National University
Of Computer and Emerging Sciences

Name:

Dawood Sarfraz

Roll no:

20P-0153

Section:

BSCS-7A

Course:

Natural Language Processing

Lab:

04

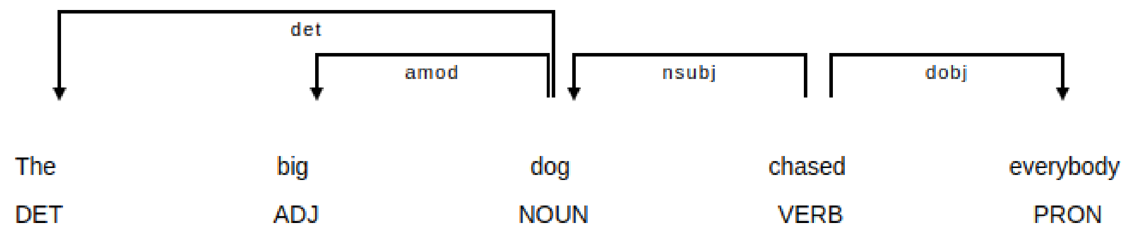
Submitted to:

Dr. Omer Usman Khan

FAST National University of Computer and Emerging Sciences

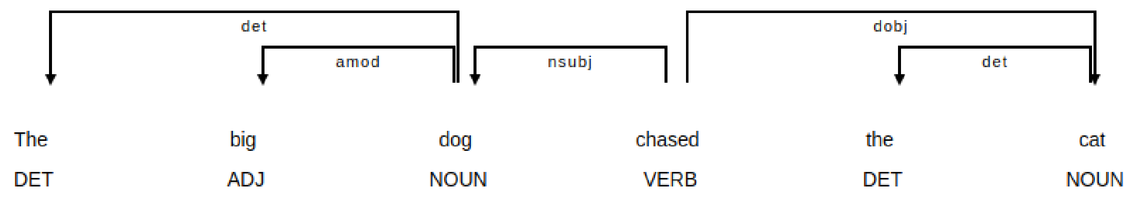
1.

What text and dependencies did the above code catch for the sentence “The big dog chased everybody”



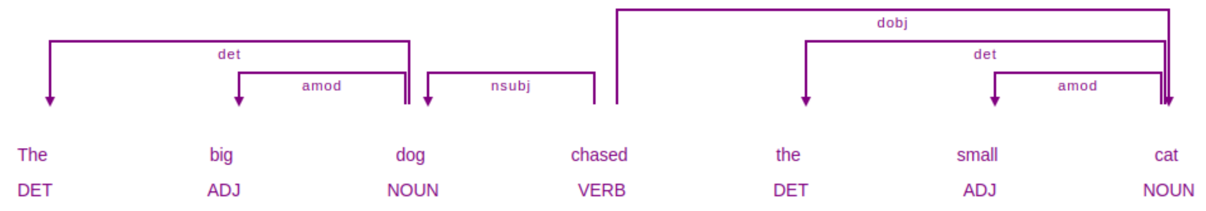
```
Matching Sentence: dog chased everybody
Pattern Type: SubRootObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: everybody--dobj
```

2. Change the sentence to “The big dog chased the cat”. Does the pattern catch the SVO pattern? If not, add another pattern2 to the matcher. The pattern should be DEP: nsubj, DEP: ROOT, DEP: det, DEP:dobj. When done, update matcher.add("SubRootDetObject", [pattern2])



```
Matching Sentence: dog chased the cat
Pattern Type: SubRootDetObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: the--det
Dependency: cat--dobj
```

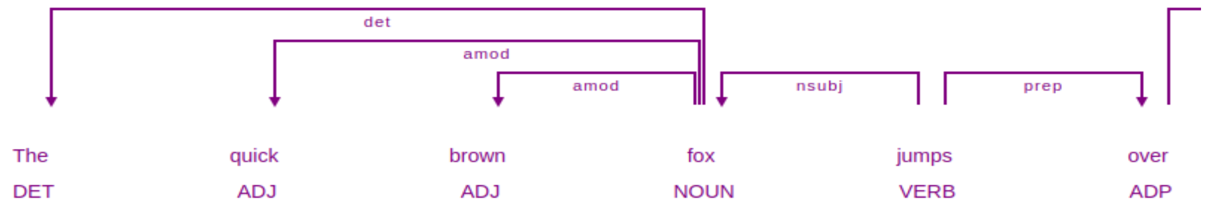
3. Now design a third pattern for the sentence “The big dog chased the small cat”.

