bot\_template = "BOT : {0}"

user\_template = "USER : {0}"

# Define a function that responds to a user's message: respond

def respond(message):

# Concatenate the user's message to the end of a standard bot respone

bot\_message = "I can hear you! You said: " + message

# Return the result

return bot\_message

# Define a function that sends a message to the bot: send\_message

def send\_message(message):

# Print user\_template including the user\_message

print(user\_template.format(message))

# Get the bot's response to the message

response = respond(message)

# Print the bot template including the bot's response.

print(bot\_template.format(response))

# Send a message to the bot

send\_message("hello")

# Define variables

name = "Greg"

weather = "cloudy"

# Define a dictionary with the predefined responses

responses = {

"what's your name?": "my name is {0}".format(name),

"what's today's weather?": "the weather is {0}".format(weather),

"default": "default message"

}

# Return the matching response if there is one, default otherwise

def respond(message):

# Check if the message is in the responses

if message in responses:

# Return the matching message

bot\_message = responses[message]

else:

# Return the "default" message

bot\_message = responses["default"]

return bot\_message

# Import the random module

import random

name = "Greg"

weather = "cloudy"

# Define a dictionary containing a list of responses for each message

responses = {

"what's your name?": [

"my name is {0}".format(name),

"they call me {0}".format(name),

"I go by {0}".format(name)

],

"what's today's weather?": [

"the weather is {0}".format(weather),

"it's {0} today".format(weather)

],

"default": ["default message"]

}

# Use random.choice() to choose a matching response

def respond(message):

# Check if the message is in the responses

if message in responses:

# Return a random matching response

bot\_message = random.choice(responses[message])

else:

# Return a random "default" response

bot\_message = random.choice(responses["default"])

return bot\_message

import random

def respond(message):

# Check for a question mark

if message.endswith('?'):

# Return a random question

return random.choice(responses["question"])

# Return a random statement

return random.choice(responses["statement"])

# Send messages ending in a question mark

send\_message("what's today's weather?")

send\_message("what's today's weather?")

# Send messages which don't end with a question mark

send\_message("I love building chatbots")

send\_message("I love building chatbots")

# Define match\_rule()

def match\_rule(rules, message):

response, phrase = "default", None

# Iterate over the rules dictionary

for pattern, responses in rules.items():

# Create a match object

match = re.search(pattern, message)

if match is not None:

# Choose a random response

response = random.choice(responses)

if '{0}' in response:

phrase = match.group(1)

# Return the response and phrase

# Return the response and phrase

return response, phrase

# Test match\_rule

print(match\_rule(rules, "do you remember your last birthday"))

# Define replace\_pronouns()

def replace\_pronouns(message):

message = message.lower()

if 'me' in message:

# Replace 'me' with 'you'

return re.sub('me', 'you', message)

if 'my' in message:

# Replace 'my' with 'your'

return re.sub('my', 'your', message)

if 'your' in message:

# Replace 'your' with 'my'

return re.sub('your', 'my', message)

if 'you' in message:

# Replace 'you' with 'me'

return re.sub('you', 'me', message)

return message

print(replace\_pronouns("my last birthday"))

print(replace\_pronouns("when you went to Florida"))

print(replace\_pronouns("I had my own castle"))

# Define respond()

def respond(message):

# Call match\_rule

response, phrase = match\_rule(rules,message)

if '{0}' in response:

# Replace the pronouns in the phrase

phrase = replace\_pronouns(phrase)

# Include the phrase in the response

response = response.format(phrase)

return response

# Send the messages

send\_message("do you remember your last birthday")

send\_message("do you think humans should be worried about AI")

send\_message("I want a robot friend")

send\_message("what if you could be anything you wanted")

# Define a dictionary of patterns

patterns = {}

# Iterate over the keywords dictionary

for intent, keys in keywords.items():

# Create regular expressions and compile them into pattern objects

patterns[intent] = re.compile('|'.join(keys))

# Print the patterns

print(patterns)

# Define a function to find the intent of a message

def match\_intent(message):

matched\_intent = None

for intent, pattern in patterns.items():

# Check if the pattern occurs in the message

if pattern.search(message):

matched\_intent = intent

return matched\_intent

# Define a respond function

def respond(message):

# Call the match\_intent function

intent = match\_intent(message)

# Fall back to the default response

key = "default"

if intent in responses:

key = intent

return responses[key]

# Send messages

send\_message("hello!")

send\_message("bye byeee")

send\_message("thanks very much!")

