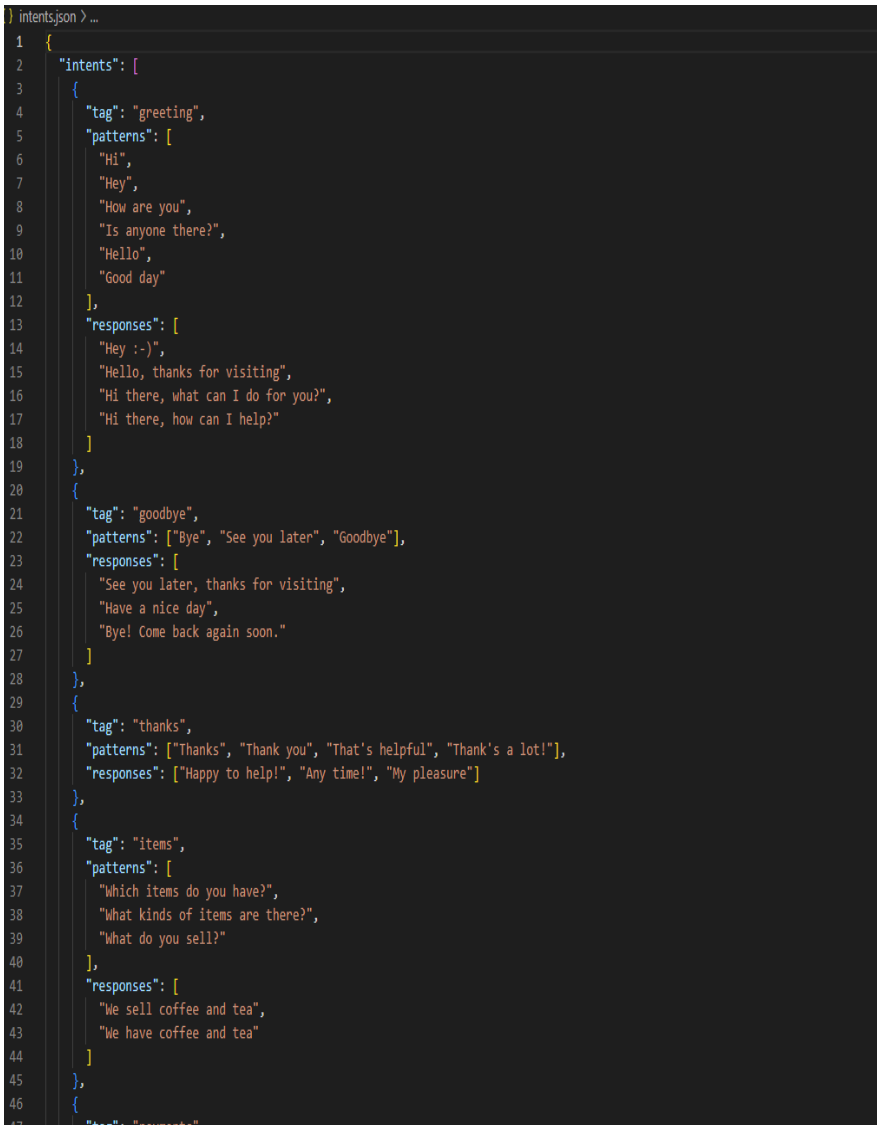
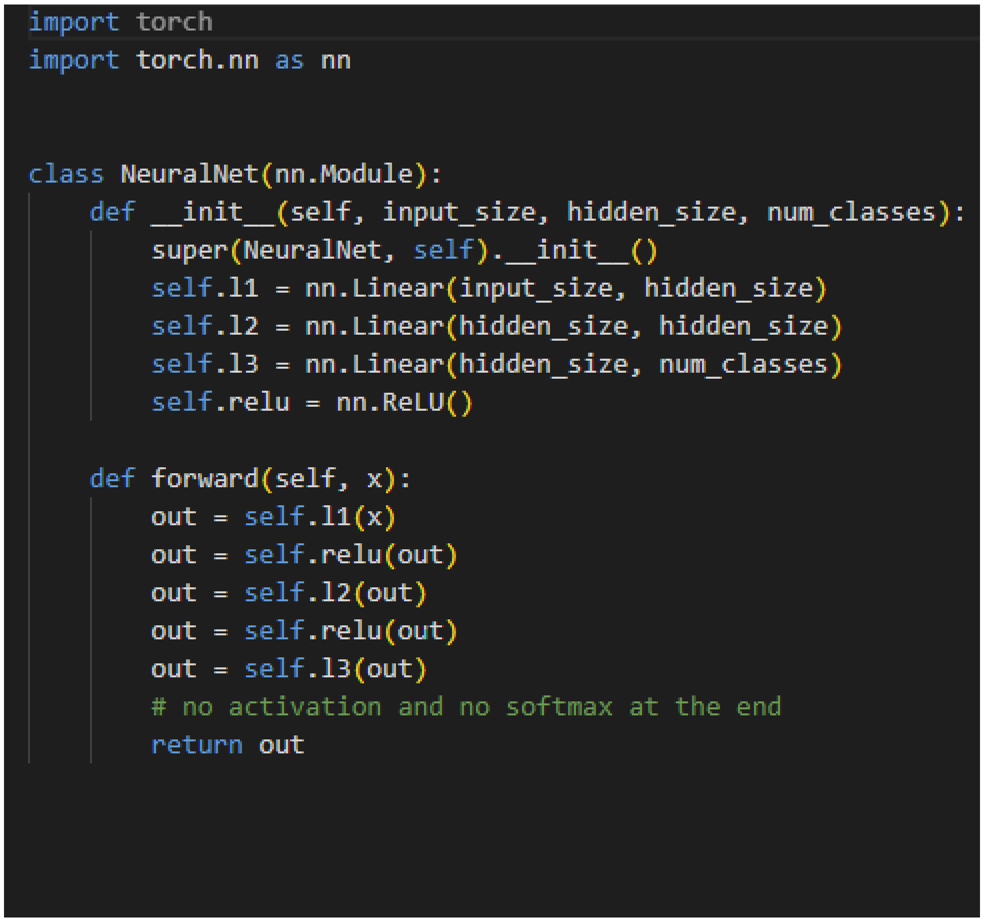
Name: Kang(Kevin) Wang
Submission date: 5/29/2023
Submitted to: Assignment “Week4: Deployment on Flask”

