Language CLASS

SOURCE

A text-processing pipeline

Usually you'll load this once per process as `nlp` and pass the instance around your application. The `Language` class is created when you call `spacy.load` and contains the shared vocabulary and language data, optional binary weights, e.g. provided by a trained pipeline, and the processing pipeline containing components like the tagger or parser that are called on a document in order. You can also add your own processing pipeline components that take a `Doc` object, modify it and return it.

Language.__init__ METHOD

Initialize a `Language` object. Note that the `meta` is only used for meta information in `Language.meta` and not to configure the `nlp` object or to override the config. To initialize from a config, use `Language.from_config` instead.

NAME	DESCRIPTION
`vocab`	A `Vocab` object. If `True`, a vocab is created using the default language data settings. **TYPE:** `Vocab`
KEYWORD-ONLY `max_length`	Maximum number of characters allowed in a single text. Defaults to `10 ** 6`. **TYPE:** `int`
`meta`	Meta data overrides. **TYPE:** `Dict[str, Any]`
`create_tokenizer`	Optional function that receives the `nlp` object and returns a tokenizer. **TYPE:** `Callable[[Language], Callable[[str],Doc]]`
`batch_size`	Default batch size for `pipe` and `evaluate`. Defaults to `1000`. **TYPE:** `int`

Language.from_config CLASSMETHOD V3.0

Create a `Language` object from a loaded config. Will set up the tokenizer and language data, add pipeline components based on the pipeline and add pipeline components based on the definitions specified in the config. If no config is provided, the default config of the given language is used. This is also how spaCy loads a model under the hood based on its `config.cfg`.

NAME	DESCRIPTION
`config`	The loaded config. TYPE: `Union[Dict[str, Any],Config]`
KEYWORD-ONLY `vocab`	A `Vocab` object. If `True` , a vocab is created using the default language data settings. TYPE: `Vocab`
`disable`	Name(s) of pipeline component(s) to disable. Disabled pipes will be loaded but they won't be run unless you explicitly enable them by calling nlp.enable_pipe . Is merged with the config entry `nlp.disabled` . TYPE: `Union[str, Iterable[str]]`
`enable` V3.4	Name(s) of pipeline component(s) to enable. All other pipes will be disabled, but can be enabled again using nlp.enable_pipe . TYPE: `Union[str, Iterable[str]]`
`exclude`	Name(s) of pipeline component(s) to exclude. Excluded components won't be loaded. TYPE: `Union[str, Iterable[str]]`
`meta`	Meta data overrides. TYPE: `Dict[str, Any]`
`auto_fill`	Whether to automatically fill in missing values in the config, based on defaults and function argument annotations. Defaults to `True` . TYPE: `bool`
`validate`	Whether to validate the component config and arguments against the types expected by the factory. Defaults to `True` . TYPE: `bool`
RETURNS	The initialized object. TYPE: `Language`

Language.component CLASSMETHOD V3.0

Register a custom pipeline component under a given name. This allows initializing the component by name using `Language.add_pipe` ≣ and referring to it in config files. This classmethod and decorator is intended for **simple stateless functions** that take a `Doc` and return it. For more complex stateful components that allow settings and need access to the shared `nlp` object, use the `Language.factory` ≣ decorator. For more details and examples, see the usage documentation.

NAME	DESCRIPTION
`name`	The name of the component factory. TYPE: `str`
KEYWORD-ONLY `assigns`	`Doc` or `Token` attributes assigned by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. TYPE: `Iterable[str]`
`requires`	`Doc` or `Token` attributes required by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. TYPE: `Iterable[str]`
`retokenizes`	Whether the component changes tokenization. Used for pipe analysis. TYPE: `bool`
`func`	Optional function if not used as a decorator. TYPE: `Optional[Callable[[Doc],Doc]]`

Language.factory CLASSMETHOD

Register a custom pipeline component factory under a given name. This allows initializing the component by name using `Language.add_pipe` ≣ and referring to it in config files. The registered factory function needs to take at least two **named arguments** which spaCy fills in automatically: `nlp` for the current `nlp` object and `name` for the component instance name. This can be useful to distinguish multiple instances of the same component and allows trainable components to add custom losses using the component instance name. The `default_config` defines the default values of the remaining factory arguments. It's merged into the `nlp.config` ≣ . For more details and examples, see the usage documentation.

