Streamlit	Website design using the Python framework
OpenAI	Communication and providing information for user

Functionality of libraries include:

Streamlit:

- Simplifies development of interactive web applications.
- Provides a responsive and navigable interface for users.
- Includes built-in UI components like buttons, text boxes, and messages.
- Reduces the need for complex web development skills.

OpenAI:

- Provides AI models to understand and generate human-like text.
- Powers natural language processing for chatbots.
- Connects via API to evaluate input and generate responses.
- Enhances chatbot intelligence, accuracy, and user interaction.

Streamlit provides a simple, user-friendly interface, while OpenAI enables the chatbot to understand questions and generate accurate, human-like responses. This allows developers to focus on Python functions while delivering a clear and helpful experience for students.

Code Walkthrough

Overall Architecture

The project contains:

- home.py – Main landing page

Pages folder:

- Calculus Professor.py
- Com Prog Professor.py
- Physics Professor.py
- Project Manager.py
- credit.py

Each page uses Streamlit's st.page link to create a sidebar navigation menu, allowing users to switch between professors easily.

Streamlit setup & layout

Every page begins with:

```
import streamlit as st
from openai import OpenAI

st.set_page_config(page_title="Credits", page_icon="🫶", layout="centered")
```

This imports the necessary library and configures the page’s title, icon, and layout.
Source: *Streamlit — Page Configuration*
https://docs.streamlit.io/library/api-reference/utilities/st.set_page_config

The sidebar is created using:

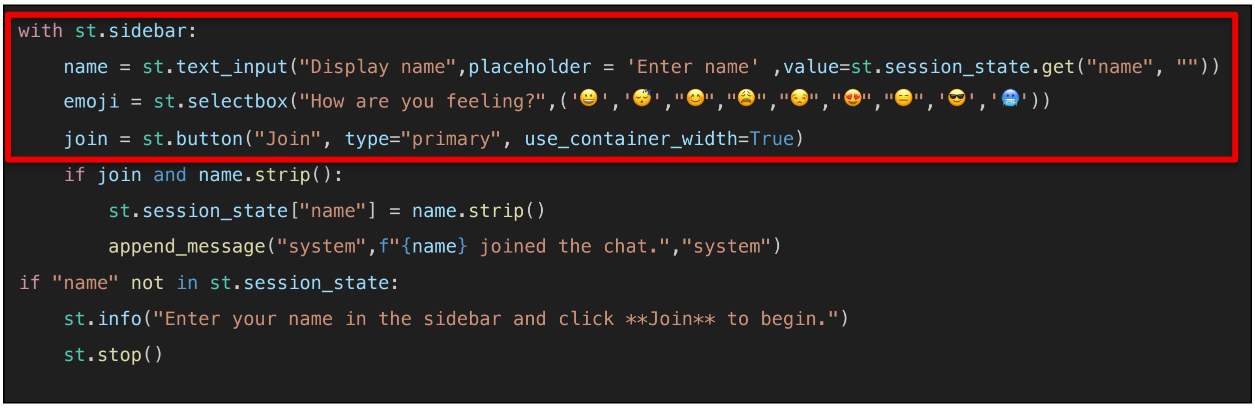
```
with st.sidebar:
    st.page_link("home.py", label="Home", icon="🏛️")
    st.page_link("pages/Calculus Professor.py", label="Calculus I Professor", icon="👨🏻‍🏫")
    st.page_link("pages/Com Prog Professor.py", label="Com Prog Professor", icon="👨🏻‍💻")
    st.page_link("pages/Physics Professor.py", label="Physics I  Professor", icon="👨🏻‍🔬")
    st.page_link("pages/Project Manager.py", label="Project Manager", icon="👨🏼‍🎓")
    st.page_link("pages/credit.py", label="Credit", icon="🫶")
```

st.page_link is Streamlit’s recommended method for custom multipage navigation.
Source: *Streamlit — st.page_link*
https://docs.streamlit.io/develop/api-reference/widgets/st.page_link

User Login ("Join" System)

