

create a document for chatbot using python

720921104094	Sherbin S	Team Leader
720921104070	Nishanth MG	Team Member
720921104119	Viswa Prathap	Team Member
720921104074	Prasanth p	Team Member
720921104072	Pandiselvam R	Team Member

Introduction:

In today's digital age, communication between humans and machines has evolved significantly. Chatbots, powered by artificial intelligence and natural language processing, have emerged as a transformative technology in this domain. These intelligent and conversational agents are designed to interact with users, understand their queries, and provide relevant responses or perform tasks, all in a conversational manner. One of the most popular programming languages for building chatbots is Python.

Problem Definition:

The challenge is to create a chatbot in Python that provides exceptional customer service, answering user queries on a website or application. The objective is to deliver high-quality support to users, ensuring a positive user experience and customer satisfaction.

Abstract:

A chatbot enables a user to simply ask questions in the same manner that they would respond to humans. The most well-known chatbots currently are voices chatbots: SIRI and Alexa. However, chatbots have been adopted and brought into the daily application at a high rate on the computer chat platform. NLP also allows computers and algorithms to understand human interactions through various languages. Recent advances in machine learning have greatly improved the accurate and effective of natural language processing, making chatbots a viable option for many organizations

Design Thinking:



Edit with WPS Office

1.Functionality:

They can automate routine tasks, freeing up time for your human workforce to handle more complex tasks. Data Collection: AI chatbots can collect valuable data from customer interactions, providing insights into customer behavior and needs.

2.User Interface:

A *chatbot user interface* (UI) is part of a chatbot that users see and interact with. This can include anything from the text on a screen to the buttons and menus that are used to control a chatbot. The chatbot UI is what allows users to send messages and tell it what they want it to do.

3. Natural Language Processing (NLP):

Natural Language Processing (NLP) is a subfield of artificial intelligence (AI). It helps machines process and understand the human language so that they can automatically perform repetitive tasks. Examples include machine translation, summarization, ticket classification, and spell check.

4.Responses:

Chatbot responses are messages that a chatbot sends to the user. Chatbots can be powered by pre-programmed responses or artificial intelligence and natural language processing. Based on the applied mechanism, they process human language to understand user queries and deliver matching answers.

5.Chatbot integration:

Chatbots can be integrated with various communication channels such as websites, social media platforms, messaging apps, and voice assistants

- Chatbot integration can be achieved through APIs, webhooks, and third-party services such as Zapier
- Chatbots can be integrated with customer relationship management (CRM) systems, marketing automation tools, and other business software to streamline workflows and improve customer service
- Chatbot integration can help businesses automate repetitive tasks, improve customer engagement, and reduce response time .

6.Testing and Improvement:



Edit with WPS Office

Chatbot testing is an essential process that ensures the chatbot's functionality, reliability, and performance. Chatbot testing can be done using various techniques such as Natural Language Processing (NLP) testing, End-to-End (E2E) testing, Voice testing, Performance testing, Security testing, and Monitoring . Chatbot testing frameworks can be categorized into three main divisions: Expected Scenarios, Possible Scenarios, and Almost Impossible Scenarios.

When it comes to chatbot improvement, it's essential to track the chatbot's performance over time and set up viable goals for the chatbot . Chatbots can be improved by analyzing user feedback and interactions and updating the chatbot's knowledge base accordingly.

Development:

Prerequisites

Before you begin, ensure you have the following:

- Python (3.x) installed on your system.
- Flask library installed.
- A provided dataset of chat messages.
- Basic knowledge of Python programming.
- Terminal or Command Prompt.

Implementation

Create Project Directory:

Start by creating a project directory for your application.

Create Python Script:

Create a Python script for the chat application. You can use your favorite code editor to create a file, e.g., chat_app.py.

Import Libraries:

Import the necessary libraries at the beginning of your Python script.

```
from flask import Flask, render_template, request
```

Set Up Flask App:

Initialize a Flask web application.