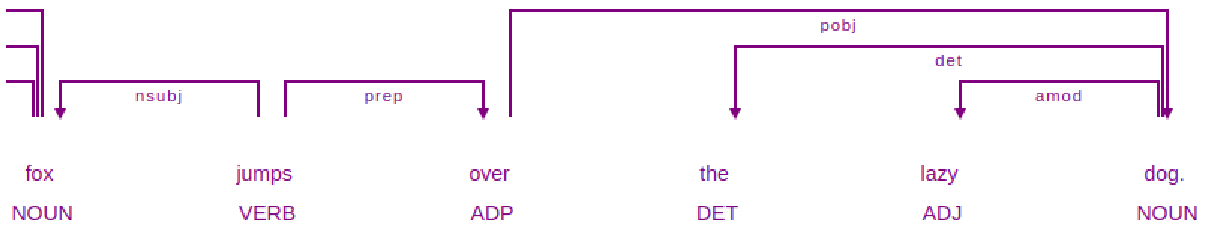


```
Matching Sentence: dog chased the small cat
Pattern Type: SubRootAdjDetObject
Dependency: dog--nsubj
Dependency: chased--ROOT
Dependency: the--det
Dependency: small--amod
Dependency: cat--dobj
```

1. Design a pattern to identify a noun at least one time.

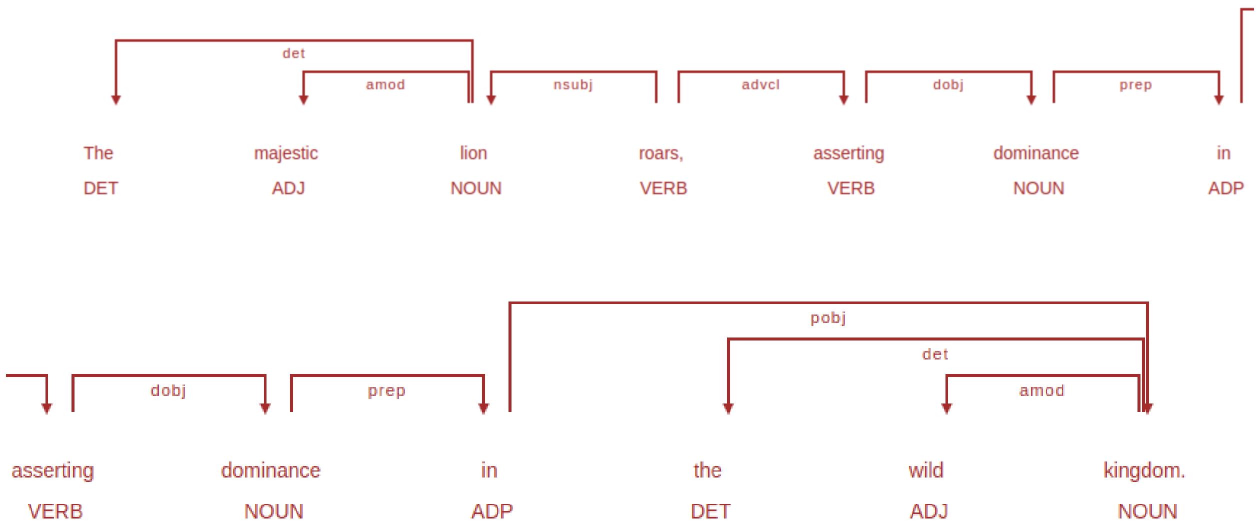
```
In [9]: pattern = [{"POS": "NOUN", "OP": "*"}]
matcher.add("OneOrMoreNouns", [pattern])

doc = nlp("The quick brown fox jumps over the lazy dog.")

matches = matcher(doc)

displacy.render(doc, style='dep',options ={ "color" : "purple","compact":True} )

# Iterate over matches
for pattern_id, start, end in matches:
    print("Matching Sentence:", doc[start:end].text)
    print("Pattern Type:", doc.vocab.strings[pattern_id])

    # Print information for each token in the match
    for token in doc[start:end]:
        print(f"Dependency: {token.text}-{token.dep_}")
        print("-"*20)
```





