

CMPE 561  
NATURAL LANGUAGE  
PROCESSING

YAŞAR ALİM TÜRKMEN  
INSTRUCTOR: TUNGA GÜNGÖR

DESIGNING CHAT BOT WITH REGULAR  
EXPRESSIONS  
PROGRAMMING PROJECT

19.11.2019

## INTRODUCTION

This project aims to design a chat bot for the undergrad computer engineering students in Boğaziçi University. The chat bot answers the students' questions about registration, taking/adding/dropping a course, the abbreviations and definitions etc. It can also ask for detail information about the question the student asked. Therefore, it doesn't only answer some particular questions but can answer questions with different forms.

A question can be asked in many different forms. Therefore, chat bot needs to recognize different patterns. In order to do that chat bot is designed using regular expressions. A regular expression is a sequence of characters that define a search pattern.

There are some patterns defined in the system and according to those patterns chat bot generates answers. The system doesn't cover all of the information about the procedures and rules of the department. A small knowledge base is supplied and chat bot only answers the questions defined in that knowledge base.

## PROGRAM INTERFACE

The program is written in Python and regular expression module `re` is used. Therefore you need 3 things to run this program:

- Python 3 (any version of it)
- A terminal program in order to run the code
- `Regex(re)` and `termcolor` module (Regular expression module)

After you install python3, `re` module and a terminal program you can simply run the code with:

```
python3 chat_bot.py
```

You don't need to set any parameters or change anything in the code. Also the program doesn't need any input file to run.

In order to terminate the program press `Ctrl+c`.

## PROGRAM EXECUTION

After you run the program you will see a welcoming message written in green. The messages coming from the chat bot are in green and yours are in white. The program runs an infinite loop, thus, unless you terminate it, it waits questions to answer. Sometimes chat bot also asks questions to get the detail information about you. You should also answer them to get the answer of your question. If the chat bot asks you a question, that means you started a sequence. In order to stop the sequence you should write "nq".

Also remember that the chat bot designed for answering single questions. You should avoid long and multiple questions. Try to ask your questions separately.

## INPUT OUTPUT

As stated before, this program doesn't need any input file but your messages/questions.

## PROGRAM STRUCTURE

Program runs an infinite loop and waits input messages from the users. For each message, the same procedures are applied and regarding to the message an answer or in other words a response is generated. First the message is simply preprocessed. Then, the message is searched in knowledge base with regular expressions. If the question is defined in the knowledge base, a response is generated otherwise user is asked to retype the question.

The default messages like welcoming and error messages or the response for unmatched questions are set at the beginning of the code. You can change them if you want. Then **char\_set**, **clitic\_set**, **meanings** and **patterns** dictionaries are defined.

**char\_set:** This dictionary maps the letters in uppercase to the lowercase forms. This dictionary is used while case folding process runs.

**clitic\_set:** In clitic set, clitics in English are defined except “s” since it causes ambiguity. It can be “is” or possessive ‘s. Each clitic is mapped to regarding word.

**meanings:** Some of the abbreviations or terms are defined here. Only a few definitions are put under this dictionary. You can expand it with other abbreviations and frequently asked terms. Here, the terms and abbreviations are keys and the meanings of them are values in the dictionary.

**patterns:** Main dictionary of the system. All of the possible patterns are introduced in this dictionary. Regular expressions are used as keys and the regarding answers are mapped to those expressions. If a person asks a question which have different answers under different conditions, it starts a sequence. After the sequence is started, the other items in that sequence are recognized with prefixes and suffixes. The prefix and suffix are generated by the index of the sequence starter question. Here's an example from the code:

```
r'.*(take|add) (5xx|cmpe 5.*).*': 'Which grade are you?', #7  
r'^7_(4|fourth|4th|senior)(\w*)': 'What is your GPA?', #8
```

Since the starting index of this sequence is 7, it is put before the next item in the sequence. All of the sequence starter patterns are stored in a list called **sequences**.

Some pattern's values consist '#'. This means when a group is detected, '#' is replaced by this detected group.

In this dictionary, a very small portion of the information is covered. You can again expand it with writing more patterns. If you want to add a sequence just examine a similar example.

The received message is preprocessed by a function called **preprocess**. It takes the message and returns the processed form it. It uses **remove\_punctuations**, **replace\_clitics**, **case\_folding** and

**tokenizer** functions. For further improvements about this project, the unchanged form of the words are kept.

**tokenizer:** Takes a string and split it by spaces into words. Each word is thought as a token.

**remove\_punctuations:** It is really simple function that removes punctuation at the end of the sentence.

**replace\_clitics:** This function first runs tokenizer to get words with “ ’ ”. After it gets them it replaces the clitic with the regarding word.