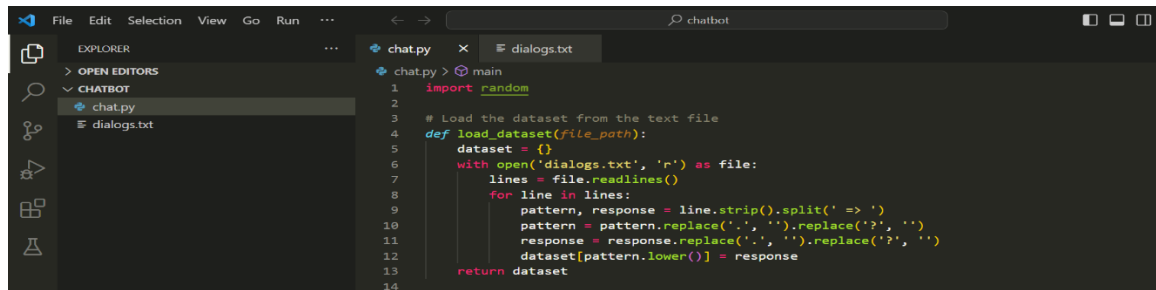
```
app = Flask(__name__)
```



Edit with WPS Office

Load the Dataset:

Load the provided dataset into your Python script, similar to the previous example.



```
1 import random
2
3 # Load the dataset from the text file
4 def load_dataset(file_path):
5     dataset = {}
6     with open('dialogs.txt', 'r') as file:
7         lines = file.readlines()
8         for line in lines:
9             pattern, response = line.strip().split('=> ')
10            pattern = pattern.replace('.', '').replace('?', '')
11            response = response.replace('.', '').replace('?', '')
12            dataset[pattern.lower()] = response
13    return dataset
14
```

Create a Route for Chat Interface:

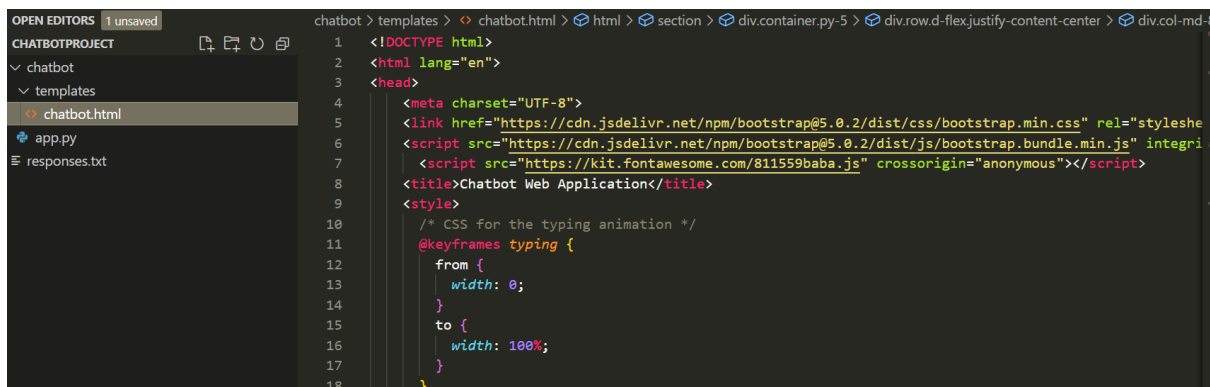
Define a route in your Flask app to render a web page for the chat interface

```
# Route for the chatbot web page
@app.route('/')
def chatbot_page():
    return render_template('chatbot.html')

# Route for receiving user input and providing chatbot responses
@app.route('/get_response', methods=['POST'])
def get_response():
    user_input = request.form['user_input']
    chatbot_response = dataset.get(user_input, "I'm sorry, I don't understand that.")
    return chatbot_response
```

Create a Chat HTML Template:

Create an HTML template for the chat interface. You can use the Jinja2 template engine to render chat messages on the web page.