NAME	DESCRIPTION
`name`	The name of the component factory. **TYPE:** `str`
KEYWORD-ONLY `default_config`	The default config, describing the default values of the factory arguments. **TYPE:** `Dict[str, Any]`
`assigns`	`Doc` or `Token` attributes assigned by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. **TYPE:** `Iterable[str]`
`requires`	`Doc` or `Token` attributes required by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. **TYPE:** `Iterable[str]`
`retokenizes`	Whether the component changes tokenization. Used for pipe analysis. **TYPE:** `bool`
`default_score_weights`	The scores to report during training, and their default weight towards the final score used to select the best model. Weights should sum to `1.0` per component and will be combined and normalized for the whole pipeline. If a weight is set to `None`, the score will not be logged or weighted. **TYPE:** `Dict[str, Optional[float]]`
`func`	Optional function if not used as a decorator. **TYPE:** `Optional[Callable[[…], Callable[[Doc],Doc]]]`

Language.__call__ METHOD

Apply the pipeline to some text. The text can span multiple sentences, and can contain arbitrary whitespace. Alignment into the original string is preserved.

Instead of text, a `Doc` can be passed as input, in which case tokenization is skipped, but the rest of the pipeline is run.

NAME	DESCRIPTION
`text`	The text to be processed, or a Doc. TYPE: `Union[str,Doc]`
KEYWORD-ONLY `disable`	Names of pipeline components to disable. TYPE: `List[str]`
`component_cfg`	Optional dictionary of keyword arguments for components, keyed by component names. Defaults to `None`. TYPE: `Optional[Dict[str, Dict[str, Any]]]`
RETURNS	A container for accessing the annotations. TYPE: `Doc`

Language.pipe METHOD

Process texts as a stream, and yield `Doc` objects in order. This is usually more efficient than processing texts one-by-one.

Instead of text, a `Doc` object can be passed as input. In this case tokenization is skipped but the rest of the pipeline is run.

NAME	DESCRIPTION
`texts`	A sequence of strings (or `Doc` objects). **TYPE:** `Iterable[Union[str,Doc]]`
KEYWORD-ONLY `as_tuples`	If set to `True`, inputs should be a sequence of `(text, context)` tuples. Output will then be a sequence of `(doc, context)` tuples. Defaults to `False`. **TYPE:** `bool`
`batch_size`	The number of texts to buffer. **TYPE:** `Optional[int]`
`disable`	Names of pipeline components to disable. **TYPE:** `List[str]`
`component_cfg`	Optional dictionary of keyword arguments for components, keyed by component names. Defaults to `None`. **TYPE:** `Optional[Dict[str, Dict[str, Any]]]`
`n_process`	Number of processors to use. Defaults to `1`. **TYPE:** `int`
YIELDS	Documents in the order of the original text. **TYPE:** `Doc`

Language.set_error_handler METHOD V3.0

Define a callback that will be invoked when an error is thrown during processing of one or more documents. Specifically, this function will call `set_error_handler` on all the pipeline components that define that function. The error handler will be invoked with the original component's name, the component itself, the list of documents that was being processed, and the original error.

NAME	DESCRIPTION
`error_handler`	A function that performs custom error handling. **TYPE:** `Callable[[str, Callable[[Doc],Doc], List[Doc], Exception]`

Language.initialize METHOD V3.0

Initialize the pipeline for training and return an `Optimizer`. Under the hood, it uses the settings defined in the `[initialize]` config block to set up the vocabulary, load in vectors and tok2vec weights and pass optional arguments to the `initialize` methods implemented by pipeline components or the tokenizer. This method is typically called automatically when you run `spacy train`. See the usage guide on the config lifecycle and initialization for details.

`get_examples` should be a function that returns an iterable of `Example` objects. The data examples can either be the full training data or a representative sample. They are used to **initialize the models** of trainable pipeline components and are passed each component's `initialize` method, if available. Initialization includes validating the network, inferring missing shapes and setting up the label scheme based on the data.

If no `get_examples` function is provided when calling `nlp.initialize`, the pipeline components will be initialized with generic data. In this case, it is crucial that the output dimension of each component has already been defined either in the config, or by calling `pipe.add_label` for each possible output label (e.g. for the tagger or textcat).

