In [ ]:
```
#### Stateful bots
```

In [ ]:
```
## Form filling
```

In [1]:
```
def send_message(policy, state, message):
    print("USER : {}".format(message))
    new_state, response = respond(policy, state, message)
    print("BOT : {}".format(response))
    return new_state

def respond(policy, state, message):
    (new_state, response) = policy[(state, interpret(message))]
    return new_state, response

def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    return 'none'
```

In [2]:
```
# Define the INIT state
INIT = 0
# Define the CHOOSE_COFFEE state
CHOOSE_COFFEE = 1
# Define the ORDERED state
ORDERED = 2

# Define the policy rules
policy = {
    (INIT, "order"): (CHOOSE_COFFEE, "ok, Colombian or Kenyan?"),
    (INIT, "none"): (INIT, "I'm sorry - I'm not sure how to help you"),
    (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!"),
    (CHOOSE_COFFEE, "none"): (CHOOSE_COFFEE, "I'm sorry - would you like Colombian or Kenyan?"),
}

# Create the list of messages
messages = [
    "I'd like to become a professional dancer",
    "well then I'd like to order some coffee",
    "my favourite animal is a zebra",
    "kenyan"
]

# Call send_message() for each message
state = INIT
for message in messages:
    state = send_message(policy, state, message)
```

```
USER : I'd like to become a professional dancer
BOT : I'm sorry - I'm not sure how to help you
USER : well then I'd like to order some coffee
BOT : ok, Colombian or Kenyan?
USER : my favourite animal is a zebra
BOT : I'm sorry - would you like Colombian or Kenyan?
USER : kenyan
BOT : perfect, the beans are on their way!
```

In [ ]:
```
## Asking contextual questions
```

In [3]:
```
def send_message(state, message):
    print("USER : {}".format(message))
    new_state, response = respond(state, message)
    print("BOT : {}".format(response))
    return new_state

def respond(state, message):
    (new_state, response) = policy_rules[(state, interpret(message))]
    return new_state, response


def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    if 'what' in msg:
        return 'ask_explanation'
    return 'none'
```

In [4]:
```python
# Define the states
INIT=0
CHOOSE_COFFEE=1
ORDERED=2

# Define the policy rules dictionary
policy_rules = {
    (INIT, "ask_explanation"): (INIT, "I'm a bot to help you order coffee beans"),
    (INIT, "order"): (CHOOSE_COFFEE, "ok, Columbian or Kenyan?"),
    (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!"),
    (CHOOSE_COFFEE, "ask_explanation"): (CHOOSE_COFFEE, "We have two kinds of coffee beans - the Kenyan ones make a slightly
}

# Define send_messages()
def send_messages(messages):
    state = INIT
    for msg in messages:
        state = send_message(state, msg)

# Send the messages
send_messages([
    "what can you do for me?",
    "well then I'd like to order some coffee",
    "what do you mean by that?",
    "kenyan"
])
```

```
USER : what can you do for me?
BOT : I'm a bot to help you order coffee beans
USER : well then I'd like to order some coffee
BOT : ok, Columbian or Kenyan?
USER : what do you mean by that?
BOT : We have two kinds of coffee beans - the Kenyan ones make a slightly sweeter coffee, and cost $6. The Brazilian beans
make a nutty coffee and cost $5.
USER : kenyan
BOT : perfect, the beans are on their way!
```

In [ ]:
```python
## Dealing with rejection
```

In [33]:
```
import sqlite3

responses = ["I'm sorry :( I couldn't find anything like that", '{} is a great hotel!', '{} or {} would work!', '{} is one of

def interpret(message):
    data = interpreter.parse(message)
    if 'no' in message:
        data["intent"]["name"] = "deny"
    return data

def find_hotels(params, excluded):
    query = 'SELECT * FROM hotels'
    if len(params) > 0:
        filters = ["{}=?".format(k) for k in params] + ["name!='?'".format(k) for k in excluded]
        query += " WHERE " + " and ".join(filters)
    t = tuple(params.values())

    # open connection to DB
    conn = sqlite3.connect('hotels.db')
    # create a cursor
    c = conn.cursor()
    c.execute(query, t)
    return c.fetchall()

# Import necessary modules
from rasa_nlu.training_data import load_data
from rasa_nlu.config import RasaNLUModelConfig
from rasa_nlu.model import Trainer
from rasa_nlu import config

# Create a trainer that uses this config
trainer = Trainer(config.load("config_spacy.yml"))

# Load the training data
training_data = load_data('demo-rasa-noents.json')

# Create an interpreter by training the model
interpreter = trainer.train(training_data)
```

```
C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\rasa\nlu\config.py:47: FutureWarning: You have specified the pipe
line template 'spacy_sklearn' which has been renamed to 'pretrained_embeddings_spacy'. Please update your configuration as
it will no longer work with future versions of Rasa.
  return RasaNLUModelConfig(config)

Fitting 2 folds for each of 6 candidates, totalling 12 fits

C:\Users\84353\Anaconda3\envs\chat_box\lib\site-packages\rasa\nlu\training_data\formats\readerwriter.py:37: FutureWarning:
Your rasa data contains 'intent_examples' or 'entity_examples' which will be removed in the future. Consider putting all y
our examples into the 'common_examples' section.
  return self.read_from_json(js, **kwargs)
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:    0.0s finished
```