# Define find\_name()

def find\_name(message):

name = None

# Create a pattern for checking if the keywords occur

name\_keyword = re.compile('name|call')

# Create a pattern for finding capitalized words

name\_pattern = re.compile('[A-Z]{1}[a-z]\*')

if name\_keyword.search(message):

# Get the matching words in the string

name\_words = name\_pattern.findall(message)

if len(name\_words) > 0:

# Return the name if the keywords are present

name = ' '.join(name\_words)

return name

# Define respond()

def respond(message):

# Find the name

name = find\_name(message)

if name is None:

return "Hi there!"

else:

return "Hello, {0}!".format(name)

# Send messages

send\_message("my name is David Copperfield")

send\_message("call me Ishmael")

send\_message("People call me Cassandra")

# Load the spacy model: nlp

nlp = spacy.load('en')

# Calculate the length of sentences

n\_sentences = len(sentences)

# Calculate the dimensionality of nlp

embedding\_dim = nlp.vocab.vectors\_length

# Initialize the array with zeros: X

X = np.zeros((n\_sentences, embedding\_dim))

# Iterate over the sentences

for idx, sentence in enumerate(sentences):

# Pass each each sentence to the nlp object to create a document

doc = nlp(sentence)

# Save the document's .vector attribute to the corresponding row in X

X[idx, :] = doc.vector

# Import SVC

from sklearn.svm import SVC

# Create a support vector classifier

clf = SVC(C=1)

# Fit the classifier using the training data

clf.fit(X\_train, y\_train)

# Predict the labels of the test set

y\_pred = clf.predict(X\_test)

#knn.fit(X\_train, y\_train)

# Count the number of correct predictions

n\_correct = 0

for i in range(len(y\_test)):

if y\_pred[i] == y\_test[i]:

n\_correct += 1

print("Predicted {0} correctly out of {1} test examples".format(n\_correct, len(y\_test)))

# Define included entities

include\_entities = ['DATE', 'ORG', 'PERSON']

# Define extract\_entities()

def extract\_entities(message):

# Create a dict to hold the entities

ents = dict.fromkeys(include\_entities)

# Create a spacy document

doc = nlp(message)

for ent in doc.ents:

if ent.label\_ in include\_entities:

# Save interesting entities

ents[ent.label\_] = ent.text

return ents

print(extract\_entities('friends called Mary who have worked at Google since 2010'))

print(extract\_entities('people who graduated from MIT in 1999'))

# Create the document

doc = nlp("let's see that jacket in red and some blue jeans")

# Iterate over parents in parse tree until an item entity is found

def find\_parent\_item(word):

# Iterate over the word's ancestors

for parent in word.ancestors:

# Check for an "item" entity

if entity\_type(parent) == "item":

return parent.text

return None

# For all color entities, find their parent item

def assign\_colors(doc):

# Iterate over the document

for word in doc:

# Check for "color" entities

if entity\_type(word) == "color":

# Find the parent

item = find\_parent\_item(word)

print("item: {0} has color : {1}".format(item, word))

# Assign the colors

assign\_colors(doc)

# Import necessary modules

from rasa\_nlu.converters import load\_data

from rasa\_nlu.config import RasaNLUConfig

from rasa\_nlu.model import Trainer

# Create args dictionary

args = {"pipeline":"spacy\_sklearn"}

#cmdline\_args={"pipeline": "spacy\_sklearn"})

# Create a configuration and trainer

config = RasaNLUConfig(cmdline\_args=args)

trainer = Trainer(config)

# Load the training data

training\_data = load\_data("./training\_data.json")

# Create an interpreter by training the model

interpreter = trainer.train(training\_data)

# Try it out

print(interpreter.parse("I'm looking for a Mexican restaurant in the North of town"))

# Import necessary modules

from rasa\_nlu.config import RasaNLUConfig

from rasa\_nlu.model import Trainer

pipeline = [

"nlp\_spacy",

"tokenizer\_spacy",

"ner\_crf"

]

# Create a config that uses this pipeline

config = RasaNLUConfig(cmdline\_args={'pipeline': pipeline})

# Create a trainer that uses this config

trainer = Trainer(config)

# Create an interpreter by training the model

interpreter = trainer.train(training\_data)

# Parse some messages

print(interpreter.parse("show me Chinese food in the centre of town"))

print(interpreter.parse("I want an Indian restaurant in the west"))

print(interpreter.parse("are there any good pizza places in the center?"))

# Import sqlite3

import sqlite3

# Open connection to DB

conn = sqlite3.connect('hotels.db')

# Create a cursor

c = conn.cursor()

# Define area and price

area, price = "south", "hi"

t = (area, price)

# Execute the query

c.execute('SELECT \* FROM hotels WHERE area=? AND price=?', t)

# Print the results

print(c.fetchall())

# Define find\_hotels()

def find\_hotels(params):

# Create the base query

query = 'SELECT \* FROM hotels'

# Add filter clauses for each of the parameters

if len(params) > 0:

filters = ["{}=?".format(k) for k in params]

query += " WHERE " + " and ".join(filters)

# Create the tuple of values

t = tuple(params.values())

# Open connection to DB

conn = sqlite3.connect('hotels.db')

# Create a cursor

c = conn.cursor()

# Execute the query

c.execute(query, t)

# Return the results

return c.fetchall()

# Create the dictionary of column names and values

params = {'area': 'south', 'price': 'lo'}

# Find the hotels that match the parameters

print(find\_hotels(params))

# Define respond()

def respond(message):

# Extract the entities

entities = interpreter.parse(message)["entities"]

# Initialize an empty params dictionary

params = {}

# Fill the dictionary with entities

for ent in entities:

params[ent["entity"]] = str(ent["value"])

# Find hotels that match the dictionary

results = find\_hotels(params)

# Get the names of the hotels and index of the response

names = [r[0] for r in results]

n = min(len(results),3)

# Select the nth element of the responses array

return responses[n].format(\*names)

print(respond("I want an expensive hotel in the south of town"))