NAME	DESCRIPTION
`get_examples`	Optional function that returns gold-standard annotations in the form of `Example` objects. TYPE: `Optional[Callable[[], Iterable[Example]]]`
KEYWORD-ONLY `sgd`	An optimizer. Will be created via `create_optimizer` if not set. TYPE: `Optional[Optimizer]`
RETURNS	The optimizer. TYPE: `Optimizer`

Language.resume_training METHOD EXPERIMENTAL V3.0

Continue training a trained pipeline. Create and return an optimizer, and initialize "rehearsal" for any pipeline component that has a `rehearse` method. Rehearsal is used to prevent models from "forgetting" their initialized "knowledge". To perform rehearsal, collect samples of text you want the models to retain performance on, and call `nlp.rehearse` with a batch of Example objects.

NAME	DESCRIPTION
KEYWORD-ONLY `sgd`	An optimizer. Will be created via `create_optimizer` if not set. TYPE: `Optional[Optimizer]`
RETURNS	The optimizer. TYPE: `Optimizer`

Language.update METHOD

Update the models in the pipeline.

NAME	DESCRIPTION
`examples`	A batch of `Example` objects to learn from. **TYPE:** `Iterable[Example]`
KEYWORD-ONLY `drop`	The dropout rate. **TYPE:** `float`
`sgd`	An optimizer. Will be created via `create_optimizer` if not set. **TYPE:** `Optional[Optimizer]`
`losses`	Dictionary to update with the loss, keyed by pipeline component. **TYPE:** `Optional[Dict[str, float]]`
`component_cfg`	Optional dictionary of keyword arguments for components, keyed by component names. Defaults to `None`. **TYPE:** `Optional[Dict[str, Dict[str, Any]]]`
RETURNS	The updated `losses` dictionary. **TYPE:** `Dict[str, float]`

Language.rehearse METHOD EXPERIMENTAL V3.0

Perform a "rehearsal" update from a batch of data. Rehearsal updates teach the current model to make predictions similar to an initial model, to try to address the "catastrophic forgetting" problem. This feature is experimental.

NAME	DESCRIPTION
`examples`	A batch of `Example` ≣ objects to learn from. **TYPE:** `Iterable[Example]`
KEYWORD-ONLY	
`drop`	The dropout rate. **TYPE:** `float`
`sgd`	An optimizer. Will be created via `create_optimizer` if not set. **TYPE:** `Optional[Optimizer]`
`losses`	Dictionary to update with the loss, keyed by pipeline component. **TYPE:** `Optional[Dict[str, float]]`
RETURNS	The updated `losses` dictionary. **TYPE:** `Dict[str, float]`

Language.evaluate METHOD

Evaluate a pipeline's components.

NAME	DESCRIPTION
`examples`	A batch of `Example` ≡ objects to learn from. **TYPE:** `Iterable[Example]`
KEYWORD-ONLY `batch_size`	The batch size to use. **TYPE:** `Optional[int]`
`scorer`	Optional `Scorer` ≡ to use. If not passed in, a new one will be created. **TYPE:** `Optional[Scorer]`
`component_cfg`	Optional dictionary of keyword arguments for components, keyed by component names. Defaults to `None`. **TYPE:** `Optional[Dict[str, Dict[str, Any]]]`
`scorer_cfg`	Optional dictionary of keyword arguments for the `Scorer`. Defaults to `None`. **TYPE:** `Optional[Dict[str, Any]]`
`per_component` V3.6 ?	Whether to return the scores keyed by component name. Defaults to `False`. **TYPE:** `bool`
RETURNS	A dictionary of evaluation scores. **TYPE:** `Dict[str, Union[float, Dict[str, float]]]`

Language.use_params CONTEXTMANAGER METHOD

Replace weights of models in the pipeline with those provided in the params dictionary. Can be used as a context manager, in which case, models go back to their original weights after the block.

NAME	DESCRIPTION
`params`	A dictionary of parameters keyed by model ID. **TYPE:** `dict`

Language.add_pipe METHOD

Add a component to the processing pipeline. Expects a name that maps to a component factory registered using `@Language.component` or `@Language.factory`. Components should be callables that take a `Doc` object, modify it and return it. Only one of `before`, `after`, `first` or `last` can be set. Default behavior is `last=True`.