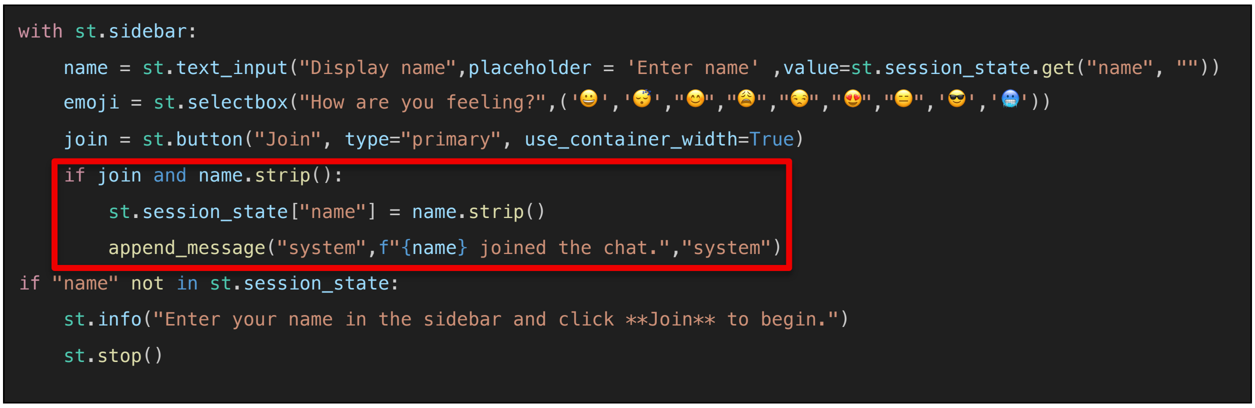
In the sidebar, users enter:

- **Name** – stored as string
- **Emoji representing their mood** – stored as character
- **Join** – stored as boolean



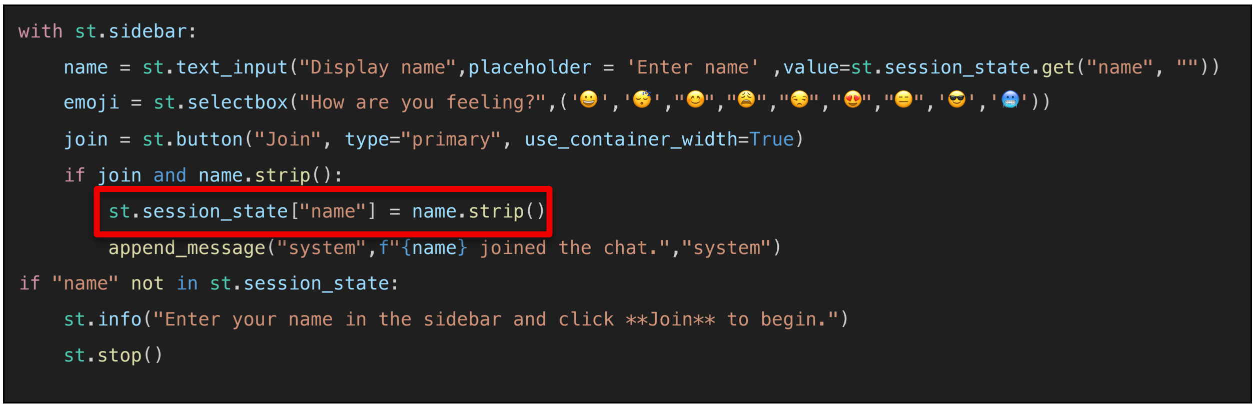
```python
with st.sidebar:
    name = st.text_input("Display name",placeholder = 'Enter name' ,value=st.session_state.get("name", ""))
    emoji = st.selectbox("How are you feeling?",('😀','😴',"😊","😩","😒","😍","😑",'😎','🥶'))
    join = st.button("Join", type="primary", use_container_width=True)
    if join and name.strip():
        st.session_state["name"] = name.strip()
        append_message("system",f"{name} joined the chat.","system")
if "name" not in st.session_state:
    st.info("Enter your name in the sidebar and click **Join** to begin.")
    st.stop()
```