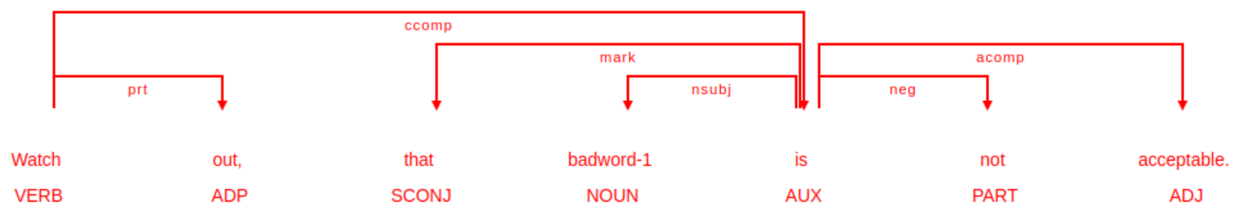
2. Design a pattern to identify a noun of length >= 10 characters.

```
pattern = [{"POS": "NOUN", "LENGTH": {">=": 10}}]
matcher.add("LongNoun", [pattern])

doc = nlp("The majestic lion roars, asserting dominance in the wild kingdom.")

matches = matcher(doc)

displacy.render(doc, style='dep', options ={ "color" : "brown","compact":True})
```



2. Design a pattern to identify vulgar language (Hint: you will need usage of IN, or NOT_IN).

```
: bad_word = ["badword-1", "badword-2", "badword-3"]

  pattern = [{"LOWER": {"IN": bad_word}}]
  matcher.add("VulgarLanguage", [pattern])

  doc = nlp("Watch out, that badword-1 is not acceptable.")

  matches = matcher(doc)

  displacy.render(doc, style='dep', options ={ "color" : "red","compact":True})

  for pattern_id, start, end in matches:
      print("Matching Sentence:", doc[start:end].text)
      print("Pattern Type:", doc.vocab.strings[pattern_id])

      for token in doc[start:end]:
          print(f"Token: {token.text} | POS: {token.text.lower()}")
```

