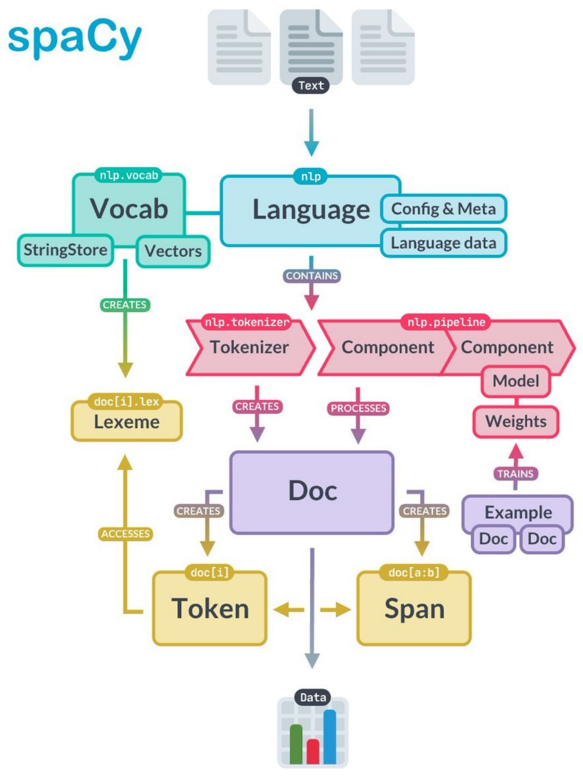
Lab 03 NLTK+spaCy for NLP Apps

spaCy[1] is an open source library for NLP application and is written in python and cython. Compared to NLTK which is used for research and prototyping, spaCy is used to create actual products. The full spacy pipeline can be represented through the following figure:



Task 1: Spacy Installation

It is recommended to run spaCy in a python virtual environment. To create a virtual environment in a directory, carry out the following steps from python shell. The first command creates a virtual environment, whereas the second activates it:

```
mkdir chatbot

cd chatbot

python -m venv .env

source .env/bin/activate
```

Once activity, populate the environment with some tools, libraries, and models of common use:

```
pip install -U pip setuptools wheel

pip install -U spacy

python -m spacy download en_core_web_sm
```

And that's it. The above is for first time use only. The next time you go to the directory, you will only need to activate it only:

```
source .env/bin/activate
```

For jupyter, you need to carry out some additional steps from within the virtual environment. This is optional.

```
pip install ipykernel

ipython kernel install --user --name=spaCy
```

Then, you can exit the virtual environment and launch the jupyter kernel. From any ipynb file, you can select:

Kernel → Change Kernel, and select spaCy from the drop down list.

Task 2: spaCy Hello World

We start from the central component of a spaCy object, the Doc() object. Run the following code:

```
from spacy.tokens.doc import Doc

from spacy.vocab import Vocab

doc = Doc(Vocab(), words = [u'Hello', u'World!'])

print(doc)
```

1 URL: https://spacy.io

```
print(type(doc))
print(doc.vocab)
for token in doc:
    lexeme = doc.vocab[token.text]
    print(lexeme.text)
```

Attempt the following Questions:

1. The Vocab() object belong to which class?
2. What is a Lexeme object (You need to check the API)

From the doc() object, we move to word level represented as tokens. Try the following:

```
import spacy
nlp = spacy.load('en_core_web_sm')
doc = nlp(u'I want to learn spaCy.')
token_text1 = [token.text for token in doc]
token_text2 = [doc[i].text for i in range(len(doc))]
print(token_text1)
print(token_text2)
```

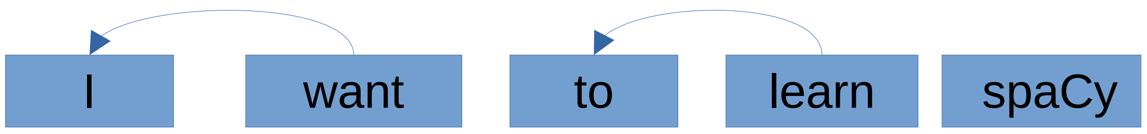
We explore the concept of a spaCy token container in above code. Attempt the following questions:

1. What is en_core_web_sm?
2. What is the size of en_core_web_sm?
3. What other variations can be used?

Taking the second token example, we can write the code in a different way as follows:

```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].lefts])
```

The output would show the following representation:



The diagram shows the left dependencies of a word. Now, modify the code to the following:

```
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].rights])
    print([t for t in doc[i].children])
```

You can view all the dependencies graphically by running:

```
from spacy import displacy
```

```
displacy.render(doc, style='dep')
```

Now, attempt the following:

1. Draw the left and right dependencies for the sentence: I want to learn spaCy.
2. Draw the children for the sentence: I want to learn spaCy.
3. Draw the left and right dependencies for the sentence: I would very much want to eat a hot dinner.
4. Present a list of all dependency grammars of your sentences above.