When the name is entered the system append it into st.session_state



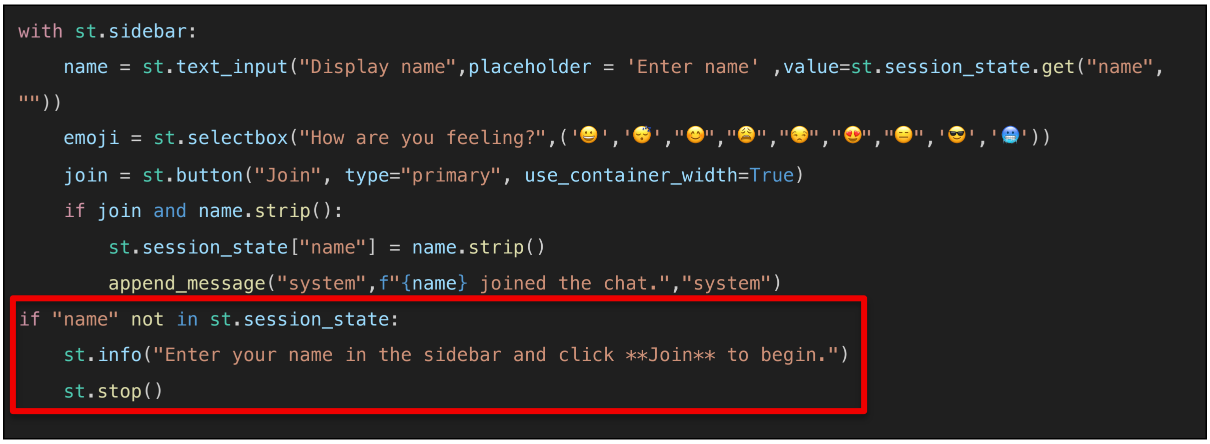
```python
with st.sidebar:
    name = st.text_input("Display name",placeholder = 'Enter name' ,value=st.session_state.get("name", ""))
    emoji = st.selectbox("How are you feeling?",('😀','😴',"😊","😩","😒","😍","😑",'😎','🥶'))
    join = st.button("Join", type="primary", use_container_width=True)
    if join and name.strip():
        st.session_state["name"] = name.strip()
        append_message("system",f"{name} joined the chat.","system")
if "name" not in st.session_state:
    st.info("Enter your name in the sidebar and click **Join** to begin.")
    st.stop()
```

After joining, the name is saved in st.session_state (dict)(Red Box):



```python
with st.sidebar:
    name = st.text_input("Display name",placeholder = 'Enter name' ,value=st.session_state.get("name", ""))
    emoji = st.selectbox("How are you feeling?",('😀','😴',"😊","😩","😒","😍","😑",'😎','🥶'))
    join = st.button("Join", type="primary", use_container_width=True)
    if join and name.strip():
        st.session_state["name"] = name.strip()
        append_message("system",f"{name} joined the chat.","system")
if "name" not in st.session_state:
    st.info("Enter your name in the sidebar and click **Join** to begin.")
    st.stop()
```

If the user has not joined yet, the page stops and wait for user’s name input:

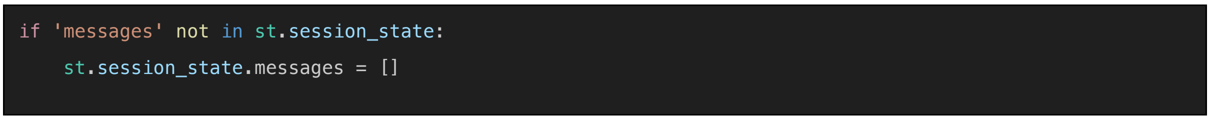


```
with st.sidebar:
    name = st.text_input("Display name",placeholder = 'Enter name' ,value=st.session_state.get("name",
""))
    emoji = st.selectbox("How are you feeling?",('😀','😴',"😊","😩","😒","😍","😑",'😎','🥶'))
    join = st.button("Join", type="primary", use_container_width=True)
    if join and name.strip():
        st.session_state["name"] = name.strip()
        append_message("system",f"{name} joined the chat.","system")
if "name" not in st.session_state:
    st.info("Enter your name in the sidebar and click **Join** to begin.")
    st.stop()
```

session_state is used to preserve information between reruns (important for chat).
Source: *State Management — Streamlit*
https://docs.streamlit.io/library/api-reference/session-state

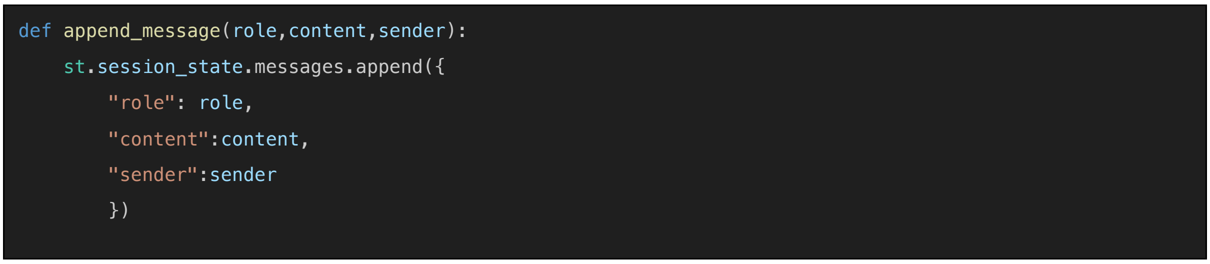
Chat History Storage

All conversation messages are kept inside st.session_state as from stated documentation above:



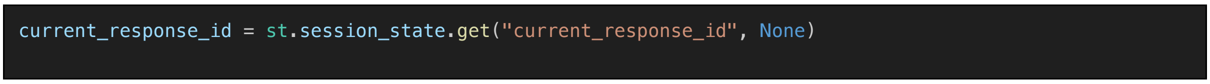
```
if 'messages' not in st.session_state:
    st.session_state.messages = []
```

A helper function appends new messages:



```
def append_message(role,content,sender):
    st.session_state.messages.append({
        "role": role,
        "content":content,
        "sender":sender
        })
```

Response_id is utilized for chat memory during conversations



```
current_response_id = st.session_state.get("current_response_id", None)
```

Chat Interface Rendering

Messages are displayed using Streamlit’s chat components:

```
user_text = st.chat_input("Type your question…")
if user_text:
    # Append user message
    append_message("user",user_text,st.session_state["name"])

    # Display user message
    with st.chat_message("user", avatar=emoji):
        st.markdown(f"**{st.session_state['name']}**: {user_text}")

    # Display assistant response with streaming
    with st.chat_message("Proj Mgr.", avatar="🧑🏼‍🎓"):
        message_placeholder = st.empty()
        full_response = ""
```

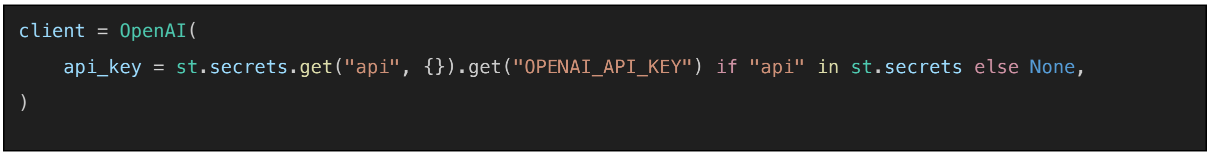
User input is captured using:

```
user_text = st.chat_input("Type your question…")
```

These components come from Streamlit’s Chat Elements API.
Source: *st.chat_message / st.chat_input*
https://docs.streamlit.io/develop/api-reference/chat/st.chat_message

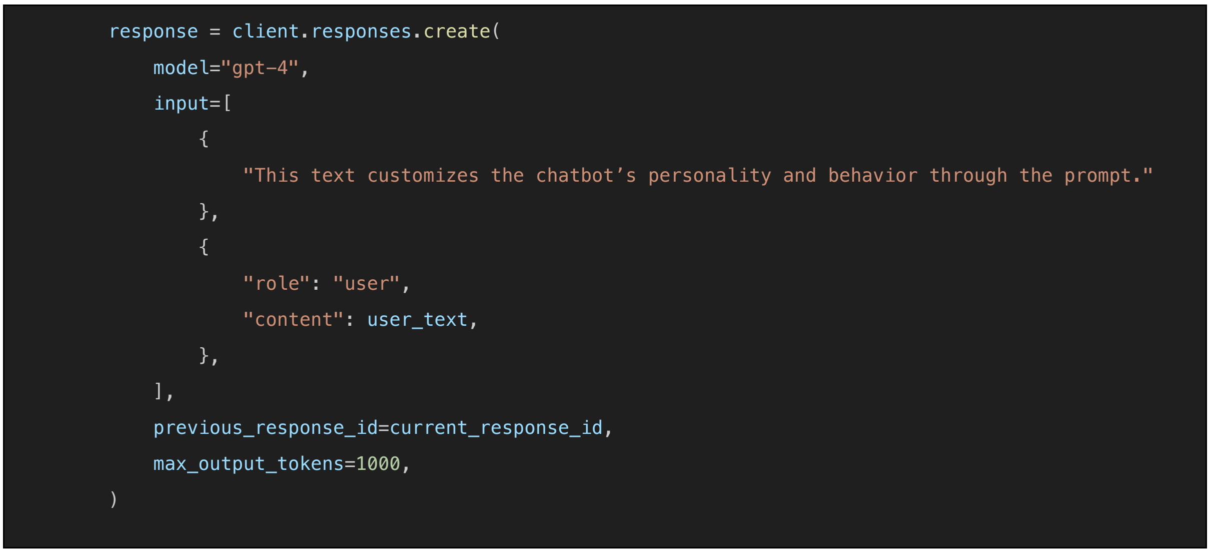
OpenAI Response Generation

Each assistant uses ChatGPT’s API which is secretly stored in secrets.toml:



```
client = OpenAI(
    api_key = st.secrets.get("api", {}).get("OPENAI_API_KEY") if "api" in st.secrets else None,
)
```