Toy data



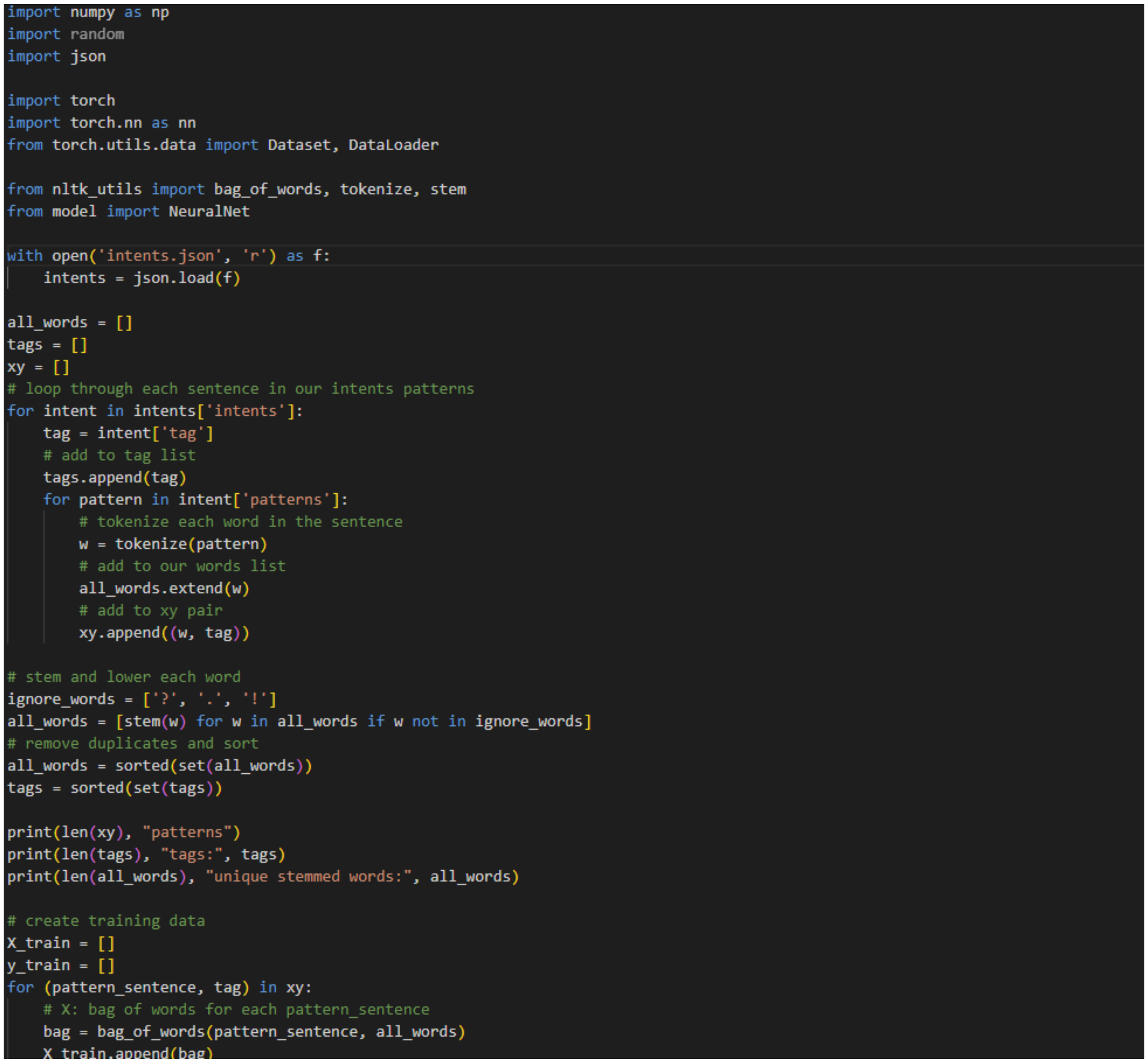
```json
{
  "intents": [
    {
      "tag": "greeting",
      "patterns": [
        "Hi",
        "Hey",
        "How are you",
        "Is anyone there?",
        "Hello",
        "Good day"
      ],
      "responses": [
        "Hey :-)",
        "Hello, thanks for visiting",
        "Hi there, what can I do for you?",
        "Hi there, how can I help?"
      ]
    },
    {
      "tag": "goodbye",
      "patterns": ["Bye", "See you later", "Goodbye"],
      "responses": [
        "See you later, thanks for visiting",
        "Have a nice day",
        "Bye! Come back again soon."
      ]
    },
    {
      "tag": "thanks",
      "patterns": ["Thanks", "Thank you", "That's helpful", "Thank's a lot!"],
      "responses": ["Happy to help!", "Any time!", "My pleasure"]
    },
    {
      "tag": "items",
      "patterns": [
        "Which items do you have?",
        "What kinds of items are there?",
        "What do you sell?"
      ],
      "responses": [
        "We sell coffee and tea",
        "We have coffee and tea"
      ]
    },
    {
```

