TCP chatbot (Assignment 5)

Team members: Saiakhil Kovvur, Kevin Maldjian

A simple chatbot using sockets in python and Tkinter (Python's GUI tool)

Our chatbot will be able to provide information on a variety of topics such as **weather**, **jokes**, **songs** and **dinner ideas**. The chatbot will feature a user-friendly GUI created using Tkinter.

Libraries used: `socket, Tkinter, Thread, sys, time, Chatterbot`

client.py

`def receive()`: This function calls the `recv()` function on the client's socket connection to listen for a response from the server and uses the `decode()` method to decode the response in `"utf8"` encoding. Uses `time.sleep(1)` that delays the displaying of the reply from the server to give it a flow. After receiving the response, it inserts it into the message list of the GUI created by Tkinter.

`def send()`: Gets the message to send to the server bot using `my_msg.get()` method of the GUI. Inserts the message into the GUI's message list using msg_list.insert() method, clears the input field of GUI using `my_msg.set("")` and sends the message to the server using socket's *send()* function encoding it into bytes.

`def exited()`: When the user closes the GUI, this function will handle closing all socket connections. Calls the `close()` method on the socket and `quit()` method on the GUI.

`tkinter.Tk()` - creates the GUI instance

`set()` - sets the message in the input field

`tkinter.Listbox(frame, height, width)` - creates the GUI frame that contains the messages and responses.

`tkinter.Button(GUI, text="Send", command=send)` - Creates the send button on the GUI

`tkinter.mainloop()` - Runs the GUI

`server_port and server_host` - Terminal inputs. We use these to create the socket using `socket(AF_INET, SOCK_STREAM)` and connect it to the host using `connect((server_host,server_port))`.

`receive_thread = Thread(target=receive)` and `receive_thread.start()` - Used to create a multi-threading environment to allow multiple users to take part.

Server.py

Establishing the server for clients:

The logic of the server is fairly simple: On startup, an instance of the chatterbot is created and the port is set from the argument given in the terminal. A socket is opened using the socket.socket and is set to the name `tcpServer`. This server then binds to localhost and the provided port. Now that the socket is bound it calls `tcpServer.listen(1)` to wait for incoming connections.

Messaging the client

Once a connection is established through using `tcpServer.accept()` the chatbot will send an initial welcome message to the user. From then on, the server waits for incoming messages and once one is received it parses the input to figure out which function to call. Once the correct response to the previous message is ready, the message is encoded using `.encode()` and then sent back to the client using the `.sendAll()` function

Types of messages

`dinnerIdeas()` - This returns a random choice from list of dinner ideas. It is called when user types “Dinner ideas”

`currentTime()` - This returns the current hour, minute, and second of the current time by using `datetime.now()`. Can be invoked when the user types “Time”

`randJoke()` -This function returns a joke and is called when the client sends the message “Joke”.

Default

If none of these commands are typed, the user will get a generated response from the Chatterbot library.

Chatterbot usage

Chatterbot stores the bots responses within a SQLite3 database. The database entries were created once we trained the chatterbot by using the function `.set_trainer()` on our chatbot object. When a user talks to the bot, the responses are fetched from the `db.sqlite3` file and then returned to the client.