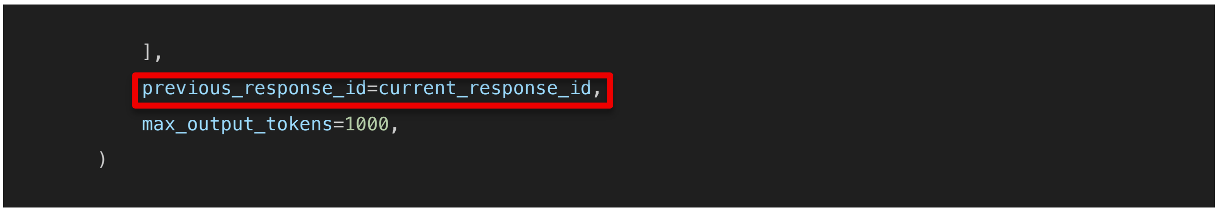
Then sends messages to the model:



```
        response = client.responses.create(
            model="gpt-4",
            input=[
                {
                    "This text customizes the chatbot’s personality and behavior through the prompt."
                },
                {
                    "role": "user",
                    "content": user_text,
                },
            ],
            previous_response_id=current_response_id,
            max_output_tokens=1000,
        )
```

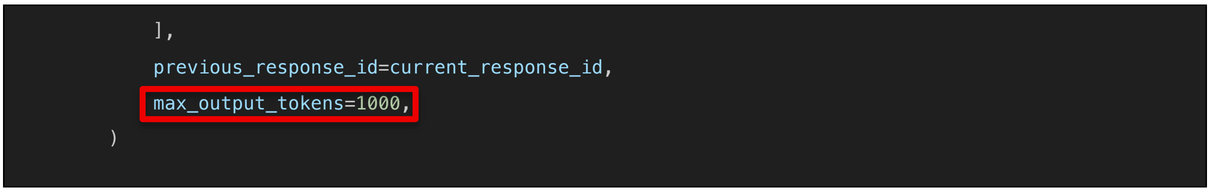
The system prompt (different in each page) defines the *persona* — e.g., Calculus professor, Python tutor, etc.
OpenAI Documentation:
https://platform.openai.com/docs/api-reference/introduction

Previous_response_id is set for chat memory



```
            ],
            previous_response_id=current_response_id,
            max_output_tokens=1000,
        )
```

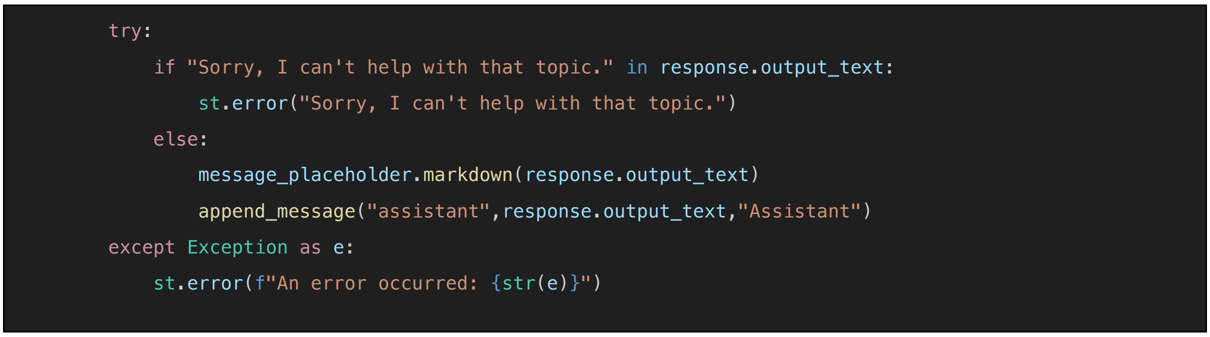
Set max_output_tokens for minimal expense



```
            ],
            previous_response_id=current_response_id,
            max_output_tokens=1000,
        )
```

