CREATING A CHATBOT USING PYTHON

TEAM MEMBER

510521104047: SIVASAKTHI C

PHASE-1: PROJECT

OBJECTIVES:

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.

PROBLEM DEFINITION:

The problem is to build an AI-powered diabetes prediction system that uses machine learning algorithms to analyze medical data and predict the likelihood of an individual developing diabetes. The system aims to provide early risk assessment and personalized preventive measures, allowing individuals to take proactive actions to manage their health.

DESIGN THINKING:

- **Functionality**: Define the scope of the chatbot's abilities, including answering common questions, providing guidance, and directing users to appropriate resources.
- **User Interface**: Determine where the chatbot will be integrated (website, app) and design a user-friendly interface for interactions.
- **Natural Language Processing (NLP):** Implement NLP techniques to understand and process user input in a conversational manner.
- **Responses:** Plan responses that the chatbot will offer, such as accurate answers, suggestions, and assistance.
- **Integration**: Decide how the chatbot will be integrated with the website or app.
- **Testing and Improvement**: Continuously test and refine the chatbot's performance based on user interactions.

DATA SET LINK:

https://www.kaggle.com/datasets/grafstor/simple-dialogs-for-chatbot

ABSTRACT :

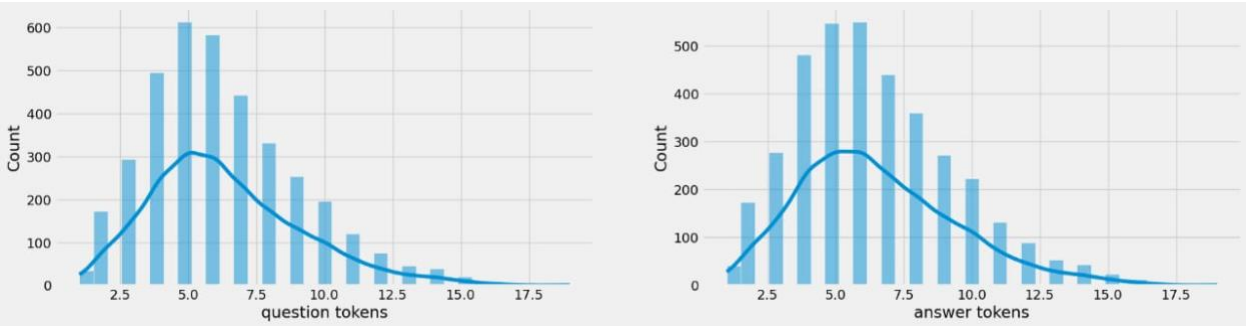
A chatbot is a computer software program that conducts a conversation via auditory or textual methods. This software is used to perform tasks such as quickly responding to users, informing them, helping to purchase products and providing better service to customers. Chatbots are programs that work on Artificial Intelligence (AI) & Machine Learning Platform. Chatbot has become more popular in business groups right now as it can reduce customer service costs and handles multiple users at a time. But yet to accomplish many tasks there is a need to make chatbots as efficient as possible. In this project, we provide the design of a chatbot, which provides a genuine and accurate

answer for any query using Artificial Intelligence Markup Language (AIML) and Latent Semantic Analysis (LSA) with python platform.

DATA PROCESSING:

Data visualization:

```
df['question tokens']=df['question'].apply(lambda x:len(x.split()))
df['answer tokens']=df['answer'].apply(lambda x:len(x.split()))
plt.style.use('fivethirtyeight')
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
sns.set_palette('Set2')
sns.histplot(x=df['question tokens'],data=df,kde=True,ax=ax[0])
sns.histplot(x=df['answer tokens'],data=df,kde=True,ax=ax[1])
sns.jointplot(x='question tokens',y='answer
tokens',data=df,kind='kde',fill=True,cmap='YlGnBu')
plt.show()
```