**case\_folding:** Again this function gets words by using tokenizer function. Then it converts the message to lowercase form. For improvements it also returns the unchanged form of the words.

After the received message is preprocessed it is put between **seq\_prefix** and **seq\_suffix** parameters. They are used for sequences and by default they are empty. When the received message is detected by the patterns in sequences, **seq\_started** is set to True and **seq\_prefix** parameter is set to “<index of the sequence starter>\_”. Then, program puts this prefix before the next message and searches this pattern. If it matches then this prefix is set to default and **seq\_suffix** is set to “\_<index of the sequence>”. By this logic sequences are controlled.

The program generates different answers by using some condition parameters. **seq\_started** is one of them. Others are **matched** and **is\_responded**. **matched** controls whether the received message matches with a pattern in the **patterns** dictionary or not whereas **is\_responded** becomes true whenever chat bot produces a response after matching.

Briefly, the text message is received, preprocessed, looked for a match with a pattern in the knowledge base and a regarding answer is generated.

## EXAMPLES

```
alim@alim-CX61-2PC:~/Desktop/NLP$ python3 chat_bot.py
> Hi, welcome. I am CHATYAT. How can I help you?
Please write your questions as gramatically correct as possible. Also please avoid misspelling and multiple questions in one sentence. :)
You: Hi
> Hi there.
You: I have a question
> Ok, please go on.
You: Can I take CMPE 561
> Which grade are you in?
You: fourth
> What is your GPA?
You: 3.23
> Yes you can, then.
You: How about 6xx courses, may I take 6xx courses?
> No, undergrad students can't take 6xx courses.
You: Thank you
> It is my pleasure.
You: Bye
> Bye
You: █
```

```

alim@alim-CX61-2PC:~/Desktop/NLP$ python3 chat_bot.py
> Hi, welcome. I am CHATYAT. How can I help you?
Please write your questions as gramatically correct as possible. Also please avoid misspelling and multiple questions in one sentence. :)
You: I've a problem
> Ok, please go on.
You: I want to drop cmpe 343
> Which grade are you in?
You: 3
> Do your total credits get under 18 when you drop/withdraw it or is this a course that you are repeating becaus of getting F?
You: yes
> Students' taking 15 credits depends on the advisor's decision.it depends on the board's decision if it goes to 11. Also you can't drop or withdraw an F course.
You: So sad
> I'm sorry, I do not understand your question. Please try it with different words.
You: 

```

```

alim@alim-CX61-2PC:~/Desktop/NLP$ python3 chat_bot.py
> Hi, welcome. I am CHATYAT. How can I help you?
Please write your questions as gramatically correct as possible. Also please avoid misspelling and multiple questions in one sentence. :)
You: I'veee a question
> I think you've misspelled something.
> I'm sorry, I do not understand your question. Please try it with different words.
You: I've a question
> Ok, please go on.
You: What is Gpa
> It is Grade Point Average .
You: thanks
> It is my pleasure.
You: 

```

## IMPROVEMENTS AND EXTENSIONS

The chat bot program is designed very simple and it has a few patterns to check. First, that pattern list should be expanded. That applies to meanings dictionary too. Fortunately, since the design is simple and clear, anybody can easily add some patterns to it. The order of the patterns are important, so don't change it.

The logic of the program is simple but the details can make it look good and sophisticated. The unchanged form of the words (before case folding runs) are also kept for further usage but it unused. On the other hand, a few patterns answer using the words in the questions and the same answers are printed. These can be improved, the answer of a question can be selected from a list that stores various answers to that question.

## DIFFICULTIES ENCOUNTERED

The most difficult part for me was creating the logic of the loop. It was hard to decide what to do after you get the message, which preprocessing steps would be helpful and how to form sequences. Thinking about different possibilities and designing the program considering all of them is hard too. After each step, I encountered a problem and went back to design and made some changes on it. Also, while testing the program there were some bugs and errors that I had to remove.

## CONCLUSION

This is a project written in Python which aims to answer undergrad students' questions automatically by using regular expressions. In other words, it is design of a chat bot which understands questions and answers considering different conditions. Chat bot can create complex sentences and asks for details.

The program doesn't need any parameters to be set or it doesn't need any input file. It runs on a terminal and executes until the user terminates it. The program needs regular expression module(re) and termcolor module for printing in different colors.

For now, it can answer to a few number of questions but it is open to improve. It is easy to add new patterns to it.