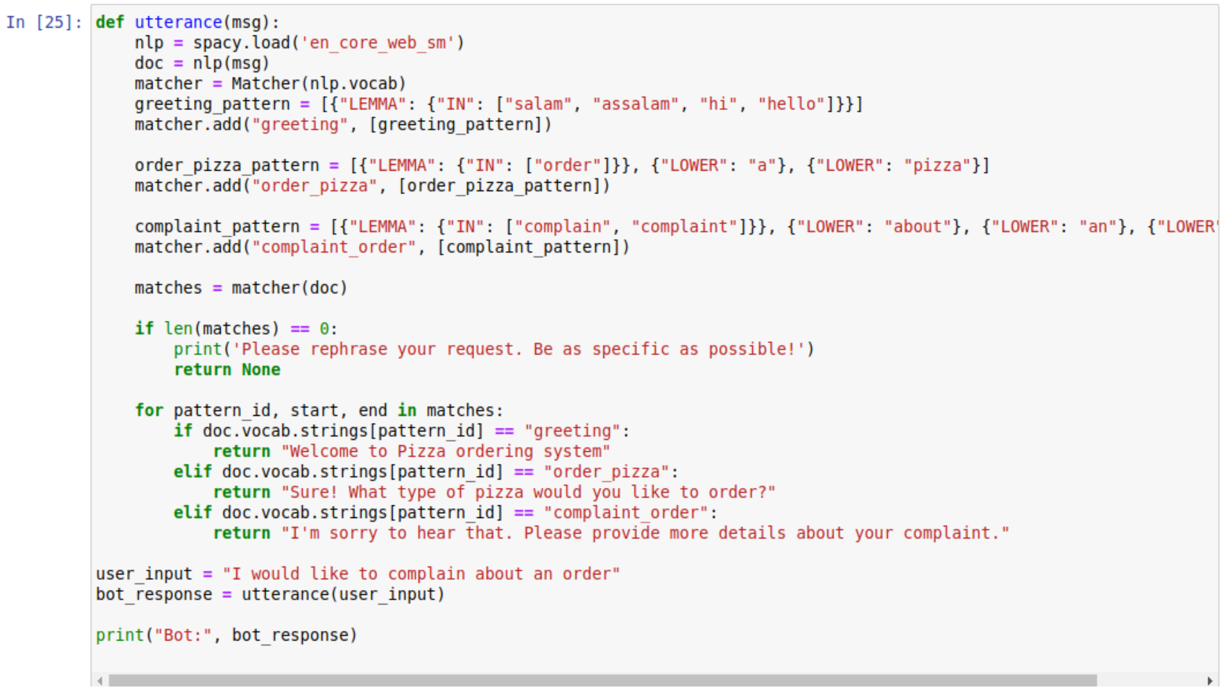


Task 6: Replies

1. **Extend the code by adding pattern and matches if a user enters: “I would like to order a pizza”. The bot should ask about which pizza type he/she wants.**

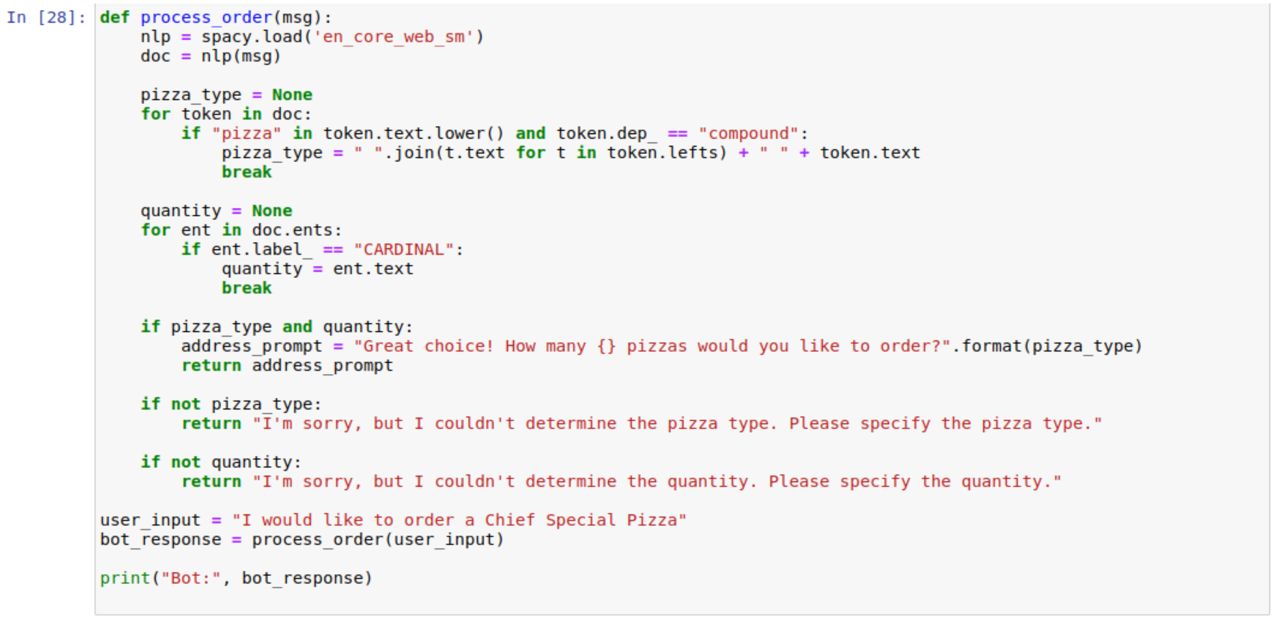
```
In [23]: def utterance(msg):
             nlp = spacy.load('en_core_web_sm')
             doc = nlp(msg)
             matcher = Matcher(nlp.vocab)

             greeting_pattern = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
             matcher.add("greeting", [greeting_pattern])

             order_pizza_pattern = [{"LEMMA": {"IN": ["order"]}}, {"LOWER": "a"}, {"LOWER": "pizza"}]
             matcher.add("order_pizza", [order_pizza_pattern])

             matches = matcher(doc)

             if len(matches) == 0:
                 print('Please rephrase your request. Be as specific as possible!')
                 return None

             for pattern_id, start, end in matches:
                 if doc.vocab.strings[pattern_id] == "greeting":
                     return "Welcome to Pizza ordering system"
                 elif doc.vocab.strings[pattern_id] == "order_pizza":
                     return "Sure! What type of pizza would you like to order?"


         user_input = "I would like to order a pizza"
         bot_response = utterance(user_input)

         print("Bot:", bot_response)
```