NAME	DESCRIPTION
`factory_name`	Name of the registered component factory. **TYPE:** `str`
`name`	Optional unique name of pipeline component instance. If not set, the factory name is used. An error is raised if the name already exists in the pipeline. **TYPE:** `Optional[str]`
KEYWORD-ONLY	
`before`	Component name or index to insert component directly before. **TYPE:** `Optional[Union[str, int]]`
`after`	Component name or index to insert component directly after. **TYPE:** `Optional[Union[str, int]]`
`first`	Insert component first / not first in the pipeline. **TYPE:** `Optional[bool]`
`last`	Insert component last / not last in the pipeline. **TYPE:** `Optional[bool]`
`config` V3.0	Optional config parameters to use for this component. Will be merged with the `default_config` specified by the component factory. **TYPE:** `Dict[str, Any]`
`source` V3.0	Optional source pipeline to copy component from. If a source is provided, the `factory_name` is interpreted as the name of the component in the source pipeline. Make sure that the vocab, vectors and settings of the source pipeline match the target pipeline. **TYPE:** `Optional[Language]`
`validate` V3.0	Whether to validate the component config and arguments against the types expected by the factory. Defaults to `True`. **TYPE:** `bool`
RETURNS	The pipeline component. **TYPE:** `Callable[[Doc],Doc]`

Language.create_pipe METHOD

Create a pipeline component from a factory.

NAME	DESCRIPTION
`factory_name`	Name of the registered component factory. **TYPE:** `str`
`name`	Optional unique name of pipeline component instance. If not set, the factory name is used. An error is raised if the name already exists in the pipeline. **TYPE:** `Optional[str]`
KEYWORD-ONLY `config` V3.0	Optional config parameters to use for this component. Will be merged with the `default_config` specified by the component factory. **TYPE:** `Dict[str, Any]`
`validate` V3.0	Whether to validate the component config and arguments against the types expected by the factory. Defaults to `True`. **TYPE:** `bool`
RETURNS	The pipeline component. **TYPE:** `Callable[[Doc],Doc]`

Language.has_factory CLASSMETHOD V3.0

Check whether a factory name is registered on the `Language` class or subclass. Will check for language-specific factories registered on the subclass, as well as general-purpose factories registered on the `Language` base class, available to all subclasses.

NAME	DESCRIPTION
`name`	Name of the pipeline factory to check. **TYPE:** `str`
RETURNS	Whether a factory of that name is registered on the class. **TYPE:** `bool`

Language.has_pipe METHOD

Check whether a component is present in the pipeline. Equivalent to `name in nlp.pipe_names`.

NAME	DESCRIPTION
`name`	Name of the pipeline component to check. **TYPE:** `str`
RETURNS	Whether a component of that name exists in the pipeline. **TYPE:** `bool`

Language.get_pipe METHOD

Get a pipeline component for a given component name.

NAME	DESCRIPTION
`name`	Name of the pipeline component to get. **TYPE:** `str`
RETURNS	The pipeline component. **TYPE:** `Callable[[Doc],Doc]`

Language.replace_pipe METHOD

Replace a component in the pipeline and return the new component.

NAME	DESCRIPTION
`name`	Name of the component to replace. **TYPE:** `str`
`component`	The factory name of the component to insert. **TYPE:** `str`
KEYWORD-ONLY	
`config` V3.0	Optional config parameters to use for the new component. Will be merged with the `default_config` specified by the component factory. **TYPE:** `Optional[Dict[str, Any]]`
`validate` V3.0	Whether to validate the component config and arguments against the types expected by the factory. Defaults to `True`. **TYPE:** `bool`
RETURNS	The new pipeline component. **TYPE:** `Callable[[Doc],Doc]`

Language.rename_pipe METHOD

Rename a component in the pipeline. Useful to create custom names for pre-defined and pre-loaded components. To change the default name of a component added to the pipeline, you can also use the `name` argument on `add_pipe` ≣ .

NAME	DESCRIPTION
`old_name`	Name of the component to rename. **TYPE:** `str`
`new_name`	New name of the component. **TYPE:** `str`

Language.remove_pipe METHOD

Remove a component from the pipeline. Returns the removed component name and component function.

NAME	DESCRIPTION
`name`	Name of the component to remove. **TYPE:** `str`
RETURNS	A `(name, component)` tuple of the removed component. **TYPE:** `Tuple[str, Callable[[Doc],Doc]]`

Language.disable_pipe METHOD V3.0

Temporarily disable a pipeline component so it's not run as part of the pipeline. Disabled components are listed in `nlp.disabled` and included in `nlp.components`, but not in `nlp.pipeline`, so they're not run when you process a `Doc` with the `nlp` object. If the component is already disabled, this method does nothing.

