spaCy

An overview of spaCy, the industrial-strength NLP library that turns raw text into tokens, entities, and parses at scale.

Load a pipeline, get a Doc

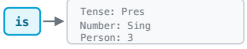
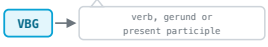
A trained pipeline maps text to an annotated Doc.

PURPOSE	COMMAND	RESULT
Load a trained pipeline.	`nlp = spacy.load("en_core_web_sm")`	
Process text into a Doc.	`doc = nlp(` `  "Apple is looking at buying U.K. ...")`	
A Doc is a sequence of tokens.	`len(doc), doc[0], doc[1:3]`	
Split into sentences.	`list(doc.sents)`	
Get back the original text.	`doc.text, token.text_with_ws`	
Start blank (no model).	`nlp = spacy.blank("en")`	

spacy.load(name) builds the nlp pipeline; nlp(text) returns a Doc you index like a sequence (doc[i] Token, doc[i:j] Span)

Tokens & Linguistic Features

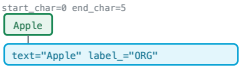
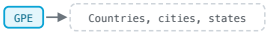
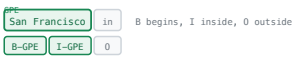
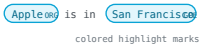
Each Token carries POS, lemma, dependency, and morphology.

PURPOSE	COMMAND	RESULT
Part-of-speech tag.	`token.pos_, token.tag_`	
Lemma (dictionary form).	`token.lemma_`	
Token shape & flags.	`token.is_stop, token.is_alpha` `token.like_num`	
Morphological features.	`token.morph.to_dict()`	
Explain a tag in words.	`spacy.explain("VBG")`	
Iterate tokens & their heads.	`[(t.text, t.pos_, t.head.text) for t in doc]`	

String annotations end in an underscore (pos_, lemma_, dep_); the bare name returns an integer hash, not text

Named Entities (doc.ents)

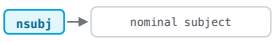
Real-world spans the model labels (ORG, GPE, MONEY, ...).

PURPOSE	COMMAND	RESULT
List the entities.	`doc.ents`	
Read one entity's fields.	`ent.text,ent.label_, ent.start_char`	
Decode a label.	`spacy.explain("GPE")`	
Per-token entity tags (BIO).	`token.ent_iob_,token.ent_type_`	
Visualize entities (HTML).	`displacy.render(doc, style="ent")`	

doc.ents is a tuple of Span objects; each ent has .text, .label_, and char offsets; per-token the same info is BIO tags

The Dependency Parse

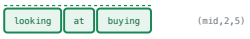
Every token has one head; arcs are typed syntactic relations.

PURPOSE	COMMAND	RESULT
Find the sentence root.	`[t for t in doc if t.dep_== "ROOT"]`	
Read a token's relation.	`token.dep_, token.head`	
Walk a token's children.	`list(token.children)`	
Extract base noun phrases.	`doc.noun_chunks`	
Decode a relation label.	`spacy.explain("nsubj")`	
Visualize the parse (SVG).	`displacy.render(doc, style="dep")`	

The parse is a tree: each token points to one head via a typed dep_ relation; the lone self-headed token is the ROOT

Spans & the Matcher

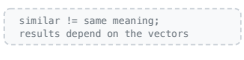
Slice the Doc into spans; find token patterns with the Matcher.

PURPOSE	COMMAND	RESULT
Slice tokens into a Span.	`span = doc[2:4]`	
Build a span from char offsets.	`doc.char_span(0, 5, label="ORG")`	
Token-attribute patterns.	`m = Matcher(nlp.vocab)` `m.add("LOOK", [[{"LOWER": "looking"}, …]])`	
Run the matcher.	`for mid, start, end in m(doc):`	
Match exact phrases fast.	`pm = PhraseMatcher(nlp.vocab)` `pm.add("CO", [nlp.make_doc("Apple")])`	
Store named span groups.	`doc.spans["sc"] = [doc[0:1], doc[5:6]]`	

a Span is a slice of a Doc; Matcher finds token-attribute patterns, PhraseMatcher matches exact phrases fast

Similarity & Word Vectors

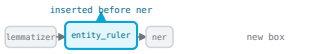
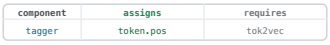
Compare meaning with vectors. Use md/lg, not sm, for real vectors.

PURPOSE	COMMAND	RESULT
Read a token's vector.	`token.vector, token.vector.shape`	
Does it have real vectors?	`token.has_vector, token.is_oov`	
Compare two docs or tokens.	`doc1.similarity(doc2)` `token1.similarity(token2)`	
Load a vectors pipeline.	`nlp = spacy.load("en_core_web_md")`	
Small models warn, not error.	`en_core_web_sm tensors, no vectors`	
Manage expectations.	`# similarity reflects training data`	

.vector and .similarity() need real word vectors: load en_core_web_md or _lg, not _sm (tensors only)

Pipeline components

nlp is a sequence of named components you can inspect and edit.

PURPOSE	COMMAND	RESULT
List the components.	`nlp.pipe_names`	
Add a built-in component.	`ruler = nlp.add_pipe( "entity_ruler", before="ner")`	
Add rule-based entity patterns.	`ruler.add_patterns([{"label": "ORG", "pattern": "spaCy"}])`	
Run without some components.	`with nlp.select_pipes(disable= ["ner"]): nlp(text)`	
Register a custom component.	`@Language.component("my_comp") nlp.add_pipe("my_comp", last=True)`	
Inspect what each needs / sets.	`nlp.analyze_pipes(pretty=True)`	

nlp is an ordered list of named components: nlp.pipe_names lists them, add_pipe inserts, select_pipes disables, analyze_pipes explains the order

Process at Scale with nlp.pipe

Stream many texts efficiently in batches.

PURPOSE	COMMAND	RESULT
Stream many texts.	`docs = nlp.pipe(texts)`	
Tune the batch size.	`nlp.pipe(texts, batch_size=50)`	
Run components in parallel.	`nlp.pipe(texts, n_process=4)`	
Keep metadata with each text.	`nlp.pipe(data, as_tuples=True)`	
Skip unused components.	`nlp.pipe(texts, disable=["parser","ner"])`	
Process, then collect results.	`for doc in nlp.pipe(texts): doc.ents`	

nlp.pipe(texts) batches internally for a big speedup; tune batch_size, n_process, as_tuples, and disable instead of looping nlp(text)