GUI BASED EDUCATIONAL CHAT BOT WITH TKINTER

This project is a simple chatbot application using the tkinter library in Python. The chatbot can process user input, extract keywords, evaluate expressions, and display messages with corresponding icons in the chat history. The chatbot also responds to user input based on a predefined set of responses.

Library Imports:

The following libraries are imported in the project:

OS: *Used for operating system dependent functionality*

tkinter*: Python's standard GUI library*

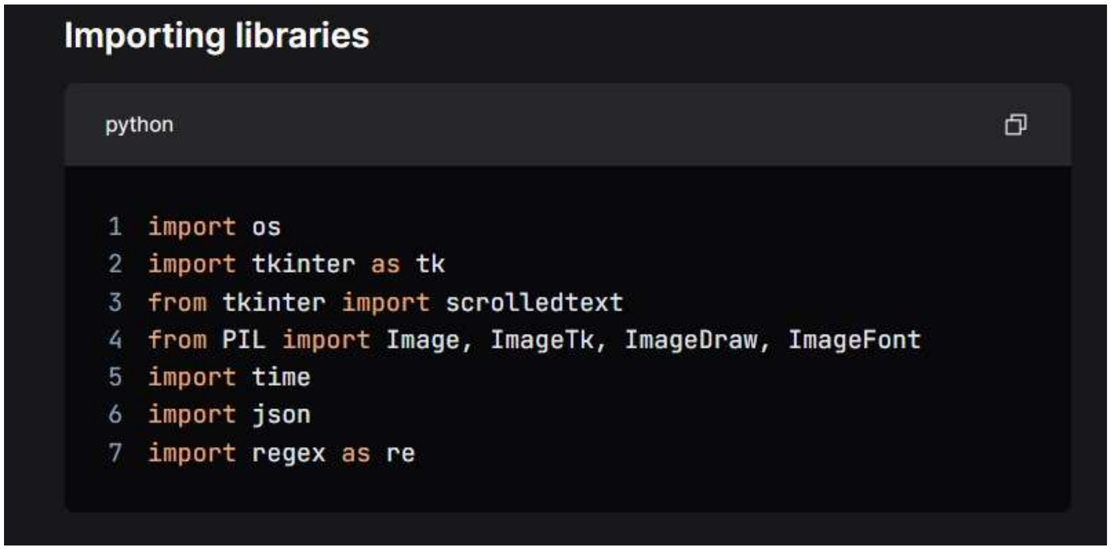
Scrolledtext*: A scrollable text widget from tkinter*

PIL (Pillow): *Used for image processing*

Time*: Used for time-related functionality*

Json*: Used for working with JSON data*

Regex (RE): *Used for regular expression operations*



Importing libraries

python

```python
import os
import tkinter as tk
from tkinter import scrolledtext
from PIL import Image, ImageTk, ImageDraw, ImageFont
import time
import json
import regex as re
```

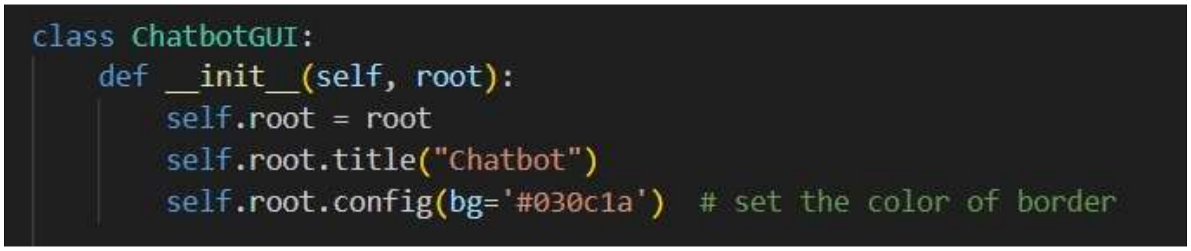
FUNCTION DOCUMENTATION

Class ChatbotGUI:

The ChatbotGUI class is the main component of the project. It handles the GUI setup, user input processing, and chatbot response generation.

init(self, root)

Initializes the ChatbotGUI class, setting up the GUI layout, chat history, user input entry, and Send button. It also initializes the chatbot responses and sets the initial message.