NAME	DESCRIPTION
`name`	Name of the component to disable. **TYPE:** `str`

Language.enable_pipe METHOD V3.0

Enable a previously disabled component (e.g. via `Language.disable_pipes`) so it's run as part of the pipeline, `nlp.pipeline`. If the component is already enabled, this method does nothing.

NAME	DESCRIPTION
`name`	Name of the component to enable. **TYPE:** `str`

Language.select_pipes CONTEXTMANAGER METHOD V3.0

Disable one or more pipeline components. If used as a context manager, the pipeline will be restored to the initial state at the end of the block. Otherwise, a `DisabledPipes` object is returned, that has a `.restore()` method you can use to undo your changes. You can specify either `disable` (as a list or string), or `enable`. In the latter case, all components not in the `enable` list will be disabled. Under the hood, this method calls into `disable_pipe` and `enable_pipe`.

NAME	DESCRIPTION
KEYWORD-ONLY	
`disable`	Name(s) of pipeline component(s) to disable. **TYPE:** `Optional[Union[str, Iterable[str]]]`
`enable`	Name(s) of pipeline component(s) that will not be disabled. **TYPE:** `Optional[Union[str, Iterable[str]]]`
RETURNS	The disabled pipes that can be restored by calling the object's `.restore()` method. **TYPE:** `DisabledPipes`

Language.get_factory_meta CLASSMETHOD V3.0

Get the factory meta information for a given pipeline component name. Expects the name of the component **factory**. The factory meta is an instance of the `FactoryMeta` dataclass and contains the information about the component and its default provided by the `@Language.component` or `@Language.factory` decorator.

NAME	DESCRIPTION
`name`	The factory name. **TYPE:** `str`
RETURNS	The factory meta. **TYPE:** `FactoryMeta`

Language.get_pipe_meta METHOD V3.0

Get the factory meta information for a given pipeline component name. Expects the name of the component **instance** in the pipeline. The factory meta is an instance of the `FactoryMeta` dataclass and contains the information about the component and its default provided by the `@Language.component` or `@Language.factory` decorator.

NAME	DESCRIPTION
`name`	The pipeline component name. **TYPE:** `str`
RETURNS	The factory meta. **TYPE:** `FactoryMeta`

Language.analyze_pipes METHOD V3.0

Analyze the current pipeline components and show a summary of the attributes they assign and require, and the scores they set. The data is based on the information provided in the `@Language.component` and `@Language.factory` decorator. If requirements aren't met, e.g. if a component specifies a required property that is not set by a previous component, a warning is shown.

Example output

NAME	DESCRIPTION
KEYWORD-ONLY	
`keys`	The values to display in the table. Corresponds to attributes of the `FactoryMeta`. Defaults to `["assigns", "requires", "scores", "retokenizes"]`. **TYPE:** `List[str]`
`pretty`	Pretty-print the results as a table. Defaults to `False`. **TYPE:** `bool`
RETURNS	Dictionary containing the pipe analysis, keyed by `"summary"` (component meta by pipe), `"problems"` (attribute names by pipe) and `"attrs"` (pipes that assign and require an attribute, keyed by attribute). **TYPE:** `Optional[Dict[str, Any]]`

Language.replace_listeners METHOD V3.0

Find listener layers (connecting to a shared token-to-vector embedding component) of a given pipeline component model and replace them with a standalone copy of the token-to-vector layer. The listener layer allows other components to connect to a shared token-to-vector embedding component like `Tok2Vec` or `Transformer`. Replacing listeners can be useful when training a pipeline with components sourced from an existing pipeline: if multiple components (e.g. tagger, parser, NER) listen to the same token-to-vector component, but some of them are frozen and not updated, their performance may degrade significantly as the token-to-vector component is updated with new data. To prevent this, listeners can be replaced with a standalone token-to-vector layer that is owned by the component and doesn't change if the component isn't updated.

This method is typically not called directly and only executed under the hood when loading a config with sourced components that define `replace_listeners`.

