Matcher CLASS

SOURCE

Match sequences of tokens, based on pattern rules

The `Matcher` lets you find words and phrases using rules describing their token attributes. Rules can refer to token annotations (like the text or part-of-speech tags), as well as lexical attributes like `Token.is_punct`. Applying the matcher to a `Doc` gives you access to the matched tokens in context. For in-depth examples and workflows for combining rules and statistical models, see the usage guide on rule-based matching.

Pattern format

EXAMPLE

```
[
  {"LOWER": "i"},
  {"LEMMA": {"IN": ["like", "love"]}},
  {"POS": "NOUN", "OP": "+"}
]
```

A pattern added to the `Matcher` consists of a list of dictionaries. Each dictionary describes **one token** and its attributes. The available token pattern keys correspond to a number of `Token` attributes. The supported attributes for rule-based matching are:

`ORTH`	The exact verbatim text of a token. **TYPE:** `str`
`TEXT`	The exact verbatim text of a token. **TYPE:** `str`
`LOWER`	The lowercase form of the token text. **TYPE:** `str`
`LENGTH`	The length of the token text. **TYPE:** `int`
`IS_ALPHA`, `IS_ASCII`, `IS_DIGIT`	Token text consists of alphabetic characters, ASCII characters, digits. **TYPE:** `bool`
`IS_LOWER`, `IS_UPPER`, `IS_TITLE`	Token text is in lowercase, uppercase, titlecase. **TYPE:** `bool`
`IS_PUNCT`, `IS_SPACE`, `IS_STOP`	Token is punctuation, whitespace, stop word. **TYPE:** `bool`
`IS_SENT_START`	Token is start of sentence. **TYPE:** `bool`
`LIKE_NUM`, `LIKE_URL`, `LIKE_EMAIL`	Token text resembles a number, URL, email. **TYPE:** `bool`
`SPACY`	Token has a trailing space. **TYPE:** `bool`
`POS`, `TAG`, `MORPH`, `DEP`, `LEMMA`, `SHAPE`	The token's simple and extended part-of-speech tag, morphological analysis, dependency label, lemma, shape. **TYPE:** `str`
`ENT_TYPE`	The token's entity label. **TYPE:** `str`
`_`	Properties in custom extension attributes.

Operators and quantifiers define **how often** a token pattern should be matched:

EXAMPLE

```
[
  {"POS": "ADJ", "OP": "*"},
  {"POS": "NOUN", "OP": "+"}
]
```

OP	DESCRIPTION
`!`	Negate the pattern, by requiring it to match exactly 0 times.
`?`	Make the pattern optional, by allowing it to match 0 or 1 times.
`+`	Require the pattern to match 1 or more times.
`*`	Allow the pattern to match 0 or more times.

Token patterns can also map to a **dictionary of properties** instead of a single value to indicate whether the expected value is a member of a list or how it compares to another value.

EXAMPLE

```
[
  {"LEMMA": {"IN": ["like", "love", "enjoy"]}},
  {"POS": "PROPN", "LENGTH": {">=": 10}},
]
```

`IN`	Attribute value is member of a list. **TYPE:** `Any`
`NOT_IN`	Attribute value is *not* member of a list. **TYPE:** `Any`
`ISSUBSET`	Attribute values (for `MORPH`) are a subset of a list. **TYPE:** `Any`
`ISSUPERSET`	Attribute values (for `MORPH`) are a superset of a list. **TYPE:** `Any`
`==` , `>=` , `<=` , `>` , `<`	Attribute value is equal, greater or equal, smaller or equal, greater or smaller. **TYPE:** `Union[int, float]`

Matcher.__init__ METHOD

Create the rule-based `Matcher` . If `validate=True` is set, all patterns added to the matcher will be validated against a JSON schema and a `MatchPatternError` is raised if problems are found. Those can include incorrect types (e.g. a string where an integer is expected) or unexpected property names.

EXAMPLE

```
from spacy.matcher import Matcher
matcher = Matcher(nlp.vocab)
```

`vocab`	The vocabulary object, which must be shared with the documents the matcher will operate on. **TYPE:** `Vocab`
`validate`	Validate all patterns added to this matcher. **TYPE:** `bool`

Matcher.__call__ METHOD

Find all token sequences matching the supplied patterns on the `Doc` or `Span` .