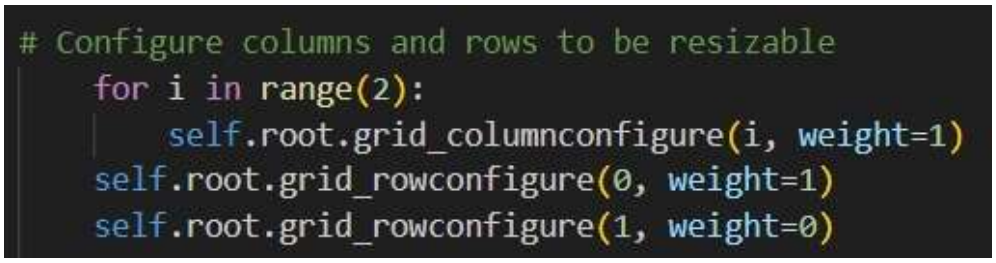
```
class ChatbotGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("Chatbot")
        self.root.config(bg='#030c1a')  # set the color of border
```

Grid_columnconfigure*(self, i, weight=1)*

Configures the grid columns to be resizable.

Grid_rowconfigure(self, row, weight=1)

Configures the grid rows to be resizable.



```
# Configure columns and rows to be resizable
    for i in range(2):
        self.root.grid_columnconfigure(i, weight=1)
    self.root.grid_rowconfigure(0, weight=1)
    self.root.grid_rowconfigure(1, weight=0)
```

Display_initial_message(self)

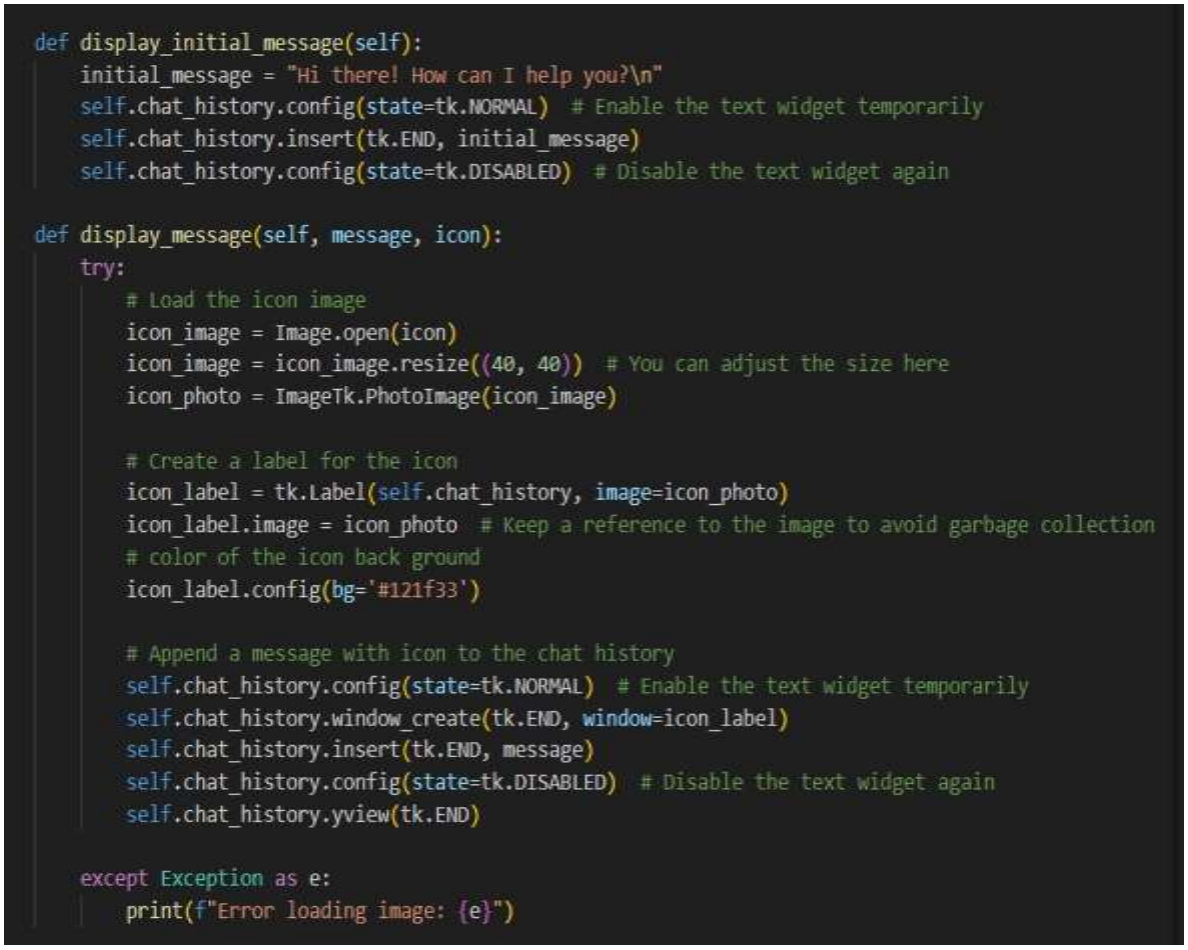
Displays the initial message

"Hi there! How can I help you?"

in the chat history.