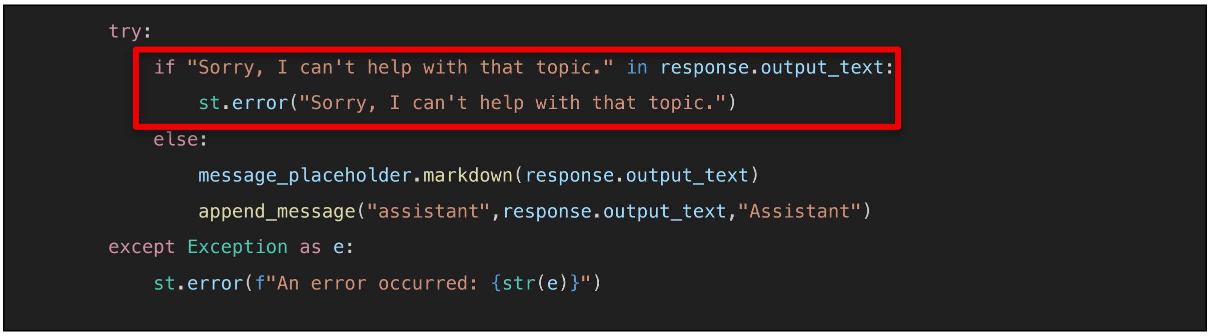
Output Handling

The assistant’s message is shown to the user and stored.



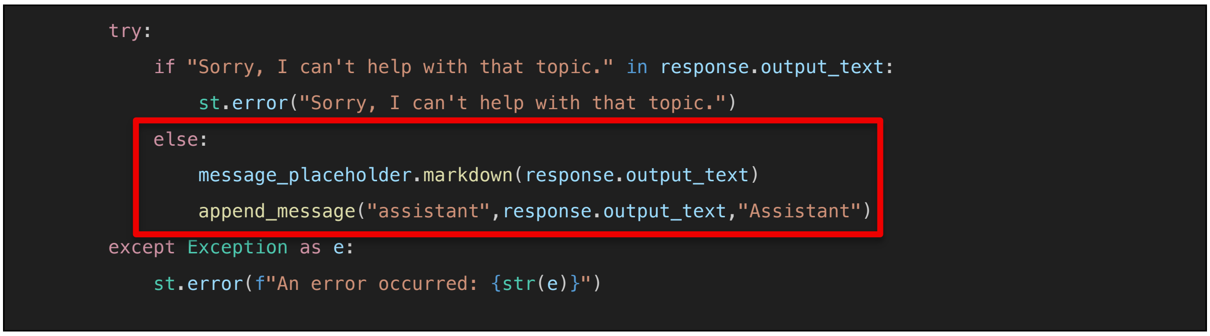
```
try:
    if "Sorry, I can't help with that topic." in response.output_text:
        st.error("Sorry, I can't help with that topic.")
    else:
        message_placeholder.markdown(response.output_text)
        append_message("assistant",response.output_text,"Assistant")
except Exception as e:
    st.error(f"An error occurred: {str(e)}")
```

If the model returns a restricted topic message (e.g., “Sorry, I can't help with that topic”), the app displays:



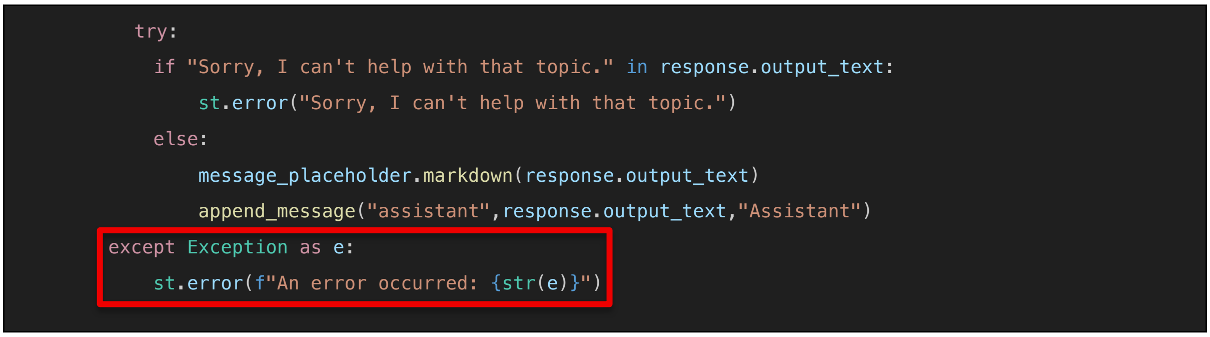
```
try:
    if "Sorry, I can't help with that topic." in response.output_text:
        st.error("Sorry, I can't help with that topic.")
    else:
        message_placeholder.markdown(response.output_text)
        append_message("assistant",response.output_text,"Assistant")
except Exception as e:
    st.error(f"An error occurred: {str(e)}")
```