EXAMPLE

```python
from spacy.matcher import Matcher

matcher = Matcher(nlp.vocab)
pattern = [{"LOWER": "hello"}, {"LOWER": "world"}]
matcher.add("HelloWorld", [pattern])
doc = nlp("hello world!")
matches = matcher(doc)
```

`doclike`	The `Doc` or `Span` to match over. **TYPE:** `Union[Doc, Span]`
KEYWORD-ONLY	
`as_spans` **V3.0**	Instead of tuples, return a list of `Span` ≣ objects of the matches, with the `match_id` assigned as the span label. Defaults to `False`. **TYPE:** `bool`
`allow_missing` **V3.0**	Whether to skip checks for missing annotation for attributes included in patterns. Defaults to `False`. **TYPE:** `bool`
RETURNS	A list of `(match_id, start, end)` tuples, describing the matches. A match tuple describes a span `doc[start:end]`. The `match_id` is the ID of the added match pattern. If `as_spans` is set to `True`, a list of `Span` objects is returned instead. **TYPE:** `Union[List[Tuple[int, int, int]], List[Span]]`

Matcher.__len__ METHOD

Get the number of rules added to the matcher. Note that this only returns the number of rules (identical with the number of IDs), not the number of individual patterns.

EXAMPLE

```
matcher = Matcher(nlp.vocab)
assert len(matcher) == 0
matcher.add("Rule", [[{"ORTH": "test"}]])
assert len(matcher) == 1
```

RETURNS	The number of rules. TYPE: `int`

Matcher.__contains__ METHOD

Check whether the matcher contains rules for a match ID.

EXAMPLE

```
matcher = Matcher(nlp.vocab)
assert "Rule" not in matcher
matcher.add("Rule", [[{'ORTH': 'test'}]])
assert "Rule" in matcher
```

NAME	DESCRIPTION
`key`	The match ID. TYPE: `str`
RETURNS	Whether the matcher contains rules for this match ID. TYPE: `bool`

Matcher.add METHOD

`i` and `matches`. If a pattern already exists for the given ID, the patterns will be extended. An `on_match` callback will be overwritten.

EXAMPLE

```
def on_match(matcher, doc, id, matches):
    print('Matched!', matches)

matcher = Matcher(nlp.vocab)
patterns = [
   [{"LOWER": "hello"}, {"LOWER": "world"}],
   [{"ORTH": "Google"}, {"ORTH": "Maps"}]
]
matcher.add("TEST_PATTERNS", patterns)
doc = nlp("HELLO WORLD on Google Maps.")
matches = matcher(doc)
```

Changed in v3.0

As of spaCy v3.0, `Matcher.add` takes a list of patterns as the second argument (instead of a variable number of arguments). The `on_match` callback becomes an optional keyword argument.

```
patterns = [[{"TEXT": "Google"}, {"TEXT": "Now"}], [{"TEXT": "GoogleNow"}]]
- matcher.add("GoogleNow", on_match, *patterns)
+ matcher.add("GoogleNow", patterns, on_match=on_match)
```

`match_id`	An ID for the thing you're matching. TYPE: `str`
`patterns`	Match pattern. A pattern consists of a list of dicts, where each dict describes a token. TYPE: `List[List[Dict[str, Any]]]`
KEYWORD-ONLY	
`on_match`	Callback function to act on matches. Takes the arguments `matcher` , `doc` , `i` and `matches` . TYPE: `Optional[Callable[[Matcher, Doc, int, List[tuple], Any]]`
`greedy` V3.0	Optional filter for greedy matches. Can either be `"FIRST"` or `"LONGEST"` . TYPE: `Optional[str]`

Matcher.remove METHOD

Remove a rule from the matcher. A `KeyError` is raised if the match ID does not exist.

EXAMPLE

```
matcher.add("Rule", [[{"ORTH": "test"}]])
assert "Rule" in matcher
matcher.remove("Rule")
assert "Rule" not in matcher
```

key	The ID of the match rule.
	TYPE: str

Matcher.get METHOD

Retrieve the pattern stored for a key. Returns the rule as an `(on_match, patterns)` tuple containing the callback and available patterns.

EXAMPLE

```
matcher.add("Rule", [[{"ORTH": "test"}]])
on_match, patterns = matcher.get("Rule")
```

NAME	DESCRIPTION
key	The ID of the match rule.
	TYPE: str
RETURNS	The rule, as an `(on_match, patterns)` tuple.
	TYPE: Tuple[Optional[Callable], List[List[dict]]]

SUGGEST EDITS

spaCy API

SPACY	COMMUNITY	CONNECT
Usage	Universe	Twitter
Models	GitHub Discussions	GitHub
API Reference	Issue Tracker	YouTube
Online Course	Stack Overflow	Blog

STAY IN THE LOOP!

Receive updates about new releases, tutorials and more.

Your email SIGN UP