Display_message(self, message, icon)

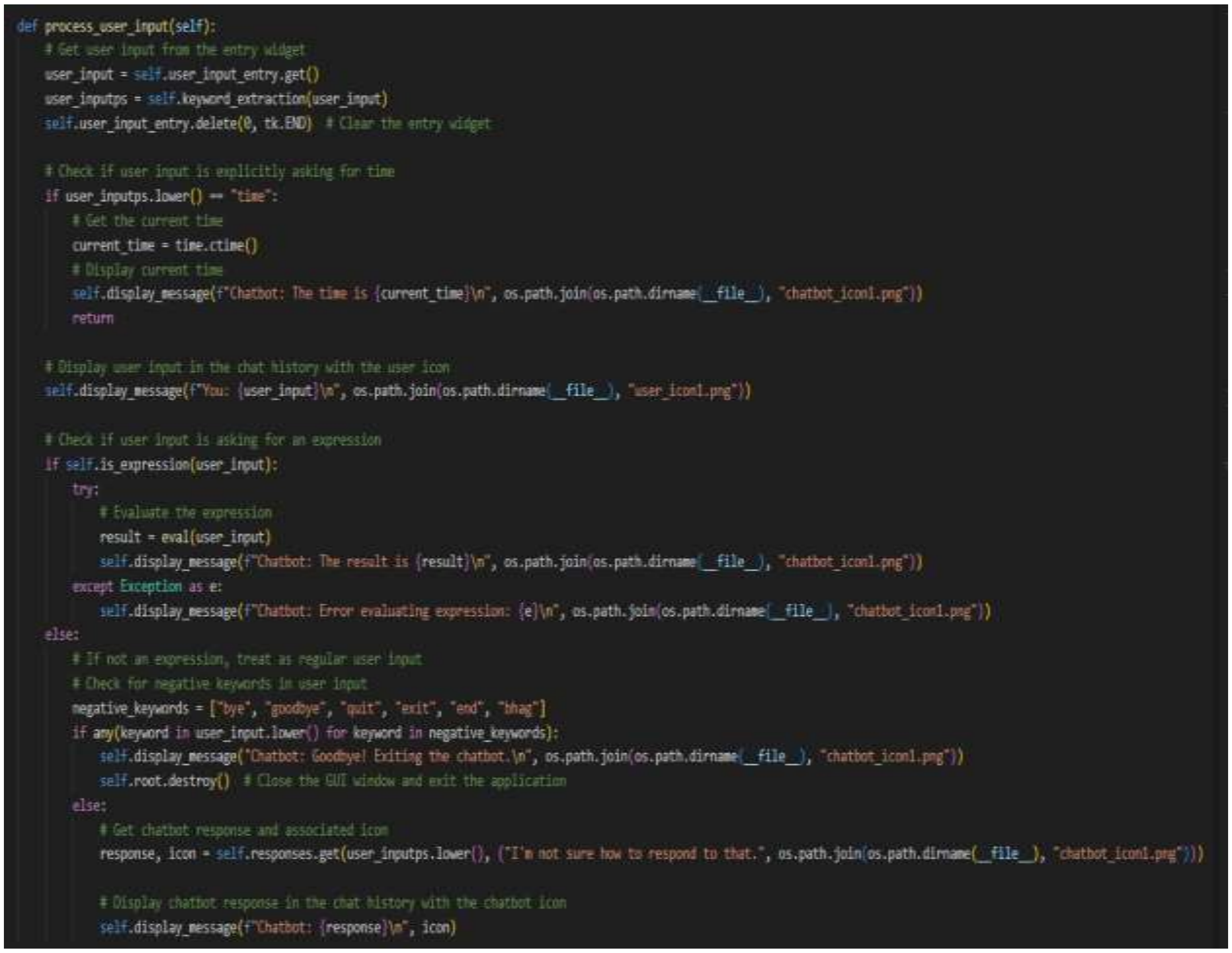
Displays a message with the corresponding icon in the chat history



```python
def display_initial_message(self):
    initial_message = "Hi there! How can I help you?\n"
    self.chat_history.config(state=tk.NORMAL)  # Enable the text widget temporarily
    self.chat_history.insert(tk.END, initial_message)
    self.chat_history.config(state=tk.DISABLED)  # Disable the text widget again

def display_message(self, message, icon):
    try:
        # Load the icon image
        icon_image = Image.open(icon)
        icon_image = icon_image.resize((40, 40))  # You can adjust the size here
        icon_photo = ImageTk.PhotoImage(icon_image)

        # Create a label for the icon
        icon_label = tk.Label(self.chat_history, image=icon_photo)
        icon_label.image = icon_photo  # Keep a reference to the image to avoid garbage collection
        # color of the icon back ground
        icon_label.config(bg='#121f33')

        # Append a message with icon to the chat history
        self.chat_history.config(state=tk.NORMAL)  # Enable the text widget temporarily
        self.chat_history.window_create(tk.END, window=icon_label)
        self.chat_history.insert(tk.END, message)
        self.chat_history.config(state=tk.DISABLED)  # Disable the text widget again
        self.chat_history.yview(tk.END)

    except Exception as e:
        print(f"Error loading image: {e}")
```