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4     <meta charset="UTF-8">
5     <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">
6     <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js" integrity="sha384-MQLwNjkR7W6JmQ835t0Z16R066Q4382388" crossorigin="anonymous"></script>
7     <script src="https://kit.fontawesome.com/811559baba.js" crossorigin="anonymous"></script>
8     <title>Chatbot Web Application</title>
9     <style>
10        /* CSS for the typing animation */
11        @keyframes typing {
12            from {
13                width: 0;
14            }
15            to {
16                width: 100%;
17            }
18        }
19    </style>
```

Run the Flask App:

Run your Flask application.



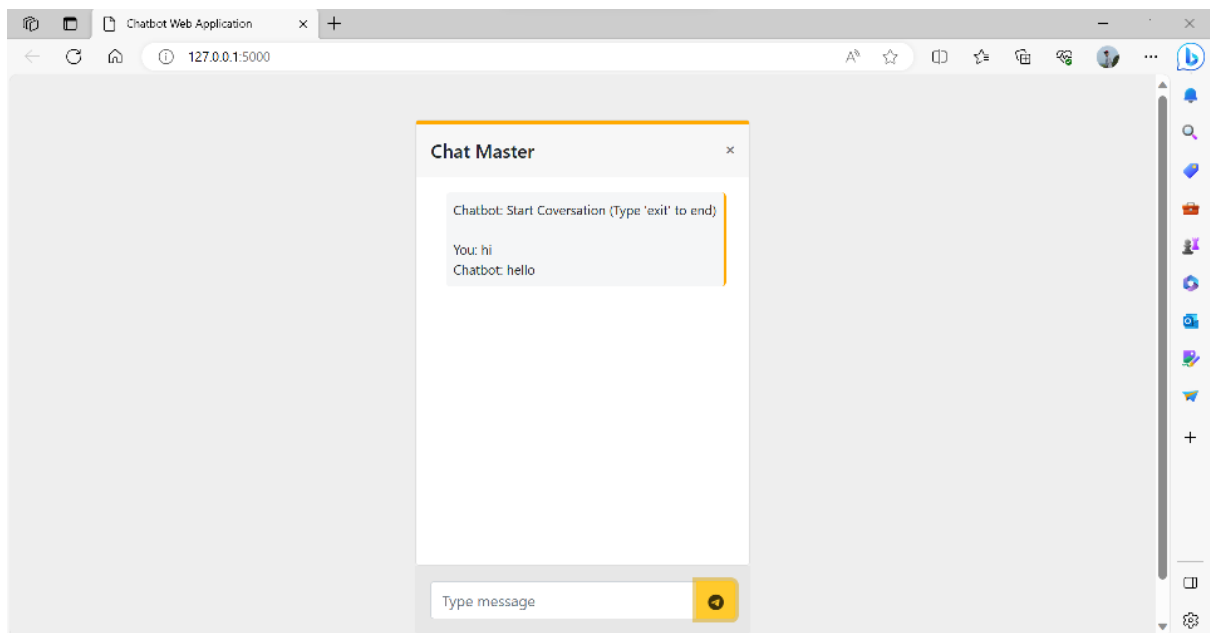
Edit with WPS Office

```
# Route for receiving user input and providing chatbot responses
@app.route('/get_response', methods=['POST'])
def get_response():
    user_input = request.form['user_input']
    chatbot_response = dataset.get(user_input, "I'm sorry, I don't understand that.")
    return chatbot_response

if __name__ == '__main__':
    app.run(debug=True)
```

Testing:

Open a web browser and navigate to <http://127.0.0.1:5000/> to access the chat interface. You should see the chat messages from the provided dataset displayed on the web page.



Improvements:

You can extend this application by allowing user input, interactive chat features, and real-time updates.

Sample code:

App.py

```
from flask import Flask, render_template, request

app = Flask(__name__)

# Load responses from the text file
def load_responses():
    dataset = {}
    with open('responses.txt', 'r') as file:
```



Edit with WPS Office

```

lines = file.readlines()
for line in lines:
    pattern, response = line.strip().split('=>')
    pattern = pattern.replace('.', '').replace('?', '')
    response = response.replace('.', '').replace('?', '')
    dataset[pattern.lower()] = response
return dataset

dataset = load_responses()

# Route for the chatbot web page
@app.route('/')
def chatbot_page():
    return render_template('chatbot.html')

# Route for receiving user input and providing chatbot responses
@app.route('/get_response', methods=['POST'])
def get_response():
    user_input = request.form['user_input']
    chatbot_response = dataset.get(user_input, "I'm sorry, I don't understand that.")
    return chatbot_response

if __name__ == '__main__':
    app.run(debug=True)

```

Chatbot Html File

```

<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
EVSTQN3/azprG1Anm3QDgpJLIm9Nao0Yz1ztcQTwFspd3yD65VohhpuuCOmLASjC"
crossorigin="anonymous">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"
integrity="sha384-
MrcW6ZMFYlzcLA8Nl+NtUVF0sA7MsXsP1UyJoMp4YLEuNSfAP+JcXn/tWtIaxVXM"
crossorigin="anonymous"></script>
    <script src="https://kit.fontawesome.com/811559baba.js"
crossorigin="anonymous"></script>
<title>Chatbot Web Application</title>
<style>
    /* CSS for the typing animation */
    @keyframes typing{
        from {
            width: 0;
        }
        to {
            width: 100%;
        }
    }