And append to st.session_state when there is no problem with the output



```
try:
    if "Sorry, I can't help with that topic." in response.output_text:
        st.error("Sorry, I can't help with that topic.")
    else:
        message_placeholder.markdown(response.output_text)
        append_message("assistant",response.output_text,"Assistant")
except Exception as e:
    st.error(f"An error occurred: {str(e)}")
```

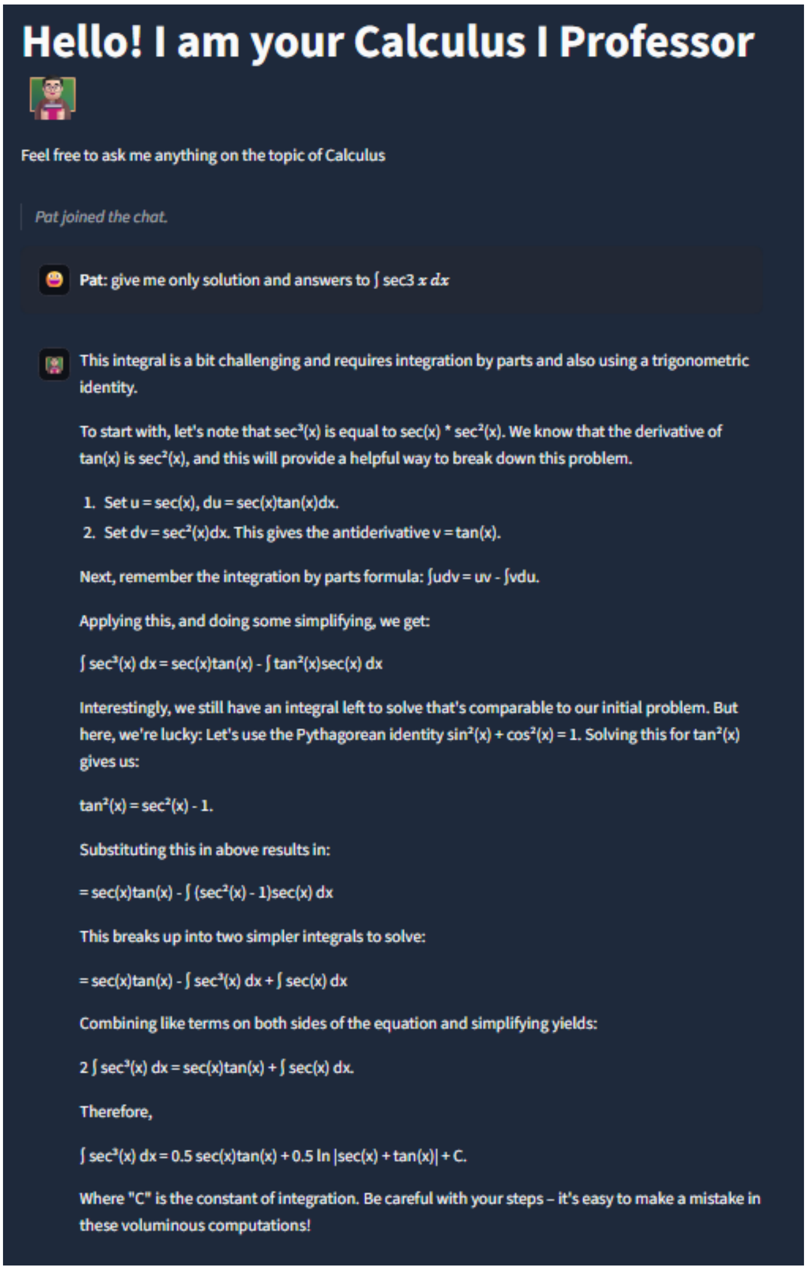
We included the try and except to not let the program crash if an error is encountered



```
  try:
    if "Sorry, I can't help with that topic." in response.output_text:
        st.error("Sorry, I can't help with that topic.")
    else:
        message_placeholder.markdown(response.output_text)
        append_message("assistant",response.output_text,"Assistant")
except Exception as e:
    st.error(f"An error occurred: {str(e)}")
```

Results / Output Examples

The project's result is the response of the chatbot to whatever input the user gives it. Upon a question from the student, the chatbot analyzes the text and provides an appropriate response based on the data and logic stored. Outputs are simple text responses that are relevant to the context and purpose of the query, similar to ChatGPT and other chatbots. In this regard, the aim of the output is to provide Chulalongkorn students with concise, exact, and helpful answers to their questions.



Hello! I am your Calculus I Professor

Feel free to ask me anything on the topic of Calculus

Pat joined the chat.