Text cleaning:

```
def clean_text(text):
    text=re.sub('-',' ',text.lower())
    text=re.sub('[.]',' . ',text)
    text=re.sub('[1]',' 1 ',text)
    text=re.sub('[2]',' 2 ',text)
```

```
    text=re.sub('[3]',' 3 ',text)
    text=re.sub('[4]',' 4 ',text)
    text=re.sub('[5]',' 5 ',text)
    text=re.sub('[6]',' 6 ',text)
    text=re.sub('[7]',' 7 ',text)
    text=re.sub('[8]',' 8 ',text)
    text=re.sub('[9]',' 9 ',text)
    text=re.sub('[0]',' 0 ',text)
    text=re.sub('[,]',' , ',text)
    text=re.sub('[?]',' ? ',text)
    text=re.sub('[!]',' ! ',text)
    text=re.sub('[$]',' $ ',text)
    text=re.sub('[&]',' & ',text)
    text=re.sub('[/]',' / ',text)
    text=re.sub('[:]',' : ',text)
    text=re.sub('[;]',' ; ',text)
    text=re.sub('[*]',' * ',text)
    text=re.sub('[\']',' \' ',text)
    text=re.sub('[\"]',' \" ',text)
    text=re.sub('\t',' ',text)
    return text

df.drop(columns=['answer tokens','question
tokens'],axis=1,inplace=True)
df['encoder_inputs']=df['question'].apply(clean_text)
df['decoder_targets']=df['answer'].apply(clean_text)+' <end>'
df['decoder_inputs']='<start> '+df['answer'].apply(clean_text)+' <end>'

df.head(10)
```

output:

	question	answer	encoder_inputs	decoder_target	decoder_inputs
0	hi, how are you doing?	i'm fine. how about yourself?	hi , how are you doing ?	i ' m fine . how about yourself ? <end>	<start> i ' m fine . how about yourself ? <end>
1	i'm fine. how about yourself?	i'm pretty good. thanks for asking	i ' m fine . how about yourself ?	i ' m pretty good . thanks for asking . <end>	<start> i ' m pretty good . thanks for asking...
2	i'm pretty good. thanks for asking.	no problem. so how have you been?	i ' m pretty good . thanks for asking .	no problem . so how have you been ? <end>	<start> no problem . so how have you been ? ...
3	no problem. so how have you been?	i've been great. what about you?	no problem . so how have you been ?	i ' ve been great . what about you ? <end>	<start> i ' ve been great . what about you ? ...
4	i've been great. what about you?	i've been good. i'm in school right now.	i ' ve been great . what about you ?	i ' ve been good . i ' m in school right now ...	<start> i ' ve been good . i ' m in school ri...
5	i've been good. i'm in school right now.	what school do you go to?	i ' ve been good . i ' m in school right now .	what school do you go to ? <end>	<start> what school do you go to ? <end>

6	what school do you go to?	i go to pcc.	what school do you go to ?	i go to pcc . <end>	<start> i go to pcc . <end>
7	i go to pcc.	do you like it there?	i go to pcc .	do you like it there ? <end>	<start> do you like it there ? <end>
8	do you like it there?	it's okay. it's a really big campus.	do you like it there ?	it ' s okay . it ' s a really big campus . <...	<start> it ' s okay . it ' s a really big cam...
9	it's okay. it's a really big campus.	good luck with school.	it ' s okay . it ' s a really big campus .	good luck with school .<end>	<start>good luck with school.<end>

BUILD DECODER:

The seq2seq model also called the encoder-decoder model uses Long Short Term Memory- LSTM for text generation from the training corpus. The seq2seq model is also useful in machine translation applications. What does the seq2seq or encoder-decoder model do in simple words? It predicts a word given in the user input and then each of the next words is predicted using the probability of likelihood of that word to occur. In building our Generative chatbot we will use this approach for *text generation* given in the user input.

```
class Decoder(tf.keras.models.Model):
  def _init_(self,units,embedding_dim,vocab_size,args,*kwargs) ->
None:
    super()._init_(args,*kwargs)
    self.units=units
    self.embedding_dim=embedding_dim
```

```
        self.vocab_size=vocab_size
        self.embedding=Embedding(
            vocab_size,
            embedding_dim,
            name='decoder_embedding',
            mask_zero=True,
            embeddings_initializer=tf.keras.initializers.HeNormal()
        )
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='decoder_lstm',
            kernel_initializer=tf.keras.initializers.HeNormal()
)
        self.normalize=LayerNormalization()
        self.lstm=LSTM(
            units,
            dropout=.4,
            return_state=True,
            return_sequences=True,
            name='decoder_lstm',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )
        self.fc=Dense(
            vocab_size,
            activation='softmax',
```

```
            name='decoder_dense',
            kernel_initializer=tf.keras.initializers.HeNormal()
        )

    def call(self,decoder_inputs,encoder_states):
        x=self.embedding(decoder_inputs)
        x=self.normalize(x)
        x=Dropout(.4)(x)

x,decoder_state_h,decoder_state_c=self.lstm(x,initial_state=encoder_sta
tes)
        x=self.normalize(x)
        x=Dropout(.4)(x)
        return self.fc(x)

decoder=Decoder(lstm_cells,embedding_dim,vocab_size,name='decode
r')
decoder([1][:1],encoder([0][:1]))
```