<u>*Process_user_input(self)*</u>

Processes user input by getting the user input from the entry widget, extracting keywords, checking if the input is an expression as well as if the user is asking for time and displaying the corresponding chatbot response.



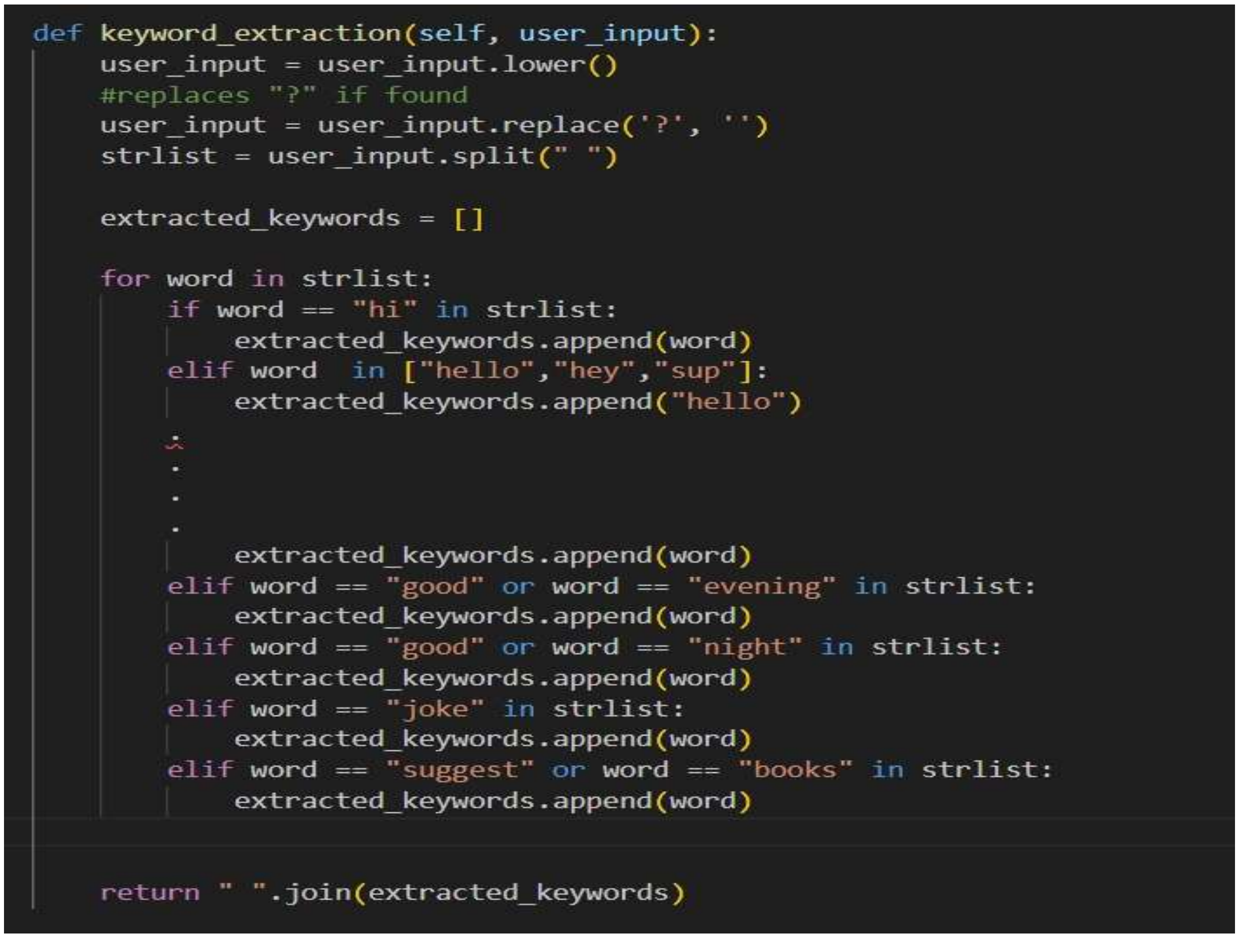
```python
def process_user_input(self):
    # Get user input from the entry widget
    user_input = self.user_input_entry.get()
    user_inputps = self.keyword_extraction(user_input)
    self.user_input_entry.delete(0, tk.END)  # Clear the entry widget

    # Check if user input is explicitly asking for time
    if user_inputps.lower() == "time":
        # Get the current time
        current_time = time.ctime()
        # Display current time
        self.display_message(f"Chatbot: The time is {current_time}\n", os.path.join(os.path.dirname(__file__), "chatbot_icon1.png"))
        return

    # Display user input in the chat history with the user icon
    self.display_message(f"You: {user_input}\n", os.path.join(os.path.dirname(__file__), "user_icon1.png"))

    # Check if user input is asking for an expression
    if self.is_expression(user_input):
        try:
            # Evaluate the expression
            result = eval(user_input)
            self.display_message(f"Chatbot: The result is {result}\n", os.path.join(os.path.dirname(__file__), "chatbot_icon1.png"))
        except Exception as e:
            self.display_message(f"Chatbot: Error evaluating expression: {e}\n", os.path.join(os.path.dirname(__file__), "chatbot_icon1.png"))
    else:
        # If not an expression, treat as regular user input
        # Check for negative keywords in user input
        negative_keywords = ["bye", "goodbye", "quit", "exit", "end", "bhag"]
        if any(keyword in user_input.lower() for keyword in negative_keywords):
            self.display_message("Chatbot: Goodbye! Exiting the chatbot.\n", os.path.join(os.path.dirname(__file__), "chatbot_icon1.png"))
            self.root.destroy()  # Close the GUI window and exit the application
        else:
            # Get chatbot response and associated icon
            response, icon = self.responses.get(user_inputps.lower(), ("I'm not sure how to respond to that.", os.path.join(os.path.dirname(__file__), "chatbot_icon1.png")))

            # Display chatbot response in the chat history with the chatbot icon
            self.display_message(f"Chatbot: {response}\n", icon)
```