Create model structure



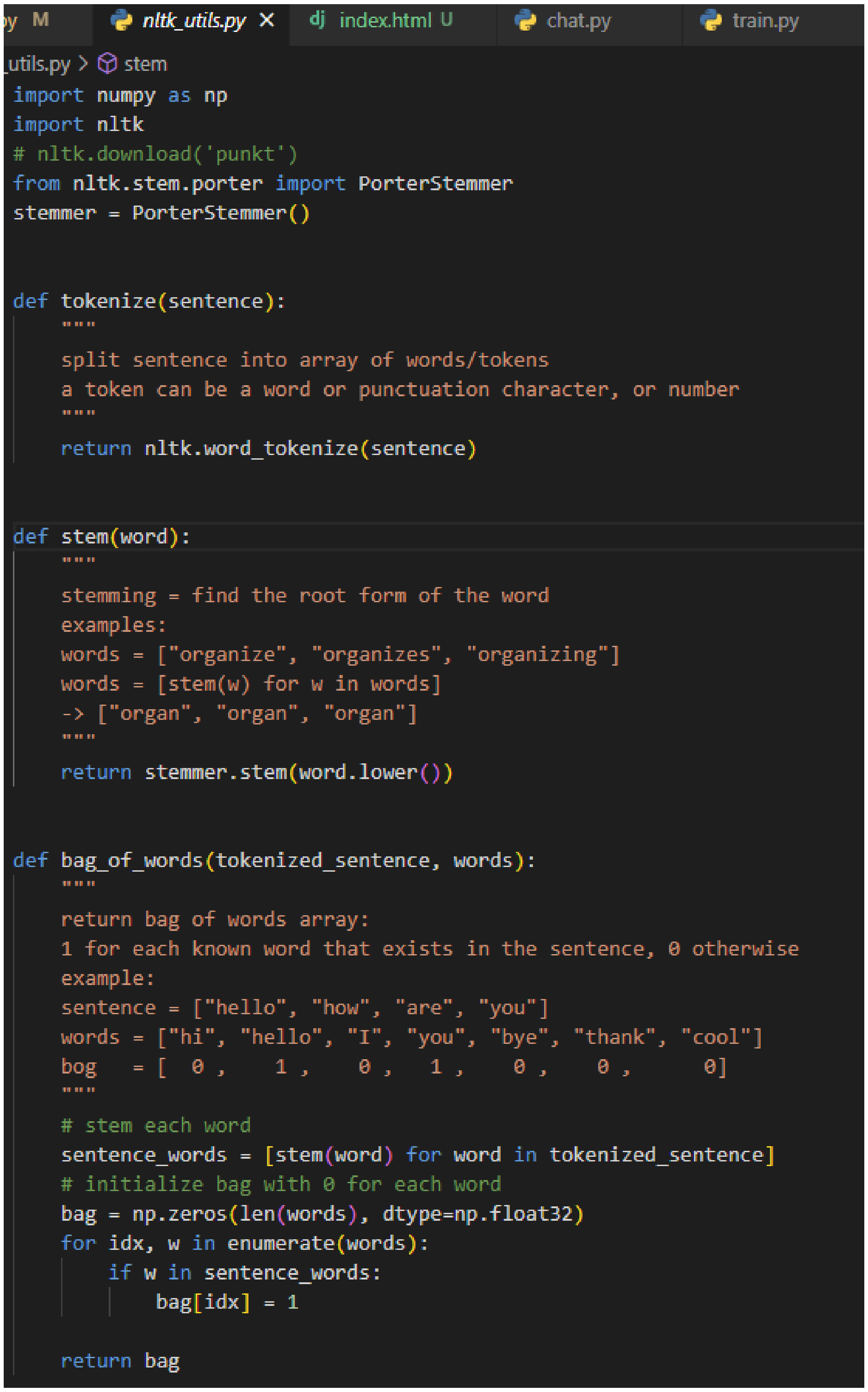
```python
import torch
import torch.nn as nn


class NeuralNet(nn.Module):
    def __init__(self, input_size, hidden_size, num_classes):
        super(NeuralNet, self).__init__()
        self.l1 = nn.Linear(input_size, hidden_size)
        self.l2 = nn.Linear(hidden_size, hidden_size)
        self.l3 = nn.Linear(hidden_size, num_classes)
        self.relu = nn.ReLU()

    def forward(self, x):
        out = self.l1(x)
        out = self.relu(out)
        out = self.l2(out)
        out = self.relu(out)
        out = self.l3(out)
        # no activation and no softmax at the end
        return out
```

Train the model



```
import numpy as np
import random
import json

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from nltk_utils import bag_of_words, tokenize, stem
from model import NeuralNet

with open('intents.json', 'r') as f:
    intents = json.load(f)

all_words = []
tags = []
xy = []
# loop through each sentence in our intents patterns
for intent in intents['intents']:
    tag = intent['tag']
    # add to tag list
    tags.append(tag)
    for pattern in intent['patterns']:
        # tokenize each word in the sentence
        w = tokenize(pattern)
        # add to our words list
        all_words.extend(w)
        # add to xy pair
        xy.append((w, tag))

# stem and lower each word
ignore_words = ['?', '.', '!']
all_words = [stem(w) for w in all_words if w not in ignore_words]
# remove duplicates and sort
all_words = sorted(set(all_words))
tags = sorted(set(tags))

print(len(xy), "patterns")
print(len(tags), "tags:", tags)
print(len(all_words), "unique stemmed words:", all_words)

# create training data
X_train = []
y_train = []
for (pattern_sentence, tag) in xy:
    # X: bag of words for each pattern_sentence
    bag = bag_of_words(pattern_sentence, all_words)
    X_train.append(bag)
```