NAME	DESCRIPTION
`tok2vec_name`	Name of the token-to-vector component, typically `"tok2vec"` or `"transformer"`. **TYPE:** `str`
`pipe_name`	Name of pipeline component to replace listeners for. **TYPE:** `str`
`listeners`	The paths to the listeners, relative to the component config, e.g. `["model.tok2vec"]`. Typically, implementations will only connect to one tok2vec component, `model.tok2vec`, but in theory, custom models can use multiple listeners. The value here can either be an empty list to not replace any listeners, or a *complete* list of the paths to all listener layers used by the model that should be replaced. **TYPE:** `Iterable[str]`

Language.meta PROPERTY

Meta data for the `Language` class, including name, version, data sources, license, author information and more. If a trained pipeline is loaded, this contains meta data of the pipeline. The `Language.meta` is also what's serialized as the `meta.json` when you save an `nlp` object to disk. See the meta data format for more details.

NAME	DESCRIPTION
RETURNS	The meta data. **TYPE:** `Dict[str, Any]`

Language.config PROPERTY V3.0

Export a trainable `config.cfg` for the current `nlp` object. Includes the current pipeline, all configs used to create the currently active pipeline components, as well as the default training config that can be used with `spacy train`. `Language.config` returns a Thinc `Config` object, which is a subclass of the built-in `dict`. It supports the additional methods `to_disk` (serialize the config to a file) and `to_str` (output the config as a string).

NAME	DESCRIPTION
RETURNS	The config. **TYPE:** `Config`

Language.to_disk METHOD

Save the current state to a directory. Under the hood, this method delegates to the `to_disk` methods of the individual pipeline components, if available. This means that if a trained pipeline is loaded, all components and their weights will be saved to disk.

NAME	DESCRIPTION
`path`	A path to a directory, which will be created if it doesn't exist. Paths may be either strings or `Path`-like objects. **TYPE:** `Union[str,Path]`
KEYWORD-ONLY	
`exclude`	Names of pipeline components or serialization fields to exclude. **TYPE:** `Iterable[str]`

Language.from_disk METHOD

Loads state from a directory, including all data that was saved with the `Language` object. Modifies the object in place and returns it.

NAME	DESCRIPTION
`path`	A path to a directory. Paths may be either strings or `Path`-like objects. **TYPE:** `Union[str, Path]`
KEYWORD-ONLY `exclude`	Names of pipeline components or serialization fields to exclude. **TYPE:** `Iterable[str]`
RETURNS	The modified `Language` object. **TYPE:** `Language`

Language.to_bytes METHOD

Serialize the current state to a binary string.

NAME	DESCRIPTION
KEYWORD-ONLY `exclude`	Names of pipeline components or serialization fields to exclude. **TYPE:** `iterable`
RETURNS	The serialized form of the `Language` object. **TYPE:** `bytes`

Language.from_bytes METHOD

Load state from a binary string. Note that this method is commonly used via the subclasses like `English` or `German` to make language-specific functionality like the lexical attribute getters available to the loaded object.

Note that if you want to serialize and reload a whole pipeline, using this alone won't work, you also need to handle the config. See "Serializing the pipeline" for details.

NAME	DESCRIPTION
`bytes_data`	The data to load from. **TYPE:** `bytes`
KEYWORD-ONLY `exclude`	Names of pipeline components or serialization fields to exclude. **TYPE:** `Iterable[str]`
RETURNS	The `Language` object. **TYPE:** `Language`

Attributes

NAME	DESCRIPTION
`vocab`	A container for the lexical types. **TYPE:** `Vocab`
`tokenizer`	The tokenizer. **TYPE:** `Tokenizer`
`make_doc`	Callable that takes a string and returns a `Doc`. **TYPE:** `Callable[[str],Doc]`
`pipeline`	List of `(name, component)` tuples describing the current processing pipeline, in order. **TYPE:** `List[Tuple[str, Callable[[Doc],Doc]]]`
`pipe_names`	List of pipeline component names, in order. **TYPE:** `List[str]`
`pipe_labels`	List of labels set by the pipeline components, if available, keyed by component name. **TYPE:** `Dict[str, List[str]]`
`pipe_factories`	Dictionary of pipeline component names, mapped to their factory names. **TYPE:** `Dict[str, str]`
`factories`	All available factory functions, keyed by name. **TYPE:** `Dict[str, Callable[[…], Callable[[Doc],Doc]]]`
`factory_names` V3.0	List of all available factory names. **TYPE:** `List[str]`
`components` V3.0	List of all available `(name, component)` tuples, including components that are currently disabled. **TYPE:** `List[Tuple[str, Callable[[Doc],Doc]]]`
`component_names` V3.0	List of all available component names, including components that are currently disabled. **TYPE:** `List[str]`
`disabled` V3.0	Names of components that are currently disabled and don't run as part of the pipeline. **TYPE:** `List[str]`
`path`	Path to the pipeline data directory, if a pipeline is loaded from a path or

