# Dependency Parser

whenever we are having some text, we calculate tf-idf of the vector (represents w importance of word). here there are no semantics. but irl the position of the word greatly affects the meaning.

for example: I am going for a class. here every word is in correct position. if the position of "class" is changed it doesnt make any sense. this is not addressed in tfidf. no meaning preserved.

we need to figure out how one word is related or dependent on anothe word. for this we use dependency parser. we will create a tree which is weighted and directed.

## Source Code

In [1]:
```
import spacy
```

In [3]:
```
def dependency_parse(sentence):
    nlp = spacy.load("en_core_web_sm")
    doc = nlp(sentence)
    dependencies = []
    for token in doc:
        dependencies.append((token.text,token.dep_,token.head.text))
    return dependencies

sentence = "My name is Ahana Sadh. My subject is NLP. I do not like this class."
dependencies = dependency_parse(sentence)
for dep in dependencies:
    print(dep)
```

```
('My', 'poss', 'name')
('name', 'nsubj', 'is')
('is', 'ROOT', 'is')
('Ahana', 'compound', 'Sadh')
('Sadh', 'attr', 'is')
('.', 'punct', 'is')
('My', 'poss', 'subject')
('subject', 'nsubj', 'is')
('is', 'ROOT', 'is')
('NLP', 'attr', 'is')
('.', 'punct', 'is')
('I', 'nsubj', 'like')
('do', 'aux', 'like')
('not', 'neg', 'like')
('like', 'ROOT', 'like')
('this', 'det', 'class')
('class', 'dobj', 'like')
('.', 'punct', 'like')
```

In [4]:
```
import spacy
```

In [5]:
```
from spacy import displacy
```

In [6]:
```
nlp = spacy.load("en_core_web_sm")
```

In [7]:
```
sentence = "My name is Ahana Sadh. My subject is NLP. I do not like this class."
```

In [8]:
```
doc = nlp(sentence)
```

In [9]:
```
displacy.render(doc,style="dep",jupyter= True)
```

My PRON name NOUN is AUX Ahana PROPN Sadh. PROPN My PRON subject NOUN is AUX NLP. PROPN I PRON do AUX not PART like VERB this DET class. NOUN poss nsubj compound attr poss nsubj attr nsubj aux neg det dobj

In [11]:
```
sentence = "The quick brown fox jumps over the lazy dog"
```

In [12]:
```
dependency_rules = {
    'jumps':{'nsubj':'fox','prep':'over'},
    'fox':{'det':'The','amod':'quick','amod':'brown'},
    'over':{'det':'the','amod':'lazy','pobj':'dog'}
}
```

In [13]:
```
def extract_dependencies(sentence,dependency_rules):
    dependencies = []
    words = sentence.split()
    for i, word in enumerate(words):
        if word in dependency_rules:
            for dep,dep_word in dependency_rules[word].items():
                dependencies.append((dep_word,dep,word))
    return dependencies
```

In [14]:
```
dependencies = extract_dependencies(sentence,dependency_rules)
```

In [15]:
```
for dep in dependencies:
    print(dep)
```

```
('The', 'det', 'fox')
('brown', 'amod', 'fox')
('fox', 'nsubj', 'jumps')
('over', 'prep', 'jumps')
('the', 'det', 'over')
('lazy', 'amod', 'over')
('dog', 'pobj', 'over')
```

## Assignment

### Create Dependency Parsers for multilingual sentence. For example: "Bonjour, I'm fine. Aujourd'hui, j'ai mangé some food and maintenant je suis sleepy."

In [21]:
```
dependency_rules = {
    "I'm": {'ROOT': 'Bonjour'},
    'fine': {'nsubj': "I'm"},
    'mangé': {'nsubj': 'j\'ai', 'nmod': 'Aujourd\'hui', 'dobj': 'food'},
    "j'ai": {'ROOT': 'Bonjour'},
    'food': {'det': 'some'},
    'sleepy': {'nsubj': 'je suis', 'nmod': 'maintenant'},
    'je': {'nsubj': 'suis', 'nmod': 'maintenant'},
}
```

In [22]:
```
sentence = "Bonjour, I'm fine. Aujourd'hui, j'ai mangé some food and maintenant je suis sleepy"
```

In [23]:
```
dependencies = extract_dependencies(sentence,dependency_rules)
```

In [24]:
```
for dep in dependencies:
    print(dep)
```

```
('Bonjour', 'ROOT', "I'm")
('Bonjour', 'ROOT', "j'ai")
("j'ai", 'nsubj', 'mangé')
("Aujourd'hui", 'nmod', 'mangé')
('food', 'dobj', 'mangé')
('some', 'det', 'food')
('suis', 'nsubj', 'je')
('maintenant', 'nmod', 'je')
('je suis', 'nsubj', 'sleepy')
('maintenant', 'nmod', 'sleepy')
```

In [ ]:
```

```