**Getting Started**

Let's get started and try out spaCy! In this exercise, you'll be able to try out some of the 45+ [available languages](https://spacy.io/usage/models#languages).

*This course introduces a lot of new concepts, so if you ever need a quick refresher, download the* [*spaCy Cheat Sheet*](http://datacamp-community-prod.s3.amazonaws.com/29aa28bf-570a-4965-8f54-d6a541ae4e06) *and keep it handy!*

**Instructions 1/3**

35 XP

* [1](javascript:void(0))
  + Import the English class from spacy.lang.en and create the nlp object.
  + Create a doc and print its text.

# Import the English language class

from spacy.lang.en import English

# Create the nlp object

nlp = English()

# Process a text

doc = nlp("This is a sentence.")

# Print the document text

print(doc.text)

* Import the German class from spacy.lang.de and create the nlp object.
* Create a doc and print its text.

# Import the German language class

from spacy.lang.de import German

# Create the nlp object

nlp = German()

# Process a text (this is German for: "Kind regards!")

doc = nlp("Liebe Grüße!")

# Print the document text

print(doc.text)

* Import the Spanish class from spacy.lang.es and create the nlp object.
* Create a doc and print its text.

# Import the Spanish language class

from spacy.lang.es import Spanish

# Create the nlp object

nlp = Spanish()

# Process a text (this is Spanish for: "How are you?")

doc = nlp("¿Cómo estás?")

# Print the document text

print(doc.text)

<script.py> output:

This is a sentence.

<script.py> output:

Liebe Grüße!

<script.py> output:

¿Cómo estás?

**Documents, spans and tokens**

When you call nlp on a string, spaCy first tokenizes the text and creates a document object. In this exercise, you'll learn more about the Doc, as well as its views Token and Span.

**Instructions 1/2**

50 XP

* [1](javascript:void(0))
* [2](javascript:void(0))
* Import the English language class and create the nlp object.
* Process the text and instantiate a Doc object in the variable doc.
* Select the first token of the Doc and print its text.

# Import the English language class and create the nlp object

from spacy.lang.en import English

nlp = English()

# Process the text

doc = nlp("I like tree kangaroos and narwhals.")

# Select the first token

first\_token = doc[0]

# Print the first token's text

print(first\_token.text)

<script.py> output:

I

**Instructions 2/2**

50 XP

* [2](javascript:void(0))
* Create a slice of the Doc for the tokens "tree kangaroos" and "tree kangaroos and narwhals".

# Import the English language class and create the nlp object

from spacy.lang.en import English

nlp = English()

# Process the text

doc = nlp("I like tree kangaroos and narwhals.")

# A slice of the Doc for "tree kangaroos"

tree\_kangaroos = doc[2:4]

print(tree\_kangaroos.text)

# A slice of the Doc for "tree kangaroos and narwhals" (without the ".")

tree\_kangaroos\_and\_narwhals = doc[2:-1]

print(tree\_kangaroos\_and\_narwhals.text)

<script.py> output:

tree kangaroos

tree kangaroos and narwhals

**Lexical attributes**

In this example, you'll use spaCy's Doc and Token objects, and lexical attributes to find percentages in a text. You'll be looking for two subsequent tokens: a number and a percent sign. The English nlp object has already been created.

**Instructions**

100 XP

* Use the like\_num token attribute to check whether a token in the doc resembles a number.
* Get the token *following* the current token in the document. The index of the next token in the doc is token.i + 1.
* Check whether the next token's text attribute is a percent sign "%".

# Process the text

doc = nlp("In 1990, more than 60% of people in East Asia were in extreme poverty. Now less than 4% are.")

# Iterate over the tokens in the doc

for token in doc:

# Check if the token resembles a number

if token.like\_num:

# Get the next token in the document

next\_token = doc[token.i + 1]

# Check if the next token's text equals '%'

if next\_token.text == '%':

print('Percentage found:', token.text)

<script.py> output:

Percentage found: 60

Percentage found: 4

Well done! As you can see, you can do a lot of very powerful analyses using the tokens and their attributes.

**Model packages**

What's **not** included in a model package that you can load into spaCy?

**Points** :

All models include a meta.json that defines the language to initialize, the pipeline component names to load as well as general meta information like the model name, version, license, data sources, author and accuracy figures (if available).

To predict linguistic annotations like part-of-speech tags, dependency labels or named entities, models include binary weights.

Model packages include a strings.json that stores the entries in the model's vocabulary and the mapping to hashes. This allows spaCy to only communicate in hashes and look up the corresponding string if needed.

**Answer the question**

50 XP

**Possible Answers**

* 

A meta file including the language, pipeline and license.

press1

* 

Binary weights to make statistical predictions.

press2

* 

The labelled data that the model was trained on. **(A)**

press3

* 

Strings of the model's vocabulary and their hashes.

press4

[That's correct! Statistical models allow you to generalize based on a set of training examples. Once they're trained, they use binary weights to make predictions. That's why it's not necessary to ship them with their training data.]

# Loading models

Let's start by loading a model. spacy is already imported.