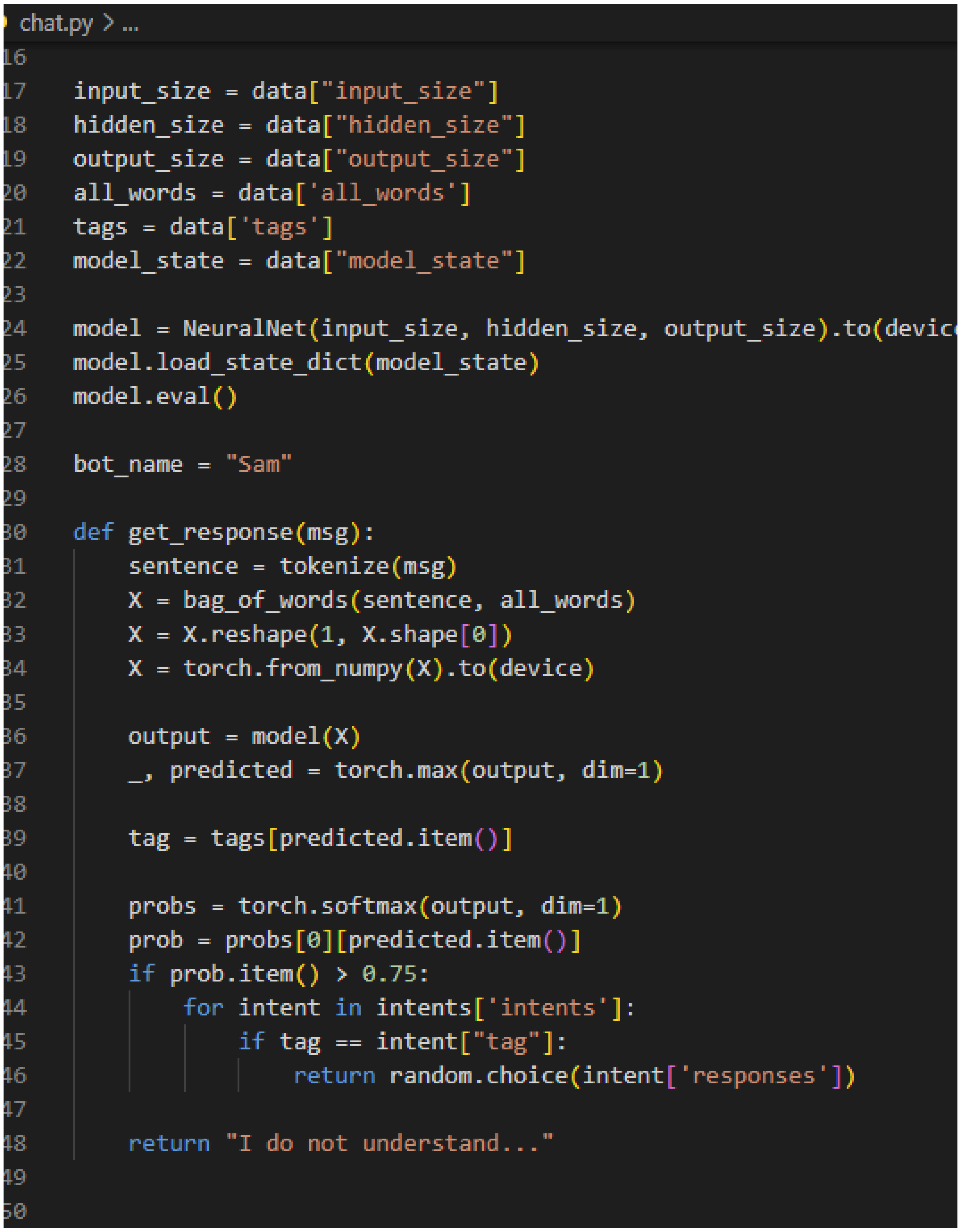
Functions used to preprocess the response from the user and the data in the toy dataset



```python
import numpy as np
import nltk
# nltk.download('punkt')
from nltk.stem.porter import PorterStemmer
stemmer = PorterStemmer()


def tokenize(sentence):
    """
    split sentence into array of words/tokens
    a token can be a word or punctuation character, or number
    """
    return nltk.word_tokenize(sentence)


def stem(word):
    """
    stemming = find the root form of the word
    examples:
    words = ["organize", "organizes", "organizing"]
    words = [stem(w) for w in words]
    -> ["organ", "organ", "organ"]
    """
    return stemmer.stem(word.lower())


def bag_of_words(tokenized_sentence, words):
    """
    return bag of words array:
    1 for each known word that exists in the sentence, 0 otherwise
    example:
    sentence = ["hello", "how", "are", "you"]
    words = ["hi", "hello", "I", "you", "bye", "thank", "cool"]
    bog   = [  0 ,    1 ,    0 ,   1 ,    0 ,    0 ,      0]
    """
    # stem each word
    sentence_words = [stem(word) for word in tokenized_sentence]
    # initialize bag with 0 for each word
    bag = np.zeros(len(words), dtype=np.float32)
    for idx, w in enumerate(words):
        if w in sentence_words:
            bag[idx] = 1

    return bag
```

