Task 1: Spacy Installation

Task 2: spaCy Hello World

In [1]:
```python
from spacy.tokens.doc import Doc

from spacy.vocab import Vocab

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

In [2]:
```python
print(type(doc))
print(doc.vocab)
for token in doc:
    lexeme = doc.vocab[token.text]
    print(lexeme.text)
```
```
<class 'spacy.tokens.doc.Doc'>
<spacy.vocab.Vocab object at 0x7fb88de569e0>
Hello
World!
```

Questions:

1. The `Vocab()` object belongs to which class?

Belongs to Class:

- The `Vocab()` object belongs to the `spacy.vocab.Vocab` class.

Purpose:

- Serves as a memory-efficient container.
- Manages the vocabulary of the language model.
- Facilitates quick access to word representations.

---

2. What is a lexeme object (You need to check the API)?

Definition:

- The Lexeme object allows access to properties and features associated with individual words.

Purpose:

- Provides a foundation for linguistic analysis.
- Essential for various natural language processing tasks.

API Check:

- The Lexeme object is an instance of the `spacy.lexeme.Lexeme` class.

In [3]:
```python
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)
```
```
['I', 'want', 'to', 'learn', 'spacy']
['I', 'want', 'to', 'learn', 'spacy']
```

Questions:

1. What is `en_core_web_sm`?

Answer: `en_core_web_sm` is a pre-trained English language model in Spacy. It belongs to the "web" models and offers various functions for performing natural language tasks.

---

2. What is the size of `en_core_web_sm`?

Answer: `en_core_web_sm` is an English multi-task CNN (Convolutional Neural Network) model trained on OntoNotes, and its size is approximately 11 MB.

---

3. What other variations can be used?

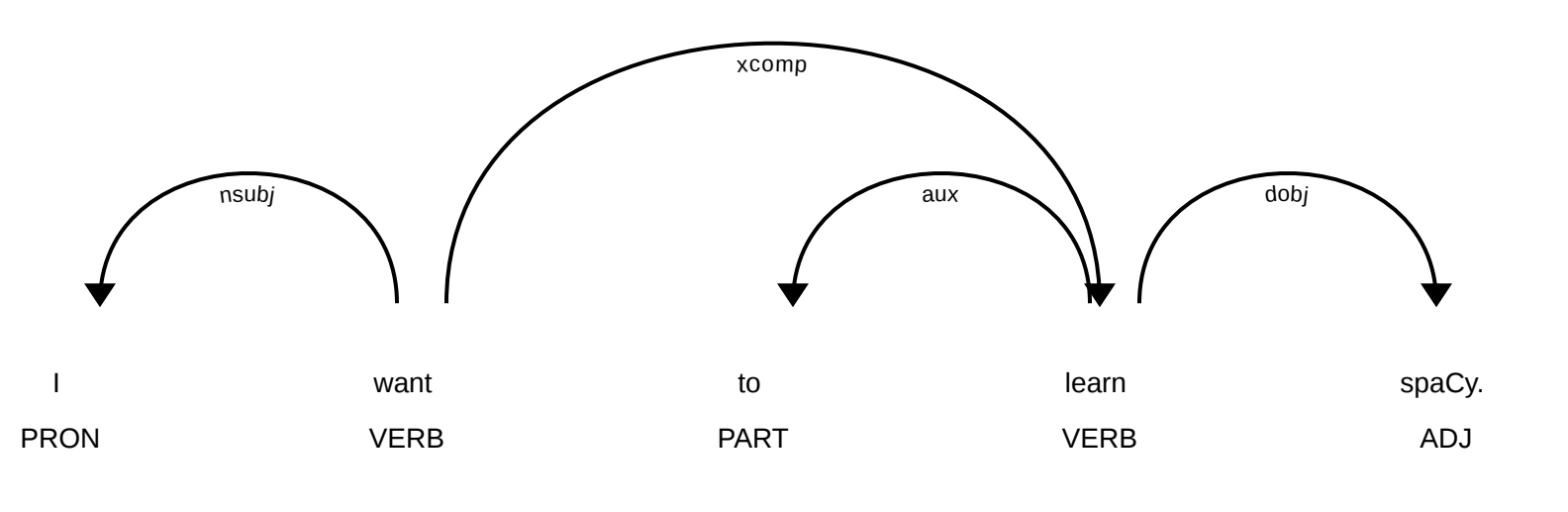