##### Instructions 1/2

50 XP

* [1](javascript:void(0))
  + Use spacy.load to load the small English model 'en\_core\_web\_sm'.
  + Process the text and print the document text.

# Load the 'en\_core\_web\_sm' model – spaCy is already imported

nlp = spacy.load('en\_core\_web\_sm')

text = "It’s official: Apple is the first U.S. public company to reach a $1 trillion market value"

# Process the text

doc = nlp(text)

# Print the document text

print(doc.text)

<script.py> output:

It’s official: Apple is the first U.S. public company to reach a $1 trillion market value

* Use spacy.load to load the small German model 'de\_core\_news\_sm'.
* Process the text and print the document text.

# Load the 'de\_core\_news\_sm' model – spaCy is already imported

nlp = spacy.load('de\_core\_news\_sm')

text = "Als erstes Unternehmen der Börsengeschichte hat Apple einen Marktwert von einer Billion US-Dollar erreicht"

# Process the text

doc = nlp(text)

# Print the document text

print(doc.text)

<script.py> output:

Als erstes Unternehmen der Börsengeschichte hat Apple einen Marktwert von einer Billion US-Dollar erreicht

Well done! Now that you've practiced loading models, let's look at some of their predictions.

# Predicting linguistic annotations

You'll now get to try one of spaCy's pre-trained model packages and see its predictions in action. Feel free to try it out on your own text! The small English model is already available as the variable nlp.

To find out what a tag or label means, you can call spacy.explain in the IPython shell. For example: spacy.explain('PROPN') or spacy.explain('GPE').

##### Instructions 1/2

50 XP

* [1](javascript:void(0))
* [2](javascript:void(0))
* Process the text with the nlp object and create a doc.
* For each token, print the token text, the token's .pos\_ (part-of-speech tag) and the token's .dep\_ (dependency label).

text = "It’s official: Apple is the first U.S. public company to reach a $1 trillion market value"

# Process the text

doc = nlp(text)

for token in doc:

# Get the token text, part-of-speech tag and dependency label

token\_text = token.text

token\_pos = token.pos\_

token\_dep = token.dep\_

# This is for formatting only

print('{:<12}{:<10}{:<10}'.format(token\_text, token\_pos, token\_dep))

<script.py> output:

It PRON nsubj

’s PROPN ROOT

official NOUN acomp

: PUNCT punct

Apple PROPN nsubj

is VERB ROOT

the DET det

first ADJ amod

U.S. PROPN nmod

public ADJ amod

company NOUN attr

to PART aux

reach VERB relcl

a DET det

$ SYM quantmod

1 NUM compound

trillion NUM nummod

market NOUN compound

value NOUN dobj

##### Instructions 2/2

50 XP

* [2](javascript:void(0))
* Process the text and create a doc object.
* Iterate over the doc.ents and print the entity text and label\_ attribute.

text = "It’s official: Apple is the first U.S. public company to reach a $1 trillion market value"

# Process the text

doc = nlp(text)

# Iterate over the predicted entities

for ent in doc.ents:

# print the entity text and its label

print(ent.text, ent.label\_)

<script.py> output:

Apple ORG

first ORDINAL

U.S. GPE

$1 trillion MONEY

Great work! So far, the model has been correct every single time. In the next exercise, you'll see what happens if the model is wrong, and how to adjust it.

# Predicting named entities in context

Models are statistical and not always right. Whether their predictions are correct depends on the training data and the text you're processing. Let's take a look at an example. The small English model is available as the variable nlp.

##### Instructions 1/2

50 XP

* [1](javascript:void(0))
* [2](javascript:void(0))
* Process the text with the nlp object.
* Iterate over the entities with the iterator ent and print the entity text and label.

text = "New iPhone X release date leaked as Apple reveals pre-orders by mistake"

# Process the text

doc = nlp(text)

# Iterate over the entities

for ent in doc.ents:

# print the entity text and label

print(ent.text , ent.label\_)

<script.py> output:

Apple ORG

##### Instructions 2/2

50 XP

* [2](javascript:void(0))
* Looks like the model didn't predict "iPhone X". Create a span for those tokens manually.

text = "New iPhone X release date leaked as Apple reveals pre-orders by mistake"

from spacy.tokens import Span

# Process the text

doc = nlp(text)

# Iterate over the entities

for ent in doc.ents:

# print the entity text and label

print(ent.text, ent.label\_)

# Get the span for "iPhone X"

iphone\_x = Span(doc , 1 , 3 , label= "iPhone X")

# Print the span text

print('Missing entity:', iphone\_x.text)

<script.py> output:

Apple ORG

Missing entity: iPhone X

Perfect! Of course, you don't always have to do this manually. In the next video, you'll learn about spaCy's rule-based matcher, which can help you find certain words and phrases in text.

# Using the Matcher

Let's try spaCy's rule-based Matcher. You'll be using the example from the previous exercise and write a pattern that can match the phrase "iPhone X" in the text. The nlp object and a processed doc are already available.

##### Instructions 1/3

35 XP

* [1](javascript:void(0))
* [2](javascript:void(0))
* [3](javascript:void(0))
* Import the Matcher from spacy.matcher.
* Initialize it with the nlp object's shared vocab.