Keyword_extraction(self, user_input)

the keyword extraction function takes the user input from the entry widget box and first checks if the user input contains any “?” or not if found the “?” gets removed then the string is converted into list based on the spaces and the availablity ofn certain keywords are checked if found they are appended into an empty list

after that the list is returned as a string and those keywords are matched with the keywords in another JSON file if the kerywords matces the corrosponding output Ism printed

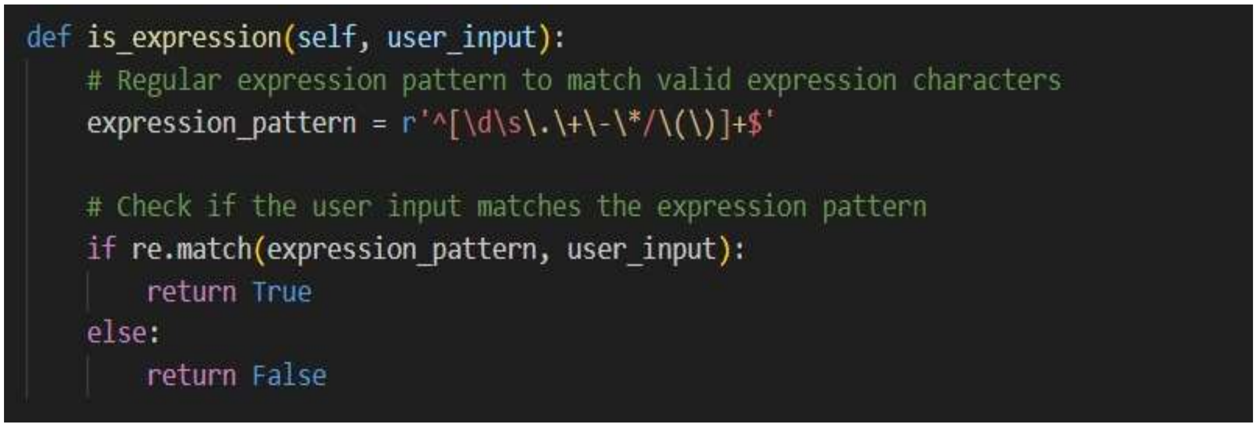
The centeral idea behind this function was to use it as a Natural Language Processing model



```python
def keyword_extraction(self, user_input):
    user_input = user_input.lower()
    #replaces "?" if found
    user_input = user_input.replace('?', '')
    strlist = user_input.split(" ")

    extracted_keywords = []

    for word in strlist:
        if word == "hi" in strlist:
            extracted_keywords.append(word)
        elif word  in ["hello","hey","sup"]:
            extracted_keywords.append("hello")
        .
        .
        .
        .
            extracted_keywords.append(word)
        elif word == "good" or word == "evening" in strlist:
            extracted_keywords.append(word)
        elif word == "good" or word == "night" in strlist:
            extracted_keywords.append(word)
        elif word == "joke" in strlist:
            extracted_keywords.append(word)
        elif word == "suggest" or word == "books" in strlist:
            extracted_keywords.append(word)


    return " ".join(extracted_keywords)
```

is_expression(self, user_input)

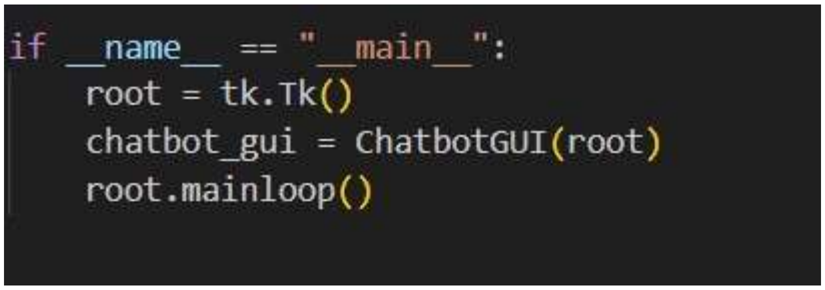
*Checks if the user input is an expression
by matching it against a regular expression pattern.*



```
def is_expression(self, user_input):
    # Regular expression pattern to match valid expression characters
    expression_pattern = r'^[\d\s\.\+\-\*/\(\)]+$'

    # Check if the user input matches the expression pattern
    if re.match(expression_pattern, user_input):
        return True
    else:
        return False
```