- **en_core_web_md:**
  - A medium-sized English model with word vectors. It includes more features and is larger than `en_core_web_sm`.
- **en_core_web_lg:**
  - A large-sized English model with word vectors. It is the largest among the English models provided by Spacy and includes even more features.
- **Specialized Models:**
  - Spacy offers models trained for specific tasks, such as `en_core_web_trf`, which is a transformer-based model. Transformer-based models may provide improved performance in certain scenarios.

In [4]:
```python
doc = nlp(u'I want to learn spaCy.')
for i in range(len(doc)):
    print([t for t in doc[i].lefts])
```
```
[]
[I]
[]
[to]
[]
[]
```

In [5]:
```python
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])
```
```
[]
[]
[learn, .]
[I, learn, .]
[]
[]
[spaCy]
[to, spaCy]
[]
[]
[]
[]
```

In [6]:
```python
from spacy import displacy
```

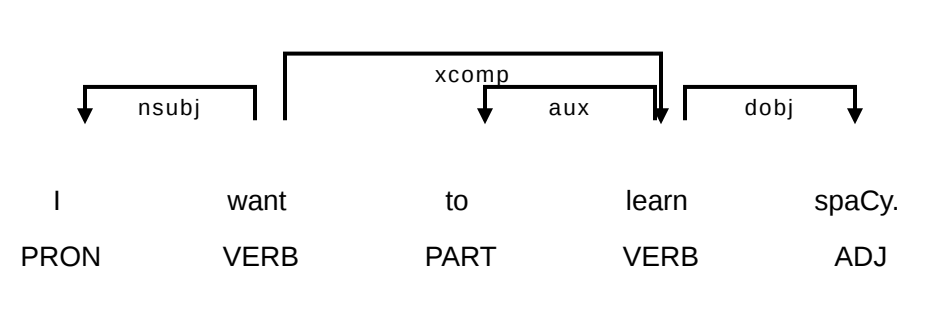
In [7]:
```python
displacy.render(doc, style='dep')
```



Questions

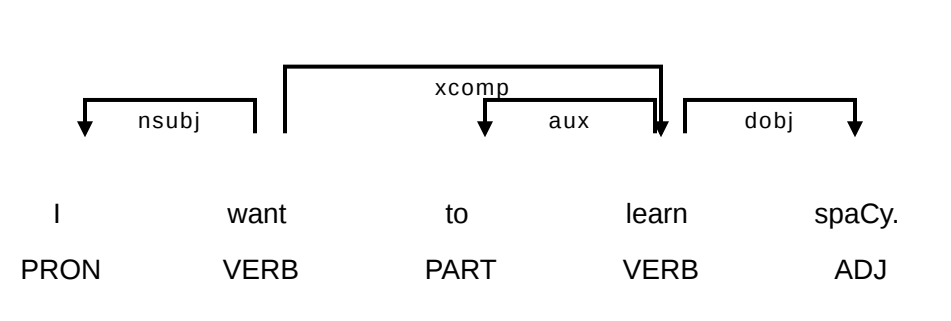
1. Draw the left and right dependencies for the sentence: I want to learn spaCy.

In [8]:
```python
# Draw left and right dependencies
displacy.render(doc, style='dep', options={'distance': 100, 'compact': True}, jupyter=True)
```



2. Draw the children for the sentence: I want to learn spaCy.

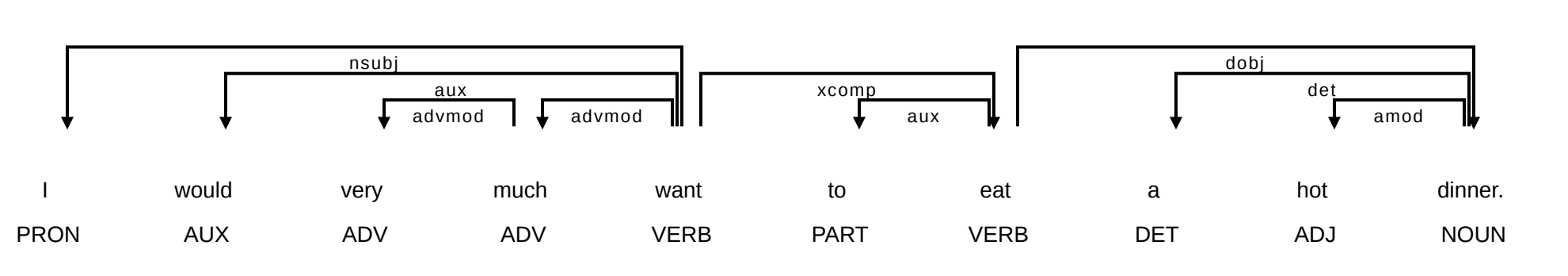
In [9]:
```python
# Draw children
displacy.render(doc, style='dep', options={'distance': 100, 'compact': True, 'mode': 'dep'}, jupyter=True)
```



3. Draw the left and right dependencies for the sentence: I would very much want to eat a hot dinner.

In [10]:
```python
# Process the second sentence
doc2 = nlp("I would very much want to eat a hot dinner.")