Task 3: NLTK vs spaCy Pipelines

An NLTK Pipeline typically has the following steps:

Step 01: Take Raw Text

```
texts = [u"We are nearing the end of the semester at Peshawar. Final exams of the Fall 2023 semester will start soon."]
```

Step 02: Do Sentence Level Segmentation

```
for text in texts:

    sentences = nltk.sent_tokenize(text)

    print(sentences)
```

Step 03: Extract Tokens in Sentences

```
    for sentence in sentences:

        words = nltk.word_tokenize(sentence)

        print(words)
```

Step 04: Perform Part of Speech Tagging

```
        tagged_words = nltk.pos_tag(words)

        print(tagged_words)
```

Step 05: Detect Named Entity of Tagged Words

```
        ne_tagged_words = nltk.ne_chunk(tagged_words)

        print(ne_tagged_words)
```

From step 03 onwards, there may be variations depending on the type of NLP task that needs to be performed. To get the above to work, you will require to download the following packages in NLTK

```
nltk.download('punkt')                      # Sentence Tokenize

nltk.download('averaged_perceptron_tagger') # POS Tagging

nltk.download('maxent_ne_chunker')          # Named Entity Chunking

nltk.download('words')                      # Word Tokenize
```

To attempt the NLP pipeline in spaCy, use the following:

```
from spacy import displacy

doc = nlp(u'We are nearing the end of the semester at Peshawar. Final exams of the Fall 2023 semester will start soon.')

displacy.render(doc, style='ent')
```

```
for ent in doc.ents:
    print(ent.text, ent.label_)
```

Attempt the following questions:

1. How did the Named Entity Output of the NLTK pipeline look like? Present its output.
2. How did the Named Entity Output of the spaCy pipeline look like? Present its output.

To view the sequence of commands in a spaCy pipeline, you can view the following:

```
import spacy
nlp = spacy.load('en_core_web_sm')
nlp.pipe_names
```

Attempt the following questions:

1. What is the default pipeline structure of spaCy?

If we are not interested in any specific tool in the pipeline, you can disable it as follows:

```
nlp = spacy.load('en_core_web_sm',disable=['parser'])
```

To add a custom component to the pipeline, you can use the following template from the spaCy documentation:

```
from spacy.language import Language
@Language.component("my_component")
def my_component(doc):
   # Do something to the doc here
   return doc
```

Then, add it to the NLP spaCy pipeline as follows:

```
nlp.add_pipe("my_component")
```

If you want to control its positioning, you can use:

```
nlp.add_pipe("my_component", first=True)
nlp.add_pipe("my_component", last=True)
nlp.add_pipe("my_component", before="parser")
nlp.add_pipe("my_component", after="parser")
```

Task 4: Finding Patterns in Sentences

In many applications, it is necessary to find patterns in a sentence. For example, differentiation between the sentences “I want to learn spaCy” and “How do I learn spaCy” is important in a chat-bot type of setup so that a chatbot may understand whether the user is asking a question or making a statement? Consider the code below:

```
doc = nlp(u'I want to learn spaCy.')
displacy.render(doc, style='dep')
```

This gives us an output showing the subject, object, verb, and other dependencies in a sentence. Compare it to the following:

```
doc = nlp(u'How do I learn spaCy.')
displacy.render(doc, style='dep')
```

We can identify patterns (hard-coded) as follows:

```
import spacy

nlp = spacy.load('en_core_web_sm')

def dep_pattern(doc):

    for i in range(len(doc)-1):

        print(doc[i].dep_)

            if doc[i].dep_ == 'nsubj' and doc[i+1].dep_ == 'ROOT' and doc[i+2].dep_ ==
'acomp':

            return True

    return False

doc = nlp(u'How do I learn spaCy.')

if dep_pattern(doc):

    print('Found')

else:

    print('Not found')
```