## APPENDIX

```
import re
from termcolor import colored
```

```
#The message parameters. As it is understood from the names these are presented in the defined situations.
```

```
WELCOME_MSG = "> Hi, welcome. I am CHATYAT. How can I help you?\n\
```

```
Please write your questions as gramatically correct as possible. Also please avoid misspelling and \
multiple questions in one sentence. :)"
```

```
NO_MATCH_ANSWER = "> I don't know the answer of your question. Is it possible that you
misspelled something?"
```

```
DEFAULT_RESPONSE = "> I'm sorry, I do not understand your question. Please try it with
different words. "
```

```
NOT_UNDERSTOOD_ANSWER = '> Sorry, I couldn\'t get your answer. Is it possible that you
write something wrong? If you want to \
ask another question please write \'nq_\.'
```

```
#This char_set is used to turn a text into lowercase.
```

```
char_set = {'A':'a', 'B':'b', 'C':'c', 'D':'d', 'E':'e',
```

```
'F':'f', 'G':'g', 'H':'h', 'I':'i', 'J':'j', 'K':'k',
```

```
'L':'l', 'M':'m', 'N':'n', 'O':'o', 'P':'p', 'Q':'q',
```

```
'R':'r', 'S':'s', 'T':'t', 'V':'v', 'U':'u', 'X':'x',
```

```
'Y':'y', 'Z':'z', 'W':'w'}
```

```
#Clitic set that is used to replace clitics with regarding words.
```

```
clitic_set = {
```

```
'll' : 'will',
```

```
'm' : 'am',
```

```
're' : 'are',
```

```
'd' : 'would',
```

```
've' : 'have',
```

```
't' : 'not'
```

```
}
```

```
#Meaning dictionary keeps the meaning of some words and abbreviations.
```

```

meanings = {
'gpa': 'Grade Point Average',
'spa': 'Semester Point Average',
'repeat': 'the status of a student whose GPA is under 2.0 and whose SPA is also under 2.0 for two semesters',
'graduation credits': '145',
'senior': 'the student who have completed 108 credits',
'senior student': 'the student who has completed 108 credits',
'fourth grade': 'the student who has completed 108 credits',
'irregular': 'the student starts the university in spring.'
}

```

# The main knowledgebase of the system.

# Questions are detected by regular expressions and the regarding answer is generated.

# There are some sequences in the knowledge base. If a key starts with a number with underscore

# that means follows the key stated with that number. Similarly, if a key ends with underscore

# and number that means it is the next sentence in the sequence.

```

patterns = {
r'.*what does (.+) (mean|stand for).*': 'It is # .', #1
r'.*what is (.+).*': 'It is # .', #2
r'.*need approval for (.+).*': 'Yes, you need approval from your advisor for # .', #3
r'.*course from(.school of.*)?applied science.*': 'No, you can\'t take credit course from school of applied sciences',
r'.*(least|min|minimum) credits.*graduation.*': '145 credits.',
r'.*reject.*': 'You should review your program and make changes on it. \
Don\'t forget to send it to approval before due date. Otherwise you have to register \
in late registration period and need to pay registration money.', #4
r'.*(take|add) (5xx|cmpe 5.*).*': 'Which grade are you in?', #7
r'^7_(4|fourth|4th|senior)(\w*)': 'What is your GPA?', #8
r'^7_(1|2|3|third|3rd|first|1st|second|2nd|junior|freshman|sophomore)(\w*)': 'You need to be fourth grade to take 5xx courses.', #9
r'(~\d)*((([0|1][\.,])([2|\.,][0|1|2|3|4]))(\w*)_7$': 'No you can\'t. Your GPA should be greater than 2.5', #10
r'.*((([2|\.,])([5|6|7|8|9])))([4|\.,][0+])([3|\.,][\d*])_7$': 'Yes you can, then.', #11
r'.*(drop|withdraw).*(course|cmpe).*': 'Which grade are you in?', #12
r'^12_(1|first|freshman|1st)(\w*)': 'If you are an irregular student having GPA greater than 2.75 you can only change \
courses. Otherwise you cannot. In any ways your credits cannot go under 18', #13
r'^12_(2|3|4|sophomore|senior|junior|second|third|fourth|4th|3rd|2nd)(\w*)': 'Do your total credits get under 18 when \
you drop/withdraw it or is this a course that you are repeating because of getting F?',
r'(yes|yeap|unfortunately)_12$': 'Students\' taking 15 credits depends on the advisor\'s decision.\
it depends on the board\'s decision if it goes to 11. Also you can\'t drop or withdraw an F course.',
r'(no|nope|not)_12$': 'Yes you can then. I hope it is the best for you :)',
r'.*conflict.*': 'You should prepare a program without conflicts. If you have then it depends on your advisor.', #14
r'.*(take|add) 6xx.*': 'No, undergrad students can\'t take 6xx courses.', #15
r'.*nq.*': "OK, I'm listening.", #16
r'(hi|.*hello|hey).*': "Hi there.", #17
r'.*(have|ask).*(question|problem).*': "Ok, please go on.", #18
r'thank.*': "It is my pleasure. ", #19
r'.*bye.*': "Bye"