# Draw left and right dependencies
displacy.render(doc2, style='dep', options={'distance': 100, 'compact': True}, jupyter=True)
```



4. Present a list of all dependency grammars of your sentences above.

In [11]:
```python
# Display dependency grammars for the first sentence
for token in doc:
    print(f"{token.text}: {token.dep_}")
print("\n")

# Display dependency grammars for the second sentence
for token in doc2:
    print(f"{token.text}: {token.dep_}")
print("\n")
```
```
I: nsubj
want: ROOT
to: aux
learn: xcomp
spaCy: dobj
.: punct


I: nsubj
would: aux
very: advmod
much: advmod
want: ROOT
to: aux
eat: xcomp
a: det
hot: amod
dinner: dobj
.: punct
```

Task 3: NLTK vs spaCy Pipelines

Step 01: Take Raw Text

In [12]:
```python
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

In [13]:
```python
import nltk
```

In [14]:
```python
for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)
```
```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall 2023 semester will start soon.']
```

Step03: Extract Tokens in Sentences

In [15]:
```python
for sentence in sentences:
    words = nltk.word_tokenize(sentence)
    print(words)
```
```
['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at', 'Peshawar', '.']
['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start', 'soon', '.']
```

Step 04: Perform Part of Speech Tagging

In [16]:
```python
tagged_words = nltk.pos_tag(words)
print(tagged_words)
```
```
[('Final', 'JJ'), ('exams', 'NN'), ('of', 'IN'), ('the', 'DT'), ('Fall', 'NN'), ('2023', 'CD'), ('semester', 'NN'), ('will', 'MD'), ('start', 'VB'), ('soon', 'RB'), ('.', '.')]
```

Step 05: Detect Named Entity of Tagged Words

In [17]:
```python
ne_tagged_words = nltk.ne_chunk(tagged_words)
print(ne_tagged_words)
```
```
(S
  Final/JJ
  exams/NN
  of/IN
  the/DT
  Fall/NN
  2023/CD
  semester/NN
  will/MD
  start/VB
  soon/RB
  ./.)
```

In [18]:
```python
nltk.download('punkt') # Sentence Tokenize
```
```
[nltk_data] Downloading package punkt to /home/jadi/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```
Out[18]:
```
True
```

In [19]:
```python
nltk.download('averaged_perceptron_tagger') # POS Tagging
```
```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jadi/nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```
Out[19]:
```
True
```

In [20]:
```python
nltk.download('maxent_ne_chunker')# Named Entity Chunking
```
```
[nltk_data] Downloading package maxent_ne_chunker to
[nltk_data]     /home/jadi/nltk_data...
[nltk_data]   Package maxent_ne_chunker is already up-to-date!
```
Out[20]:
```
True
```

In [21]:
```python
nltk.download('words')# Word Tokenize
```
```
[nltk_data] Downloading package words to /home/jadi/nltk_data...
[nltk_data]   Package words is already up-to-date!
```
Out[21]:
```
True
```

In [22]:
```python
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')
```

We are nearing the end of the semester DATE at Peshawar GPE . Final exams of the Fall 2023 semester will start soon.