OUTPUT:

```
<tf.Tensor: shape=(1, 30, 2443), dtype=float32, numpy=
array([[[2.7713785e-04, 9.3130053e-05, 3.3838153e-04, ...,
         6.9886592e-04, 2.0830621e-04, 1.2430754e-04],
        [9.0222697e-05, 2.3659073e-04, 2.2081460e-04, ...,
         8.5589236e-05, 4.0829674e-04, 5.0412118e-04],
        [1.9082776e-05, 3.8902971e-04, 4.1477769e-04, ...,
         3.8503637e-04, 3.9356644e-04, 3.3656124e-04],
        ...,
        [4.0933277e-04, 4.0933277e-04, 4.0933277e-04, ...,
         4.0933277e-04, 4.0933277e-04, 4.0933277e-04],
```

[4.0933277e-04, 4.0933277e-04, 4.0933277e-04, ...,
4.0933277e-04, 4.0933277e-04, 4.0933277e-04],
[4.0933277e-04, 4.0933277e-04, 4.0933277e-04, ...,
4.0933277e-04, 4.0933277e-04, 4.0933277e-04]]],
dtype=float32)>

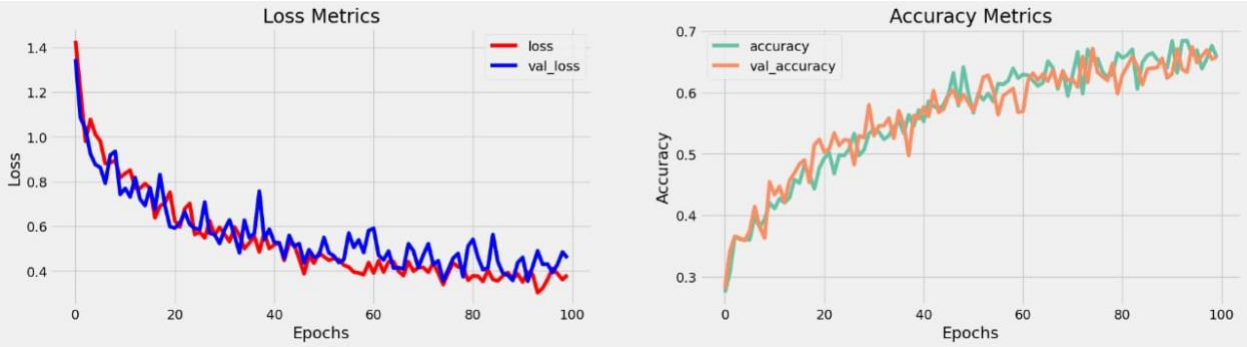
VISUALIZE METRICS:

The 9 most important chatbot metrics to track

1. Average conversation length
2. Total number of conversations
3. Total number of engaged conversations
4. Total number of unique users
5. Missed messages
6. Human takeover rate
7. Goal completion rate
8. Customer satisfaction scores
9. Average response time

```
fig,ax=plt.subplots(nrows=1,ncols=2,figsize=(20,5))
ax[0].plot(history.history['loss'],label='loss',c='red')
ax[0].plot(history.history['val_loss'],label='val_loss',c = 'blue')
ax[0].set_xlabel('Epochs')
ax[1].set_xlabel('Epochs')
ax[0].set_ylabel('Loss')
ax[1].set_ylabel('Accuracy')
ax[0].set_title('Loss Metrics')
ax[1].set_title('Accuracy Metrics')
ax[1].plot(history.history['accuracy'],label='accuracy')
ax[1].plot(history.history['val_accuracy'],label='val_accuracy')
ax[0].legend()
```

ax[1].legend()
plt.show()