# Import the Matcher

from spacy.matcher import Matcher

# Initialize the Matcher with the shared vocabulary

matcher = Matcher(nlp.vocab)

##### Instructions 2/3

35 XP

* [2](javascript:void(0))
* [3](javascript:void(0))
* Create a pattern that matches the 'TEXT' values of two tokens: "iPhone" and "X".
* Use the matcher.add method to add the pattern to the matcher.

# Import the Matcher

from spacy.matcher import Matcher

# Initialize the Matcher with the shared vocabulary

matcher = Matcher(nlp.vocab)

# Create a pattern matching two tokens: "iPhone" and "X"

pattern = [{'TEXT' : 'iPhone'} ,

{'TEXT' : 'X'}]

# Add the pattern to the matcher

matcher.add('IPHONE\_X\_PATTERN', None, pattern)

##### Instructions 3/3

30 XP

* [3](javascript:void(0))
* Call the matcher on the doc and store the result in the variable matches.
* Iterate over the matches and get the matched span from the start to the end index.

# Import the Matcher and initialize it with the shared vocabulary

from spacy.matcher import Matcher

matcher = Matcher(nlp.vocab)

# Create a pattern matching two tokens: "iPhone" and "X"

pattern = [{'TEXT': 'iPhone'}, {'TEXT': 'X'}]

# Add the pattern to the matcher

matcher.add('IPHONE\_X\_PATTERN', None, pattern)

# Use the matcher on the doc

matches = matcher(doc)

print('Matches:', [doc[start:end].text for match\_id, start, end in matches])

New iPhone X release date leaked as Apple reveals pre-orders by mistake

New iPhone X release date leaked as Apple reveals pre-orders by mistake

New iPhone X release date leaked as Apple reveals pre-orders by mistake

<script.py> output:

Matches: ['iPhone X']

Well done! You successfully found one match: the tokens at doc[1:3] describing the span for "iPhone X".

# Writing match patterns

In this exercise, you'll practice writing more complex match patterns using different token attributes and operators. A matcher is already initialized and available as the variable matcher.

##### Instructions 1/3

35 XP

* [1](javascript:void(0))
  + Write **one** pattern that only matches mentions of the full iOS versions: "iOS 7", "iOS 11" and "iOS 10".

doc = nlp("After making the iOS update you won't notice a radical system-wide redesign: nothing like the aesthetic upheaval we got with iOS 7. Most of iOS 11's furniture remains the same as in iOS 10. But you will discover some tweaks once you delve a little deeper.")

# Write a pattern for full iOS versions ("iOS 7", "iOS 11", "iOS 10")

pattern = [{'TEXT': 'iOS'}, {'IS\_DIGIT': True}]

# Add the pattern to the matcher and apply the matcher to the doc

matcher.add('IOS\_VERSION\_PATTERN', None, pattern)

matches = matcher(doc)

print('Total matches found:', len(matches))

# Iterate over the matches and print the span text

for match\_id, start, end in matches:

print('Match found:', doc[start:end].text)

<script.py> output:

Total matches found: 3

Match found: iOS 7

Match found: iOS 11

Match found: iOS 10

Write **one** pattern that only matches forms of "download" (tokens with the lemma "download"), followed by a token with the part-of-speech tag 'PROPN' (proper noun).

doc = nlp("i downloaded Fortnite on my laptop and can't open the game at all. Help? so when I was downloading Minecraft, I got the Windows version where it is the '.zip' folder and I used the default program to unpack it... do I also need to download Winzip?")

# Write a pattern that matches a form of "download" plus proper noun

pattern = [{'LEMMA': 'download'}, {'POS': 'PROPN'}]

# Add the pattern to the matcher and apply the matcher to the doc

matcher.add('DOWNLOAD\_THINGS\_PATTERN', None, pattern)

matches = matcher(doc)

print('Total matches found:', len(matches))

# Iterate over the matches and print the span text

for match\_id, start, end in matches:

print('Match found:', doc[start:end].text)

<script.py> output:

Total matches found: 3

Match found: downloaded Fortnite

Match found: downloading Minecraft

Match found: download Winzip

Write **one** pattern that matches adjectives ('ADJ') followed by one or two 'NOUN's (one noun and one optional noun).

doc = nlp("Features of the app include a beautiful design, smart search, automatic labels and optional voice responses.")

# Write a pattern for adjective plus one or two nouns

pattern = [{'POS': 'ADJ'}, {'POS': 'NOUN'}, {'POS': 'NOUN', 'OP': '?'}]

# Add the pattern to the matcher and apply the matcher to the doc

matcher.add('ADJ\_NOUN\_PATTERN', None, pattern)

matches = matcher(doc)

print('Total matches found:', len(matches))

# Iterate over the matches and print the span text

for match\_id, start, end in matches:

print('Match found:', doc[start:end].text)

<script.py> output:

Total matches found: 4

Match found: beautiful design

Match found: smart search

Match found: automatic labels

Match found: optional voice responses

Great work – those were some pretty complex patterns! Let's move on to the next chapter and take a look at how to use spaCy for more advanced text analysis.