2. **Extend the code by adding pattern and matches if a user enters: "I would like to complain about an order".**



```
In [25]: def utterance(msg):
             nlp = spacy.load('en_core_web_sm')
             doc = nlp(msg)
             matcher = Matcher(nlp.vocab)
             greeting_pattern = [{"LEMMA": {"IN": ["salam", "assalam", "hi", "hello"]}}]
             matcher.add("greeting", [greeting_pattern])

             order_pizza_pattern = [{"LEMMA": {"IN": ["order"]}}, {"LOWER": "a"}, {"LOWER": "pizza"}]
             matcher.add("order_pizza", [order_pizza_pattern])

             complaint_pattern = [{"LEMMA": {"IN": ["complain", "complaint"]}}, {"LOWER": "about"}, {"LOWER": "an"}, {"LOWER
             matcher.add("complaint_order", [complaint_pattern])

             matches = matcher(doc)

             if len(matches) == 0:
                 print('Please rephrase your request. Be as specific as possible!')
                 return None

             for pattern_id, start, end in matches:
                 if doc.vocab.strings[pattern_id] == "greeting":
                     return "Welcome to Pizza ordering system"
                 elif doc.vocab.strings[pattern_id] == "order_pizza":
                     return "Sure! What type of pizza would you like to order?"
                 elif doc.vocab.strings[pattern_id] == "complaint_order":
                     return "I'm sorry to hear that. Please provide more details about your complaint."

         user_input = "I would like to complain about an order"
         bot_response = utterance(user_input)

         print("Bot:", bot_response)
```

2. In respone to what pizza type user wants, the user may want to enter "Chief Special Pizza". Use the .lefts (mentioned in Lab 03) to get the pizza type. Ask about quantity. Use Cardinal as ent type to get the quantity, and place the order. Ask for address, and confirm the user with address.



```
In [28]: def process_order(msg):
             nlp = spacy.load('en_core_web_sm')
             doc = nlp(msg)

             pizza_type = None
             for token in doc:
                 if "pizza" in token.text.lower() and token.dep_ == "compound":
                     pizza_type = " ".join(t.text for t in token.lefts) + " " + token.text
                     break

             quantity = None
             for ent in doc.ents:
                 if ent.label_ == "CARDINAL":
                     quantity = ent.text
                     break

             if pizza_type and quantity:
                 address_prompt = "Great choice! How many {} pizzas would you like to order?".format(pizza_type)
                 return address_prompt

             if not pizza_type:
                 return "I'm sorry, but I couldn't determine the pizza type. Please specify the pizza type."

             if not quantity:
                 return "I'm sorry, but I couldn't determine the quantity. Please specify the quantity."

         user_input = "I would like to order a Chief Special Pizza"
         bot_response = process_order(user_input)

         print("Bot:", bot_response)
```