CREATE INFERENCE MODEL:

```
class ChatBot(tf.keras.models.Model):
    def _init_(self,base_encoder,base_decoder,args,*kwargs):
        super()._init_(args,*kwargs)

self.encoder,self.decoder=self.build_inference_model(base_encoder,bas
e_decoder)

    def build_inference_model(self,base_encoder,base_decoder):
        encoder_inputs=tf.keras.Input(shape=(None,))
        x=base_encoder.layers[0](encoder_inputs)
        x=base_encoder.layers[1](x)
        x,encoder_state_h,encoder_state_c=base_encoder.layers[2](x)

encoder=tf.keras.models.Model(inputs=encoder_inputs,outputs=[encode
r_state_h,encoder_state_c],name='chatbot_encoder')

        decoder_input_state_h=tf.keras.Input(shape=(lstm_cells,))
        decoder_input_state_c=tf.keras.Input(shape=(lstm_cells,))
        decoder_inputs=tf.keras.Input(shape=(None,))
```

```
        x=base_decoder.layers[0](decoder_inputs)
        x=base_encoder.layers[1](x)

x,decoder_state_h,decoder_state_c=base_decoder.layers[2](x,initial_stat
e=[decoder_input_state_h,decoder_input_state_c])
        decoder_outputs=base_decoder.layers[-1](x)
        decoder=tf.keras.models.Model(

inputs=[decoder_inputs,[decoder_input_state_h,decoder_input_state_c]],

outputs=[decoder_outputs,[decoder_state_h,decoder_state_c]],name='ch
atbot_decoder'
        )
        return encoder,decoder

    def summary(self):
        self.encoder.summary()
        self.decoder.summary()

    def softmax(self,z):
        return np.exp(z)/sum(np.exp(z))

    def sample(self,conditional_probability,temperature=0.5):
        conditional_probability =
np.asarray(conditional_probability).astype("float64")
        conditional_probability = np.log(conditional_probability) /
temperature
        reweighted_conditional_probability =
self.softmax(conditional_probability)
```

```
        probas = np.random.multinomial(1,
reweighted_conditional_probability, 1)
        return np.argmax(probas)

    def preprocess(self,text):
        text=clean_text(text)
        seq=np.zeros((1,max_sequence_length),dtype=np.int32)
        for i,word in enumerate(text.split()):
            seq[:,i]=sequences2ids(word).numpy()[0]
        return seq

    def postprocess(self,text):
        text=re.sub(' - ','-',text.lower())
        text=re.sub(' [.] ','. ',text)
        text=re.sub(' [1] ','1',text)
        text=re.sub(' [2] ','2',text)
        text=re.sub(' [3] ','3',text)
        text=re.sub(' [4] ','4',text)
        text=re.sub(' [5] ','5',text)
        text=re.sub(' [6] ','6',text)
        text=re.sub(' [7] ','7',text)
        text=re.sub(' [8] ','8',text)
        text=re.sub(' [9] ','9',text)
        text=re.sub(' [0] ','0',text)
        text=re.sub(' [,] ',', ',text)
        text=re.sub(' [?] ','? ',text)
        text=re.sub(' [!] ','! ',text)
        text=re.sub(' [$] ','$ ',text)
        text=re.sub(' [&] ','& ',text)
```

```
        text=re.sub(' [/] ','/ ',text)
        text=re.sub(' [:] ',': ',text)
        text=re.sub(' [;] ','; ',text)
        text=re.sub(' [] ',' ',text)
        text=re.sub(' [\'] ','\'',text)
        text=re.sub(' [\"] ','\"',text)
        return text

    def call(self,text,config=None):
        input_seq=self.preprocess(text)
        states=self.encoder(input_seq,training=False)
        target_seq=np.zeros((1,1))
        target_seq[:,:]=sequences2ids(['<start>']).numpy()[0][0]
        stop_condition=False
        decoded=[]
        while not stop_condition:

decoder_outputs,new_states=self.decoder([target_seq,states],training=Fa
lse)
#             index=tf.argmax(decoder_outputs[:,-1,:],axis=-
1).numpy().item()
            index=self.sample(decoder_outputs[0,0,:]).item()
            word=ids2sequences([index])
            if word=='<end> ' or len(decoded)>=max_sequence_length:
                stop_condition=True
            else:
                decoded.append(index)
target_seq=np.zeros((1,1))
                target_seq[:,:]=index
```

```
            states=new_states
        return self.postprocess(ids2sequences(decoded))

chatbot=ChatBot(model.encoder,model.decoder,name='chatbot')
chatbot.summary()
```

OUTPUT:

```
Model: "chatbot_encoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, None)]            0

 encoder_embedding (Embeddin  (None, None, 256)        625408
 g)

 layer_normalization (LayerN  (None, None, 256)        512
 ormalization)

 encoder_lstm (LSTM)         [(None, None, 256),       525312
                              (None, 256),
                              (None, 256)]

=================================================================
Total params: 1,151,232
Trainable params: 1,151,232
Non-trainable params: 0
_________________________________________________________________
```