In [37]:
```
# Define respond()
def respond(message, params, suggestions, excluded):
    # Interpret the message
    parse_data = interpret(message)
    # Extract the intent
    intent =  parse_data["intent"]["name"]
    #print(intent)
    # Extract the entities
    entities = parse_data["entities"]
    # Add the suggestion to the excluded list if intent is "deny"
    if intent == "deny":
        excluded.extend(suggestions)
    # Fill the dictionary with entities
    for ent in entities:
        params[ent["entity"]] = str(ent["value"])
    # Find matching hotels
    results = [
        r
        for r in find_hotels(params, excluded)
        if r[0] not in excluded
    ]
    # Extract the suggestions
    names = [r[0] for r in results]
    n = min(len(results), 3)
    suggestions = names[:2]
    return responses[n].format(*names), params, suggestions, excluded

# Initialize the empty dictionary and lists
params, suggestions, excluded = {}, [], []

# Send the messages
for message in ["I want a mid range hotel", "no that doesn't work for me"]:
    print("USER: {}".format(message))
    response, params, suggestions, excluded = respond(message, params, suggestions, excluded)
    print("BOT: {}".format(response))
```

```
USER: I want a mid range hotel
BOT: Hotel for Dogs is one option, but I know others too :)
USER: no that doesn't work for me
BOT: Grand Hotel is one option, but I know others too :)
```

In [ ]:
```
#### Asking questions & queuing answers
```

In [ ]:
```
## Pending actions I
```

In [48]:
```
# Define policy()
def policy(intent):
    # Return "do_pending" if the intent is "affirm"
    if intent == "affirm":
        return "do_pending", None
    # Return "Ok" if the intent is "deny"
    if intent == "deny":
        return "Ok", None
    if intent == "order":
        return "Unfortunately, the Kenyan coffee is currently out of stock, would you like to order the Brazilian beans?", "
```

In [ ]:
```
## Pending actions II
```

In [49]:
```
def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    elif 'yes' in msg:
        return 'affirm'
    elif 'no' in msg:
        return 'deny'
    return 'none'
```

```
In [50]: # Define send_message()
def send_message(message,pending):
    print("USER : {}".format(message))
    response,pending_action = policy(interpret(message))
    if response == "do_pending" and pending is not None:
        print("BOT : {}".format(pending))
    else:
        print("BOT : {}".format(response))
    return pending_action

# Define send_messages()
def send_messages(messages):
    pending_action = None
    for msg in messages:
        pending_action = send_message(msg,pending_action)

# Send the messages
send_messages([
    "I'd like to order some coffee",
    "ok yes please"
])
```

```
USER : I'd like to order some coffee
BOT : Unfortunately, the Kenyan coffee is currently out of stock, would you like to order the Brazilian beans?
USER : ok yes please
BOT : Alright, I've ordered that for you!
```

```
In [ ]: ## Pending state transitions
```

```
In [51]: import string

def send_message(state, pending, message):
    print("USER : {}".format(message))
    new_state, response, pending_state = policy_rules[(state, interpret(message))]
    print("BOT : {}".format(response))
    if pending is not None:
        new_state, response, pending_state = policy_rules[pending]
        print("BOT : {}".format(response))
    if pending_state is not None:
        pending = (pending_state, interpret(message))
    return new_state, pending

def interpret(message):
    msg = message.lower()
    if 'order' in msg:
        return 'order'
    if 'kenyan' in msg or 'columbian' in msg:
        return 'specify_coffee'
    if any([d in msg for d in string.digits]):
        return 'number'
    return 'none'
```

In [52]:
```
# Define the states
INIT=0
AUTHED=1
CHOOSE_COFFEE=2
ORDERED=3

# Define the policy rules
policy_rules = {
    (INIT, "order"): (INIT, "you'll have to log in first, what's your phone number?", AUTHED),
    (INIT, "number"): (AUTHED, "perfect, welcome back!", None),
    (AUTHED, "order"): (CHOOSE_COFFEE, "would you like Columbian or Kenyan?", None),
    (CHOOSE_COFFEE, "specify_coffee"): (ORDERED, "perfect, the beans are on their way!", None)
}

# Define send_messages()
def send_messages(messages):
    state = INIT
    pending = None
    for msg in messages:
        state, pending = send_message(state, pending, msg)

# Send the messages
send_messages([
    "I'd like to order some coffee",
    "555-12345",
    "kenyan"
])
```

```
USER : I'd like to order some coffee
BOT : you'll have to log in first, what's your phone number?
USER : 555-12345
BOT : perfect, welcome back!
BOT : would you like Columbian or Kenyan?
USER : kenyan
BOT : perfect, the beans are on their way!
BOT : would you like Columbian or Kenyan?
```