    .typing-animation {

```



Edit with WPS Office

```

display: inline-block;
overflow: hidden;
white-space: nowrap;
border-right: 2px solid #ffa900; /* Blinking cursor */
padding-right: 3px; /* Spacing for cursor */
animation: typing 3s steps(30, end);
}
</style>
</head>
<section style="background-color: #eee; height: 600px;">
  <div class="container py-5">

    <div class="row d-flex justify-content-center">
      <div class="col-md-8 col-lg-6 col-xl-4">

        <div class="card">
          <div class="card-header d-flex justify-content-between align-items-center p-3"
            style="border-top: 4px solid #ffa900;">
            <h5 class="mb-0">Chat Master</h5>
            <div class="d-flex flex-row align-items-center">
              <i class="fas fa-times text-muted fa-xs"></i>
            </div>
          </div>
          <div class="card-body" data-mdb-perfect-scrollbar="true" style="position: relative; height:
auto">

            <div class="d-flex justify-content-between">
              <p class="typing-animation small p-2 ms-3 mb-3 rounded-3 " style="background-color:
#f5f6f7;" id="chat-output">
                Chatbot: Start Coversation (Type 'exit' to end)
                <br>
                <br>
              </p>

            </div>
            <br>
            <br> <br>
            <br> <br>
            <br> <br>
            <br> <br>
            <br> <br>

          </div>
          <div class="card-footer text-muted d-flex justify-content-start align-items-center p-3">
            <div class="input-group mb-0">
              <input type="text" class="form-control" id="user-input" placeholder="Type message"
                aria-label="Recipient's username" aria-describedby="button-addon2" />
              <button class="btn btn-warning" type="submit" id="send-button" style="padding-top:
.55rem;">
                <i class="fa-brands fa-telegram fa-beat-fade" value="PLAY" onclick="play()"></i>
                <audio id="audio" src="https://s27.aconvert.com/convert/p3r68-cdx67/c4lpg-
az7kc.mp3"></audio>
              </button>
            </div>
          </div>
        </div>
      </div>
    </div>
  </div>

```



Edit with WPS Office

```

</div>

</div>
</div>

</div>
</section>
<body>
<script>
    const chatOutput = document.getElementById('chat-output');
    const userInput = document.getElementById('user-input');
    const sendButton = document.getElementById('send-button');

    sendButton.addEventListener('click', function () {
        function play() {
            var audio = document.getElementById("audio");
            audio.play();
        }

        const message = userInput.value;
        if(message == 'exit')
        {
            window.location.reload("Refresh")
            alert("Your Conversation ends")
        }
        var audio = new Audio('sound.mp3');
        audio.play();
        if (message.trim() !== "") {
            appendMessage('You: ' + message);
            userInput.value = "";

            // Send user input to the server and get chatbot response
            fetch('/get_response', {
                method: 'POST',
                body: new URLSearchParams({ 'user_input': message }),
            })
                .then(response => response.text())
                .then(data => {
                    appendMessage('Chatbot: ' + data);
                });
        }
    });

    function appendMessage(message) {
        const messageElement = document.createElement('div');
        messageElement.textContent = message;
        chatOutput.appendChild(messageElement);
    }
</script>
</body>
</html>

```

Conclusion



Edit with WPS Office

This document provides a Python-powered chatbots are a vital force in modern communication, offering efficiency and enhanced user experiences. The versatility of Python, coupled with its rich libraries, makes it a prime choice for chatbot development. By harnessing the power of datasets, natural language processing, and thoughtful design, you can craft chatbots that understand and engage users effectively.

These intelligent agents have found applications across diverse industries, from customer support to e-commerce, delivering round-the-clock assistance. The chatbot landscape is dynamic and holds vast potential for innovation. By embracing Python for chatbot development, you can be a part of the transformative future of human-machine interactions, simplifying tasks and enriching user interactions.



Edit with WPS Office