Model: "chatbot_decoder"

_____

Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	[(None, None)]	0	[]
decoder_embedding (Embedding)	(None, None, 256)	625408	['input_4[0][0]']
layer_normalization (LayerNorm alization)	(None, None, 256)	512	['decoder_embedding[0][0]']
input_2 (InputLayer)	[(None, 256)]	0	[]
input_3 (InputLayer)	[(None, 256)]	0	[]
decoder_lstm (LSTM)	[(None, None, 256), (None, 256), (None, 256)]	525312	['layer_normalization[1][0]', 'input_2[0][0]', 'input_3[0][0]']
decoder_dense (Dense)	(None, None, 2443)	627851	['decoder_lstm[0][0]']

Total params: 1,779,083

Trainable params: 1,779,083

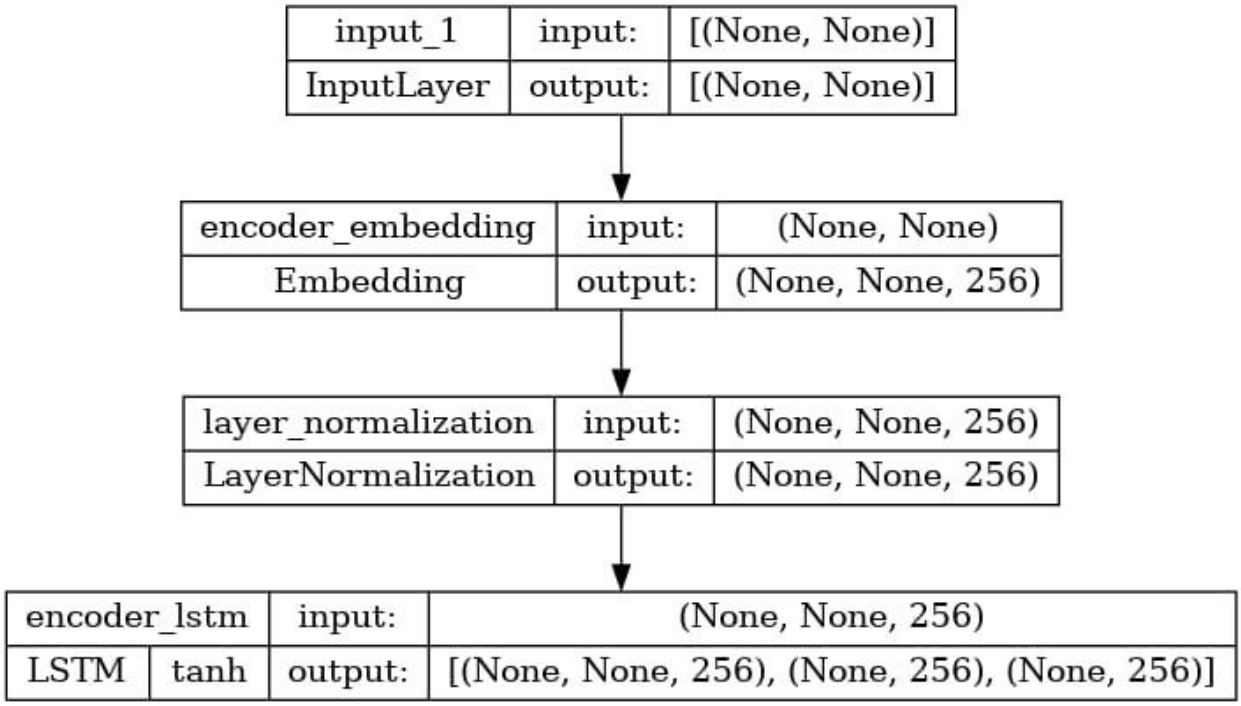
Non-trainable params: 0

_____

i).

tf.keras.utils.plot_model(chatbot.encoder,to_file='encoder.png',show_shapes=True,show_layer_activations=True)