Saved trained model

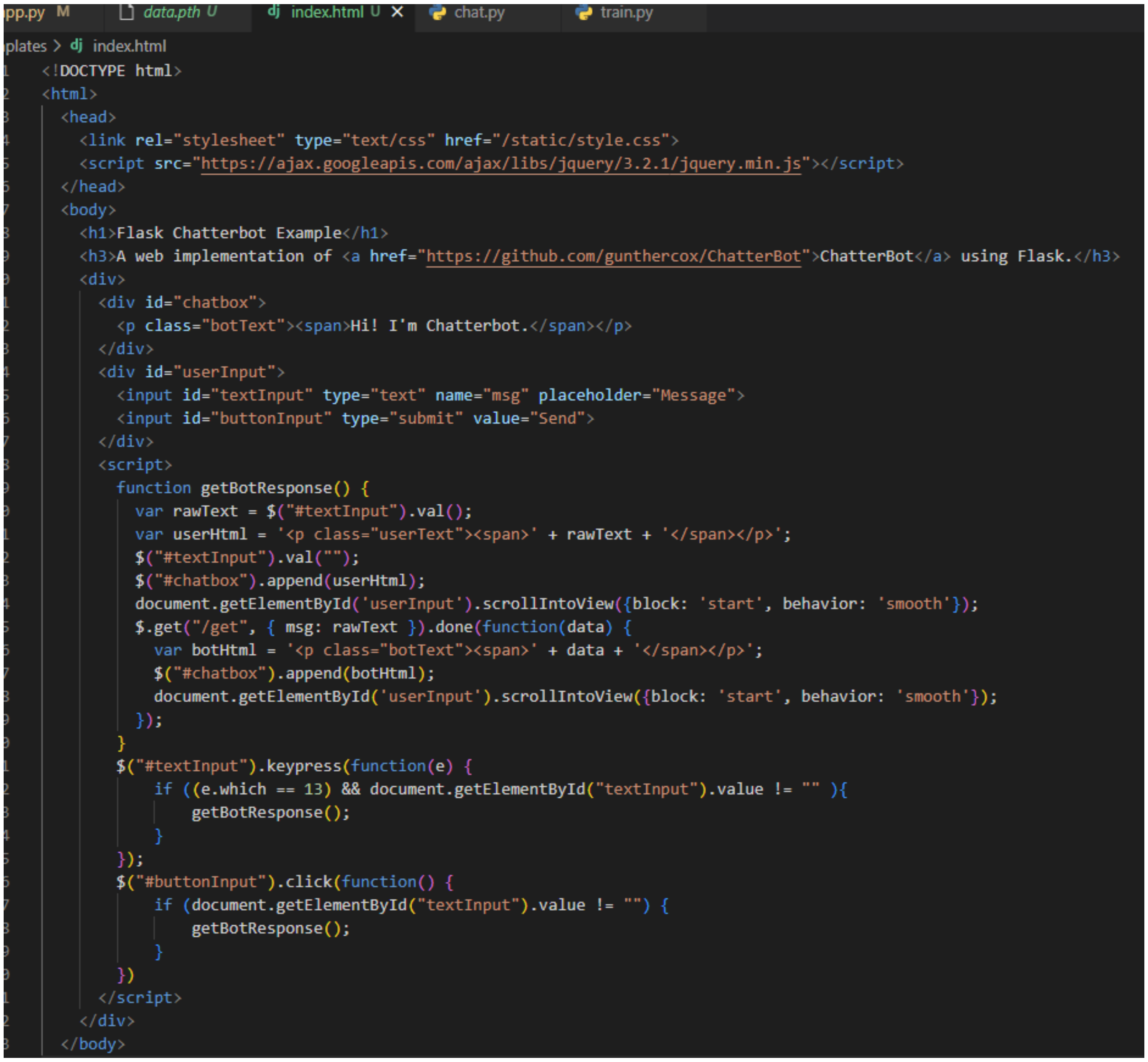
Function to generate response



chat.py > ...

```
input_size = data["input_size"]
hidden_size = data["hidden_size"]
output_size = data["output_size"]
all_words = data['all_words']
tags = data['tags']
model_state = data["model_state"]

model = NeuralNet(input_size, hidden_size, output_size).to(devic
model.load_state_dict(model_state)
model.eval()

bot_name = "Sam"

def get_response(msg):
    sentence = tokenize(msg)
    X = bag_of_words(sentence, all_words)
    X = X.reshape(1, X.shape[0])
    X = torch.from_numpy(X).to(device)

    output = model(X)
    _, predicted = torch.max(output, dim=1)

    tag = tags[predicted.item()]

    probs = torch.softmax(output, dim=1)
    prob = probs[0][predicted.item()]
    if prob.item() > 0.75:
        for intent in intents['intents']:
            if tag == intent["tag"]:
                return random.choice(intent['responses'])

    return "I do not understand..."
```