The main loop *in a tkinter application, such as this chatbot project, is responsible for handling events and updating the graphical user interface. It keeps the application responsive and enables interactions with the user.*

In this project, the main loop is started with the root.mainloop() statement:



```
if __name__ == "__main__":
    root = tk.Tk()
    chatbot_gui = ChatbotGUI(root)
    root.mainloop()
```

When the mainloop() method is called, it enters the event-driven loop and waits for events such as button clicks, window resizing, or user input. When an event occurs, tkinter processes the event and calls the corresponding event handlers. In this chatbot application, the main loop handles the following events.

User input:

When the user types a message and presses Enter or clicks the Send button, the process_user_input() function is called, which processes the user input and displays the chatbot response.

Window resizing:

When the user resizes the window, the main loop automatically adjusts the layout of the widgets according to the grid configuration rules set up in the _init_() method of the ChatbotGUI class.

Focus management:

When the user clicks or navigates to a different widget, the main loop manages the focus and ensures that the correct widget is active.

Idle time: When there are no events to process, the main loop waits for new events, conserving system resources and reducing CPU usage.

The main loop is essential for a tkinter application, as it manages the event handling, updates the user interface, and keeps the application responsive. By calling root.mainloop(), the chatbot application becomes interactive and ready to receive user input and process events.

Members

Our group members include:

- Om Dey – developed the code
- Satyabrata Das – developed then UIUX
- Piyali Sarkar – entered data feed for the bot
- Tanisha Mallick – entered data feed for the bot
- Dipayan Das – documented and made presentation for the entire project

USES

Right now, our chat bot can response to basic day to day quarries and few BCA related quarries, and it can also tell the time and solve equations

In future we plan to add more features and make the response more dynamic.

---------Thank you---------