In [ ]:
```
## Putting it all together I
```

In [53]:
```
# Define chitchat_response()
def chitchat_response(message):
    # Call match_rule()
    response, var = match_rule(rules,message)
    # Return none is response is "default"
    if response == "default":
        return None
    if '{0}' in response:
        # Replace the pronouns of phrase
        phrase = replace_pronouns(var)
        # Calculate the response
        response = response.format(phrase)
    return response
```

In [ ]:
```
## Putting it all together II
```

In [54]:
```
import re
import random

def match_rule(rules, message):
    for pattern, responses in rules.items():
        match = re.search(pattern, message)
        if match is not None:
            response = random.choice(responses)
            var = match.group(1) if '{0}' in response else None
            return response, var
    return "default", None

rules = {'if (.*)': ["Do you really think it's likely that {0}", 'Do you wish that {0}', 'What do you think about {0}', 'Rea

def replace_pronouns(message):

    message = message.lower()
    if 'me' in message:
        return re.sub('me', 'you', message)
    if 'i' in message:
        return re.sub('i', 'you', message)
    elif 'my' in message:
        return re.sub('my', 'your', message)
    elif 'your' in message:
        return re.sub('your', 'my', message)
    elif 'you' in message:
        return re.sub('you', 'me', message)

    return message
```

In [55]:
```python
# Define send_message()
def send_message(state,pending,message):
    print("USER : {}".format(message))
    response = chitchat_response(message)
    if response is not None:
        print("BOT : {}".format(response))
        return state, None

    # Calculate the new_state, response, and pending_state
    new_state, response, pending_state = policy_rules[(state, interpret(message))]
    print("BOT : {}".format(response))
    if pending is not None:
        new_state, response, pending_state = policy_rules[pending]
        print("BOT : {}".format(response))
    if pending_state is not None:
        pending = (pending_state, interpret(message))
    return new_state, pending

# Define send_messages()
def send_messages(messages):
    state = INIT
    pending = None
    for msg in messages:
        state, pending = send_message(state, pending, msg)

# Send the messages
send_messages([
    "I'd like to order some coffee",
    "555-12345",
    "do you remember when I ordered 1000 kilos by accident?",
    "kenyan"
])
```

```
USER : I'd like to order some coffee
BOT : you'll have to log in first, what's your phone number?
USER : 555-12345
BOT : perfect, welcome back!
BOT : would you like Columbian or Kenyan?
USER : do you remember when I ordered 1000 kilos by accident?
BOT : Yes .. and?
USER : kenyan
BOT : perfect, the beans are on their way!
```

In [ ]:
```
#### Frontiers of dialogue research
```

In [39]:
```python
def sample_text(seed, temperature):
    return generated[temperature]

generated = {0.2: "i'm gonna punch lenny in the back of the been a to the on the man to the mother and the father to simpson
```

In [41]:
```python
# Feed the 'seed' text into the neural network
seed = "i'm gonna punch lenny in the back of the"

# Iterate over the different temperature values
for temperature in [0.2, 0.5, 1.0, 1.2]:
    print("\nGenerating text with riskiness : {}\n".format(temperature))
    # Call the sample_text function
    print(sample_text(seed,temperature))
```

```

Generating text with riskiness : 0.2

i'm gonna punch lenny in the back of the been a to the on the man to the mother and the father to simpson the father to wi
th the marge in the for the like the fame to the been to the for my bart the don't was in the like the for the father the
father a was the father been a say the been to me the do it and the father been to go. i want to the boy i can the from a
man to be the for the been a like the father to make my bart of the father

Generating text with riskiness : 0.5

i'm gonna punch lenny in the back of the kin't she change and i'm all better it and the was the fad a drivera it? what i w
ant to did hey, he would you would in your bus who know is the like and this don't are for your this all for your manset t
he for it a man is on the see the will they want to know i'm are for one start of that and i got the better this is. it wh
oce and i don't are on the mater stop in the from a for the be your mileat

Generating text with riskiness : 1.0

i'm gonna punch lenny in the back of the to to macks how screath. firl done we wouldn't wil that kill. of this torshmobote
since, i know i ord did, can give crika of sintenn prescoam.whover my me after may? there's right. that up. there's ruinin
g isay.oh.solls.nan'h those off point chuncing car your anal medion.hey, are exallies a off while bea dolk of sure, hello,
no in her, we'll rundems... i'm eventy taving me to too the letberngonce

Generating text with riskiness : 1.2

i'm gonna punch lenny in the back of the burear prespe-nakes, 'lisa to isn't that godios.and when be the bowniday' would l
ochs meine, mind crikvin' suhle ovotaci!.... hey, a poielyfd othe flancer, this in are rightplouten of of we doll hurrs,
truelturone? rake inswaydan justy!we scrikent.ow.. by back hous, smadge, the lighel irely.yes, homer. wel'e esasmoy ryelal
rs all wronencay...... nank. i wenth makedyk. come on help cerzind, now, n
```