Frontend code

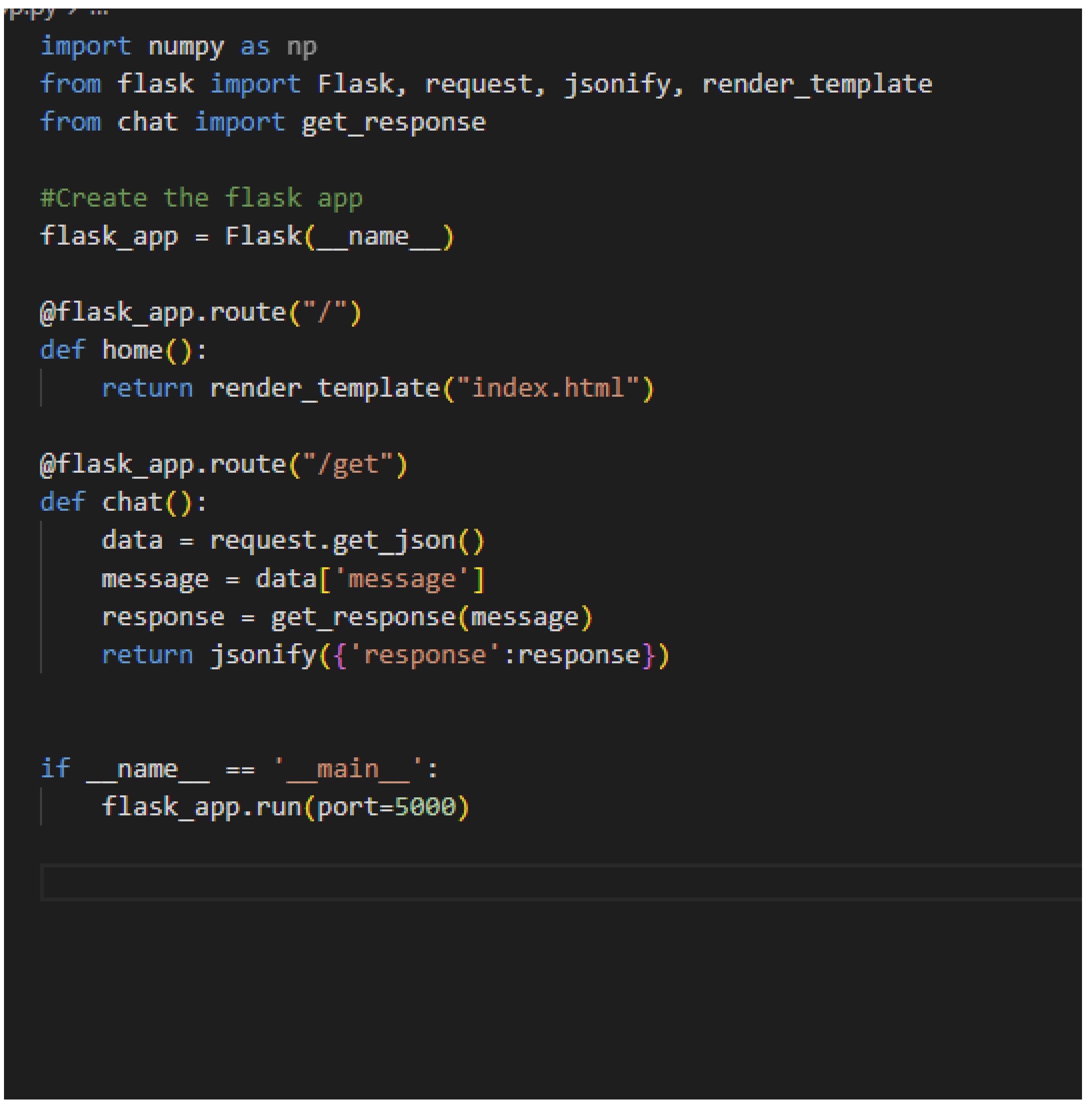


pp.py M | data.pth U | index.html U × | chat.py | train.py

plates > index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="/static/style.css">
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
  </head>
  <body>
    <h1>Flask Chatterbot Example</h1>
    <h3>A web implementation of <a href="https://github.com/gunthercox/ChatterBot">ChatterBot</a> using Flask.</h3>
    <div>
      <div id="chatbox">
        <p class="botText"><span>Hi! I'm Chatterbot.</span></p>
      </div>
      <div id="userInput">
        <input id="textInput" type="text" name="msg" placeholder="Message">
        <input id="buttonInput" type="submit" value="Send">
      </div>
      <script>
        function getBotResponse() {
          var rawText = $("#textInput").val();
          var userHtml = '<p class="userText"><span>' + rawText + '</span></p>';
          $("#textInput").val("");
          $("#chatbox").append(userHtml);
          document.getElementById('userInput').scrollIntoView({block: 'start', behavior: 'smooth'});
          $.get("/get", { msg: rawText }).done(function(data) {
            var botHtml = '<p class="botText"><span>' + data + '</span></p>';
            $("#chatbox").append(botHtml);
            document.getElementById('userInput').scrollIntoView({block: 'start', behavior: 'smooth'});
          });
        }
        $("#textInput").keypress(function(e) {
            if ((e.which == 13) && document.getElementById("textInput").value != "" ){
                getBotResponse();
            }
        });
        $("#buttonInput").click(function() {
            if (document.getElementById("textInput").value != "") {
                getBotResponse();
            }
        })
      </script>
    </div>
  </body>
```

Route connecting backend and frontend



```
import numpy as np
from flask import Flask, request, jsonify, render_template
from chat import get_response

#Create the flask app
flask_app = Flask(__name__)

@flask_app.route("/")
def home():
    return render_template("index.html")

@flask_app.route("/get")
def chat():
    data = request.get_json()
    message = data['message']
    response = get_response(message)
    return jsonify({'response':response})


if __name__ == '__main__':
    flask_app.run(port=5000)
```