In [23]:
```python
for ent in doc.ents:
    print(ent.text, ent.label_)
```
```
the end of the semester DATE
Peshawar GPE
```

Questions

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

In [24]:
```python
import nltk

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

for text in texts:
    sentences = nltk.sent_tokenize(text)
    print(sentences)

    for sentence in sentences:
        words = nltk.word_tokenize(sentence)
        print(words)

        tagged_words = nltk.pos_tag(words)
        print(tagged_words)

        ne_tagged_words = nltk.ne_chunk(tagged_words)
        print(ne_tagged_words)
```
```
['We are nearing the end of the semester at Peshawar.', 'Final exams of the Fall 2023 semester will start soon.']
['We', 'are', 'nearing', 'the', 'end', 'of', 'the', 'semester', 'at', 'Peshawar', '.']
[('We', 'PRP'), ('are', 'VBP'), ('nearing', 'VBG'), ('the', 'DT'), ('end', 'NN'), ('of', 'IN'), ('the', 'DT'), ('semester', 'NN'), ('at', 'IN'), ('Peshawar', 'NNP'), ('.', '.')]
(S
  We/PRP
  are/VBP
  nearing/VBG
  the/DT
  end/NN
  of/IN
  the/DT
  semester/NN
  at/IN
  (ORGANIZATION Peshawar/NNP)
  ./.)
['Final', 'exams', 'of', 'the', 'Fall', '2023', 'semester', 'will', 'start', 'soon', '.']
[('Final', 'JJ'), ('exams', 'NN'), ('of', 'IN'), ('the', 'DT'), ('Fall', 'NN'), ('2023', 'CD'), ('semester', 'NN'), ('will', 'MD'), ('start', 'VB'), ('soon', 'RB'), ('.', '.')]
(S
  Final/JJ
  exams/NN
  of/IN
  the/DT
  Fall/NN
  2023/CD
  semester/NN
  will/MD
  start/VB
  soon/RB
  ./.)
```

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

In [25]:
```python
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.')

# Visualize named entities
displacy.render(doc, style='ent')

# Print named entities and their labels
for ent in doc.ents:
    print(ent.text, ent.label_)
```

We are nearing the end of the semester DATE at Peshawar GPE . Final exams of the Fall 2023 semester will start soon.

```
the end of the semester DATE
Peshawar GPE
```

In [26]:
```python
import spacy
nlp = spacy.load('en_core_web_sm')
nlp.pipe_names
```
Out[26]:
```
['tok2vec', 'tagger', 'parser', 'attribute_ruler', 'lemmatizer', 'ner']
```

What is the default pipeline structure of spaCy?

1. **Tokenizer:** Splits text into individual tokens (words or subwords).
2. **Part-of-Speech Tagger:** Assigns parts of speech (e.g., noun, verb) to tokens.
3. **Dependency Parser:** Establishes syntactic relationships between words.
4. **Named Entity Recognizer (NER):** Identifies entities like persons, locations, dates, etc.
5. **Text Classification:** Assigns labels to the entire text (e.g., sentiment analysis).
6. **Entity Linker:** Connects named entities to knowledge base entries.
7. **Text Vectorization:** Generates vectors representing word and document semantics.
8. **Sentence Boundary Detector:** Identifies sentence boundaries.
9. **Morphological Analysis:** Provides information about word morphology (e.g., lemma).