Pat: give me only solution and answers to ∫ sec3 $x\,dx$

This integral is a bit challenging and requires integration by parts and also using a trigonometric identity.

To start with, let's note that $\sec^3(x)$ is equal to $\sec(x) * \sec^2(x)$. We know that the derivative of $\tan(x)$ is $\sec^2(x)$, and this will provide a helpful way to break down this problem.

1. Set $u = \sec(x)$, $du = \sec(x)\tan(x)dx$.
2. Set $dv = \sec^2(x)dx$. This gives the antiderivative $v = \tan(x)$.

Next, remember the integration by parts formula: $\int udv = uv - \int vdu$.

Applying this, and doing some simplifying, we get:

$\int \sec^3(x)\,dx = \sec(x)\tan(x) - \int \tan^2(x)\sec(x)\,dx$

Interestingly, we still have an integral left to solve that's comparable to our initial problem. But here, we're lucky: Let's use the Pythagorean identity $\sin^2(x) + \cos^2(x) = 1$. Solving this for $\tan^2(x)$ gives us:

$\tan^2(x) = \sec^2(x) - 1$.

Substituting this in above results in:

$= \sec(x)\tan(x) - \int (\sec^2(x) - 1)\sec(x)\,dx$

This breaks up into two simpler integrals to solve:

$= \sec(x)\tan(x) - \int \sec^3(x)\,dx + \int \sec(x)\,dx$

Combining like terms on both sides of the equation and simplifying yields:

$2\int \sec^3(x)\,dx = \sec(x)\tan(x) + \int \sec(x)\,dx$.

Therefore,

$\int \sec^3(x)\,dx = 0.5\sec(x)\tan(x) + 0.5\ln|\sec(x) + \tan(x)| + C$.

Where "C" is the constant of integration. Be careful with your steps – it's easy to make a mistake in these voluminous computations!

Reference

Below are the primary resources used during development. These were essential for learning Streamlit, understanding chat components, and implementing the OpenAI API.

Official Documentation

1. **Streamlit – Page Configuration (st.set_page_config)**
   https://docs.streamlit.io/library/api-reference/utilities/st.set_page_config
2. **Streamlit – Chat Elements (st.chat_message, st.chat_input)**
   https://docs.streamlit.io/develop/api-reference/chat/st.chat_message
3. **Streamlit – Session State**
   https://docs.streamlit.io/library/api-reference/session-state
4. **Streamlit – Multipage Navigation (st.page_link)**
   https://docs.streamlit.io/develop/api-reference/navigation/st.page_link
5. **OpenAI API — Python Client**
   https://platform.openai.com/docs/api-reference/introduction

Tutorials & Learning Resources

- “Streamlit Mini Course – Make Websites with ONLY Python” – Tech with Tim
  https://youtu.be/o8p7uQCGD0U?si=diWG1qA_5og-1Qf1

  “How to Build Chatbot Interfaces with Python” – Dave Ebbelaar
  https://youtu.be/nSeQl6Ehfk0?si=m8ZSc-28lV5a3haI

- “The Only Markdown Crash Course You Will Ever Need” – Web Dev Simplified
  https://youtu.be/_PPWWRV6gbA?si=iv36rU2b1Rwpqra-

These guides informed the chat UI design, state management, and integration of OpenAI’s API.

Declaration

We hereby declare that this project, entitled Academ Chatbot for Chulalongkorn Students, is our own original work conducted as part of the academic requirements of our course. This report and the accompanying project have not been submitted, either in whole or in part, for academic credit in any other course or institution. The project was conceived, designed, and implemented by our team with the aim of developing an intelligent chatbot to assist Chulalongkorn University students with academic and campus-related inquiries.

The development was carried out using Streamlit as the primary platform for building the user interface, and ChatGPT as the language model responsible for generating natural and contextually relevant responses. All tools, libraries, and references used during the creation of this project have been properly acknowledged. ChatGPT was employed only for idea generation, explanation, and general guidance, while all aspects of system design, programming, integration, and testing were completed independently by the project team. This work represents our collective effort, creativity, and understanding of the concepts learned throughout the course, and we take full responsibility for its authenticity and accuracy.

Requirements

Table 2: How Academ satisfies the necessary requirements

External Libraries	Uses Streamlit + OpenAI
100+ lines of code	Combined code exceeds requirement
5+ functions	Multiple functions for messaging, UI, processing
Python Fundamentals	if-else, loops, lists, dicts, strings
Difficulty	GPT-powered chatbot = maximum 5 pts difficulty
Code quality	Modular, readable, structured
Report requirements	Includes objective, I/O, libraries, code explanation, declaration