	package. Otherwise `None` . TYPE: `Optional[Path]`

Class attributes

NAME	DESCRIPTION
`Defaults`	Settings, data and factory methods for creating the `nlp` object and processing pipeline. TYPE: `Defaults`
`lang`	IETF language tag, such as 'en' for English. TYPE: `str`
`default_config`	Base config to use for Language.config . Defaults to `default_config.cfg` . TYPE: `Config`

Defaults

The following attributes can be set on the `Language.Defaults` class to customize the default language data:

NAME	DESCRIPTION
`stop_words`	List of stop words, used for `Token.is_stop`. **Example:** `stop_words.py` </> TYPE: `Set[str]`
`tokenizer_exceptions`	Tokenizer exception rules, string mapped to list of token attributes. **Example:** `de/tokenizer_exceptions.py` </> TYPE: `Dict[str, List[dict]]`
`prefixes`, `suffixes`, `infixes`	Prefix, suffix and infix rules for the default tokenizer. **Example:** `puncutation.py` </> TYPE: `Optional[Sequence[Union[str,Pattern]]]`
`token_match`	Optional regex for matching strings that should never be split, overriding the infix rules. **Example:** `fr/tokenizer_exceptions.py` </> TYPE: `Optional[Callable]`
`url_match`	Regular expression for matching URLs. Prefixes and suffixes are removed before applying the match. **Example:** `tokenizer_exceptions.py` </> TYPE: `Optional[Callable]`
`lex_attr_getters`	Custom functions for setting lexical attributes on tokens, e.g. `like_num`. **Example:** `lex_attrs.py` </> TYPE: `Dict[int, Callable[[str], Any]]`
`syntax_iterators`	Functions that compute views of a `Doc` object based on its syntax. At the moment, only used for noun chunks. **Example:** `syntax_iterators.py` </> . TYPE: `Dict[str, Callable[[Union[Doc,Span]], Iterator[Span]]]`
`writing_system`	Information about the language's writing system, available via `Vocab.writing_system`. Defaults to: `{"direction": "ltr", "has_case": True, "has_letters": True}.`. **Example:** `zh/__init__.py` </> TYPE: `Dict[str, Any]`
`config`	Default config added to `nlp.config`. This can include references to custom tokenizers or lemmatizers. **Example:** `zh/__init__.py` </> TYPE: `Config`

Serialization fields

During serialization, spaCy will export several data fields used to restore different aspects of the object. If needed, you can exclude them from serialization by passing in the string names via the `exclude` argument.

NAME	DESCRIPTION
`vocab`	The shared `Vocab` ≣ .
`tokenizer`	Tokenization rules and exceptions.
`meta`	The meta data, available as `Language.meta` ≣ .
...	String names of pipeline components, e.g. `"ner"`.

FactoryMeta DATACLASS V3.0

The `FactoryMeta` contains the information about the component and its default provided by the `@Language.component` ≣ or `@Language.factory` ≣ decorator. It's created whenever a component is defined and stored on the `Language` class for each component instance and factory instance.

NAME	DESCRIPTION
`factory`	The name of the registered component factory. **TYPE:** `str`
`default_config`	The default config, describing the default values of the factory arguments. **TYPE:** `Dict[str, Any]`
`assigns`	`Doc` or `Token` attributes assigned by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. **TYPE:** `Iterable[str]`
`requires`	`Doc` or `Token` attributes required by this component, e.g. `["token.ent_id"]`. Used for pipe analysis. **TYPE:** `Iterable[str]`
`retokenizes`	Whether the component changes tokenization. Used for pipe analysis. **TYPE:** `bool`
`default_score_weights`	The scores to report during training, and their default weight towards the final score used to select the best model. Weights should sum to `1.0` per component and will be combined and normalized for the whole pipeline. If a weight is set to `None`, the score will not be logged or weighted. **TYPE:** `Dict[str, Optional[float]]`
`scores`	All scores set by the components if it's trainable, e.g. `["ents_f", "ents_r", "ents_p"]`. Based on the `default_score_weights` and used for pipe analysis. **TYPE:** `Iterable[str]`

</> SUGGEST EDITS