In [27]:
```python
nlp = spacy.load('en_core_web_sm',disable=['parser'])
```

In [28]:
```python
## Adding a custom component
from spacy.language import Language
@Language.component("my_component")
def my_component(doc):
# Do something to the doc here
    return doc
```

In [29]:
```python
nlp.add_pipe("my_component")
```
Out[29]:
```
<function __main__.my_component(doc)>
```

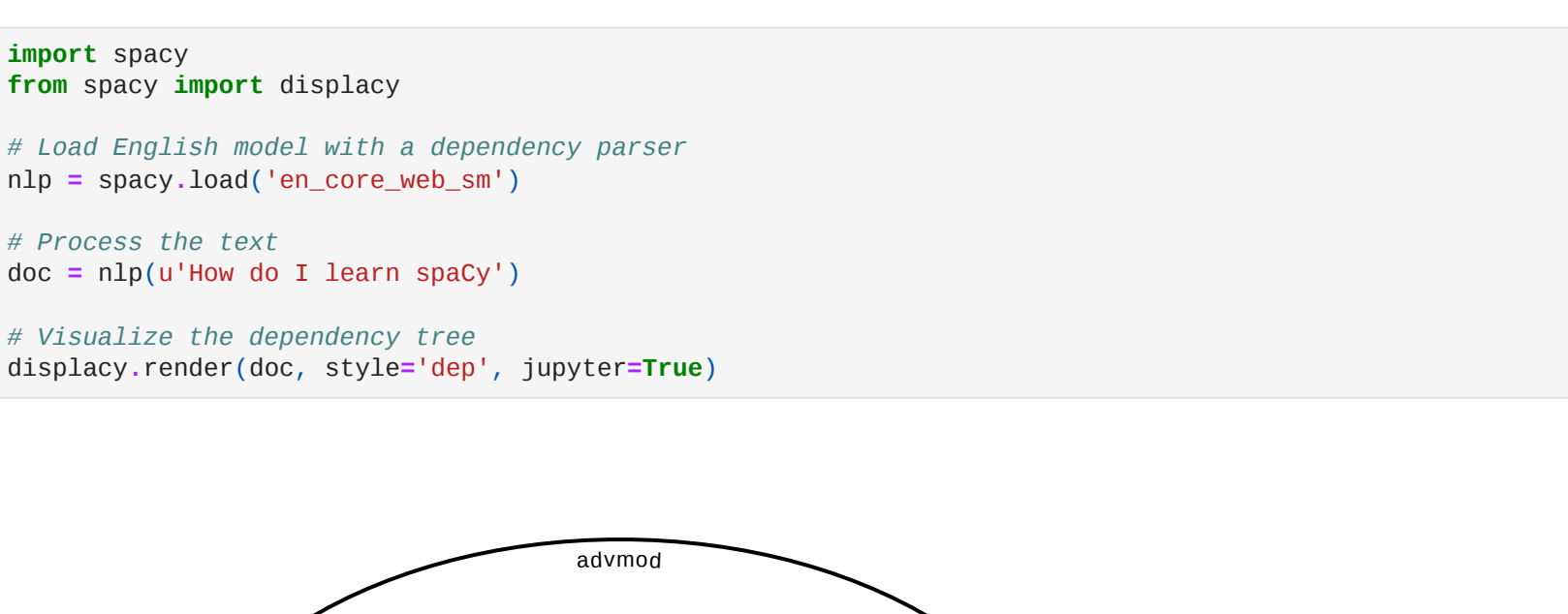
Task 4: Finding Patterns in Sentences

In [30]:
```python
import spacy
from spacy import displacy

# Load English model with a dependency parser
nlp = spacy.load('en_core_web_sm')

# Process the text
doc = nlp(u'I want to learn spaCy.')

# Visualize the dependency tree
displacy.render(doc, style='dep', jupyter=True)
```