```

```
}
```

```
# The index of sequence starting questions are stored in this list  
sequences=[6, 11]
```

```
# the list of all compiled regex expressions  
regex = map(lambda x:re.compile(x), patterns)  
regex = list(regex)
```

```
# The list of answers occur in patterns dictionary  
answers = list(patterns.values())
```

```
# A simple preprocessing function.
```

```
# It removes punctuation, replaces clitics with regarding words and does case folding for  
normalization.
```

```
def preprocess(msg):  
    if len(msg)<1:  
        return msg  
    msg = remove_punctuations(msg)  
    if len(msg)<1:  
        return msg  
    msg = replace_clitics(msg)  
    if len(msg)<1:  
        return msg  
    msg, changed_words = case_folding(msg)  
    if len(msg)<1:  
        return msg
```

```
    return msg, changed_words
```

```
# Removes punctuations at the end of the sentences.
```

```
def remove_punctuations(msg):  
    if msg[-1] in [',', '?', ':', ';', '']:  
        return msg[0:-1]  
    return msg
```

```
# Replaces clitics with regarding words.
```

```
# Returns error message if an undefined clitic is given
```

```
def replace_clitics(msg):  
    global clitic_set
```

```
    clitic = 'none'  
    words = tokenizer(msg)  
    msg_proc = "  
    for word in words:  
        i = 0  
        for char in word:  
            i += 1  
            if clitic != 'none':  
                if(char == 's'):  
                    clitic = 'none'  
                    break
```



```

clitic += char
if char == "\":
clitic = "
word = word[0:i-1]

if clitic != 'none':
try:
msg_proc = msg_proc + word + ' ' + clitic_set[clitic] + ' '
clitic = 'none'
except:
print(colored("> I think you've misspelled something.", 'green'))
return msg
else:
msg_proc = msg_proc + word + ' '

return msg_proc[0:-1]

```

```

# Splits a text by spaces
# Tokens can be thought as words.
def tokenizer(str):
str += ' '
words = []
word = ""
for char in str:
if char == ' ':
words.append(word)
word = ""
else:
word += char
return words

```

```

# Turns a given text into lowercase
def case_folding(msg):
global char_set

```

```

words = tokenizer(msg)
changed_words = {}
msg_lower = ""

for word in words:
if re.search('.*[A-Z].*', word):
word_lower = ""
for char in word:
if char in char_set.keys():
char = char_set[char]
word_lower += char
changed_words[word] = word_lower
word = word_lower
msg_lower = msg_lower + word + ' '
msg_lower = msg_lower[0:-1]
return msg_lower, changed_words

```

```
#####
###
#####
###
##### MAIN LOOP
#####
#####
###
#####
###
```

```
print(colored(WELCOME_MSG, 'green'))
```

```
seq_started = False #Checks whether any question that starts a sequence is asked
seq_prefix = " #For the detection of the first answer regarding sequence
seq_suffix = " #For the detection of the second answer regarding sequence
```

```
while True:
    msg = input('You: ')
    msg, _ = preprocess(msg)
    if seq_started:
        msg = seq_prefix + msg + seq_suffix

    for idx, exp in enumerate(regex):
        is_responded = False
        no_match = False
        match = exp.match(msg)
        if match:
            if idx == 15: #15 is the index of sequence quitter.
                seq_started = False
                seq_prefix = ""
                seq_suffix = ""
            elif idx in sequences: #Checks whether the question starts a sequence or not
                seq_started = True
                seq_prefix = str(idx+1) + '_'
            elif seq_prefix != "":
                seq_suffix = '_' + seq_prefix[0:-1]
                seq_prefix = ""
            else:
                seq_started = False
                seq_suffix = ""
        words_in_ans = tokenizer(answers[idx])
        ans = '> '
        for word in words_in_ans:
            if word == '#':
                try:
                    word = meanings[match.group(1)]
                except:
                    if idx < 2:
                        print(colored(NO_MATCH_ANSWER, 'green'))
                    no_match = True
                    is_responded = True
```

```
word = match.group(1)
ans = ans + word + ' '
if not no_match:
    print(colored(ans[0:-1], 'green'))
    is_responded = True
    break

if not is_responded:
    if seq_started:
        print(colored(NOT_UNDERSTOOD_ANSWER, 'green'))
    else:
        print(colored(DEFAULT_RESPONSE, 'green'))
```