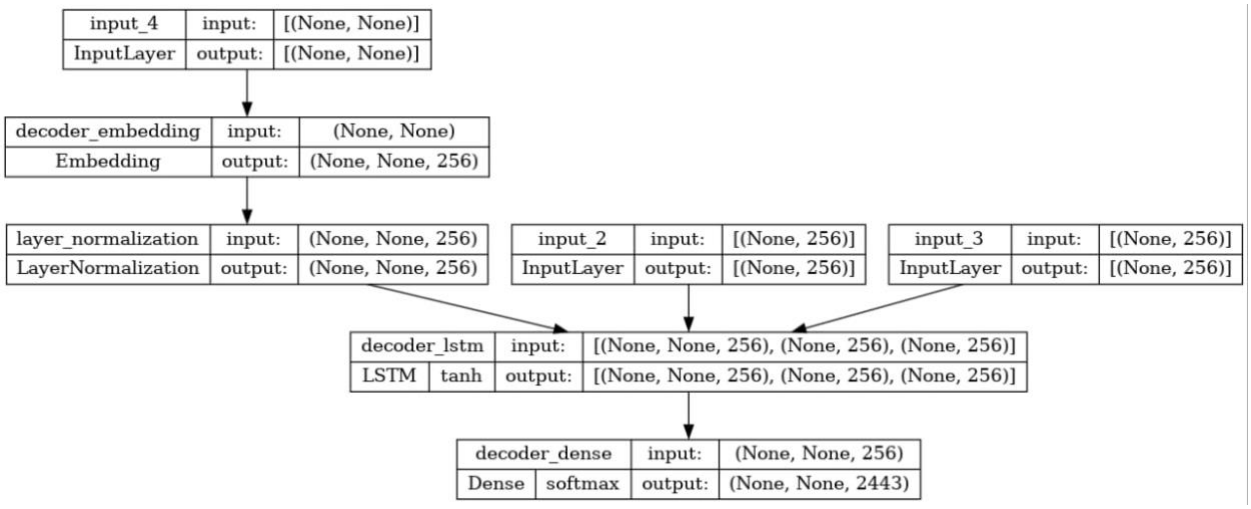
OUTPUT:



ii).

tf.keras.utils.plot_model(chatbot.decoder,to_file='decoder.png',show_shapes=True,show_layer_activations=True)

OUTPUT:



TIME TO CHAT:

We have a time to chat our communicate with chatbot and also we have to check how the chatbot response our message.so, due to this response time they can rate our product of chatbot.

PROGRAM:

```
def print_conversation(texts):
    for text in texts:
        print(f'You: {text}')
        print(f'Bot: {chatbot(text)}')
        print('==========================')
print_conversation([
    'hi',
    'do yo know me?',
    'what is your name?',
    'you are bot?',
    'hi, how are you doing?',
    "i'm pretty good. thanks for asking.",
    "Don't ever be in a hurry",
    "'I'm gonna put some dirt in your eye '",
    "'You're trash '",
```

"'I've read all your research on nano-technology '",
"'You want forgiveness? Get religion'",
"'While you're using the bathroom, i'll order some food.'",
"'Wow! that's terrible.'",
"'We'll be here forever.'",
"'I need something that's reliable.'",
"'A speeding car ran a red light, killing the girl.'",
"'Tomorrow we'll have rice and fish for lunch.'",
"'I like this restaurant because they give you free bread.'"
])
You: hi
Bot: have you been?
==========
You: do yo know me?
Bot: because if there are all the seats, the tv is gray.
==========
You: what is your name?
Bot: it's not my mud.
==========
You: you are bot?
Bot: i have a lot of a good spot.
==========
You: hi, how are you doing?
Bot: i'm going to see the movie this weekend.
==========
You: i'm pretty good. thanks for asking.
Bot: no problem. i'm sorry.
==========
You: Don't ever be in a hurry

Bot: it's not a good job.
==========
You: I'm gonna put some dirt in your eye
Bot: the one good.
==========
You: You're trash
Bot: it has lots of rooms and lots of space.
==========
You: I've read all your research on nano-technology
Bot: i'll say that you've been to see a movie this weekend.
==========
You: You want forgiveness? Get religion
Bot: no, i will. i'm still feeling a good.
==========
You: While you're using the bathroom, i'll order some food.
Bot: don't order for me, i want to go to see it.
==========
You: Wow! that's terrible.
Bot: never park your car.
==========
You: We'll be here forever.
Bot: i'll get a bait.
==========
You: I need something that's reliable.
Bot: you have a lot of plants.
==========
You: A speeding car ran a red light, killing the girl.
Bot: what happened to the other student?
==========

You: Tomorrow we'll have rice and fish for lunch.
Bot: i'll see you there.
==========
You: I like this restaurant because they give you free bread.
Bot: well, i just put a stamp on the envelope.
==========