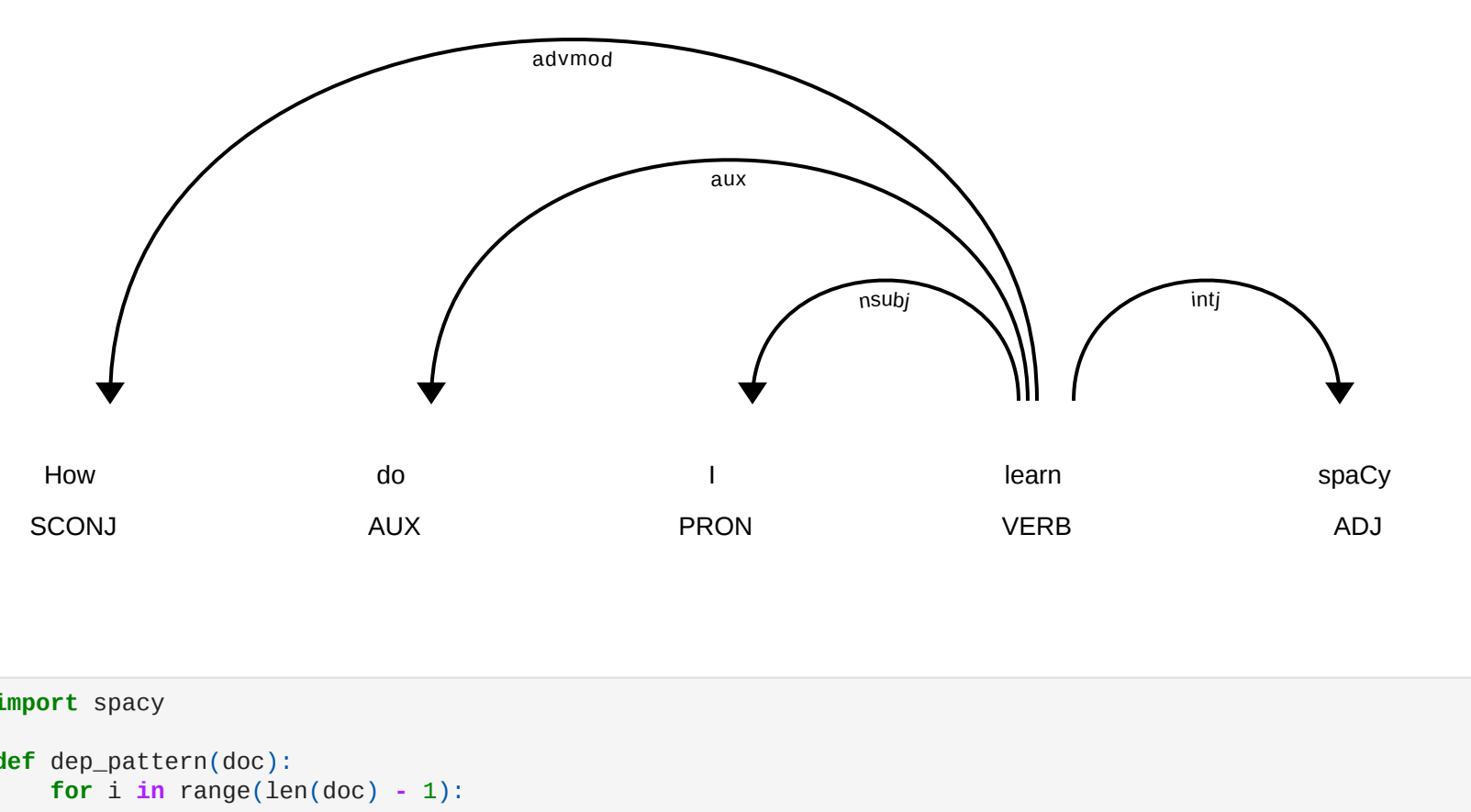


In [31]:
```python
import spacy
from spacy import displacy

# Load English model with a dependency parser
nlp = spacy.load('en_core_web_sm')

# Process the text
doc = nlp(u'How do I learn spaCy')

# Visualize the dependency tree
displacy.render(doc, style='dep', jupyter=True)
```



In [32]:
```python
import spacy

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

# Load English model with a dependency parser
nlp = spacy.load('en_core_web_sm')

# Process the text
doc = nlp(u'How do I learn spaCy.')

# Check for the dependency pattern
if dep_pattern(doc):
    print('Found')
else:
    print('Not found')
```
```
advmod
aux
nsubj
Found
```

In [33]:
```python
import spacy

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

# Load English model with a dependency parser
nlp = spacy.load('en_core_web_sm')

# Process the text
doc = nlp(u'I want to learn spacy.')

# Check for the dependency pattern
if dep_pattern(doc):
    print('Found')
else:
    print('Not found')
```
```
nsubj
ROOT
aux
xcomp
dobj
Not found
```