МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ГУМАНИТАРНЫЙ ИНСТИТУТ

КУРСОВАЯ РАБОТА

по дисциплине: «Информатика и основы программирования»

на тему:

«Реализация базовых методов NLP для финского языка при использовании функционала библиотеки “SpaCy”»

Выполнил:

студент 1 курса группы 24812

Чёрненький Егор Константинович

Научный руководитель:

магистр техники и технологии,

старший преподаватель

Леман Анна Яковлевна

Новосибирск, 2025

**ОГЛАВЛЕНИЕ**

[**ОГЛАВЛЕНИЕ** 2](#_Toc195922625)

[**ВВЕДЕНИЕ** 3](#_Toc195922626)

[**ГЛАВА 1. ОБЩАЯ ХАРАКТЕРИСТИКА ФУНКЦИОНАЛА SPACY** 5](#_Toc195922627)

[1.1. Загрузка статистической модели в проект. Language-объект 5](#_Toc195922628)

[1.2. Vocab. Линия обработки SpaCy. Doc-объект 6](#_Toc195922629)

[1.3 Основные компоненты линии обработки 7](#_Toc195922630)

[**ГЛАВА 2. ОПИСАНИЕ СКРИПТА, ИСПОЛЬЗУЮЩЕГО NLP-МЕТОДЫ SPACY** 12](#_Toc195922631)

[2.1. Инициализация Language- и создание Doc-объекта 12](#_Toc195922632)

[2.2. Цикл определения команд 13](#_Toc195922633)

[2.3. Выполнение функций, занесённых в список команд 15](#_Toc195922634)

[**ЗАКЛЮЧЕНИЕ** 18](#_Toc195922635)

[**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ** 19](#_Toc195922636)

[**ПРИЛОЖЕНИЕ А** 20](#_Toc195922637)

**ВВЕДЕНИЕ**

Данная работа направлена на изучение машинных методов обработки естественного языка (Natural Language Processing, или NLP) и освоение принципов работы с ними и их вызова на примере библиотеки SpaCy.

В последние годы, резкий рост использования языковых моделей для решения как теоретических, так и прикладных задач привёл к повышению авторитета области NLP. Данному аспекту программирования характерна проблематика создания связи между машинным кодом и человеческим языком путём автоматической обработки текстов с восприятием и систематизацией лингвистической информации (обычно с использованием n-базисных векторов), относящейся к различным уровням языковой структуры, свойственной естественным языкам. Способы общения искусственного интеллекта с пользователем основаны на вышеназванном процессе.

Актуальность данного исследования обусловлена подходом наглядной систематизации универсальных приёмов NLP и их иллюстрации в написанном скрипте, что может быть полезно для тех, кто начинает знакомиться с азами данной набирающей популярность области.

В ходе работы были использованы метод абстрагирования, метод классификации, а также метод моделирования.

Метод абстрагирования был задействован при чтении источников, когда требовалось выделить интегральные признаки объектов составляемой классификации и отсеять избыточную в её рамках информацию.

Метод классификации лёг в основу теоретической части исследования. В ней охарактеризованы как синтагматические отношения компонентов (в линии обработки SpaCy учитывается порядок), так и иерархические отношения между компонентами и их методами и атрибутами (потому что в рамках объектно-ориентированного программирования объектами в SpaCy являются компоненты линии обработки, которые определяются со свойствами, отличными от свойств друг друга).

Метод моделирования применялся непосредственно во второй части данной работы в ходе создания варианта программы, осуществляющей долю функционала библиотеки.

Итак, объектом исследования является библиотека SpaCy.

Предмет исследования – компоненты линии обработки (pipeline components) SpaCy, которые соответствуют методам NLP, а также их взаимодействие друг с другом и с остальными классами библиотеки.

Задачи работы:

1)систематизировать основные методы обработки естественных языков на примере компонентов SpaCy и охарактеризовать онтологию данной библиотеки;

2)создать скрипт в Python, демонстрирующий работу указанных методов на произвольном тексте финского (неиндоевропейского) языка.

**ГЛАВА 1. ОБЩАЯ ХАРАКТЕРИСТИКА ФУНКЦИОНАЛА SPACY**

1.1. Загрузка статистической модели в проект. Language-объект

Для полноценной работы всех компонентов SpaCy ожидается загрузить обученную линию обработки (trained pipeline), располагающую статистическими данными и способную предсказывать статус языковых единиц, например определять часть речи омонима. Ещё одним достоинством обученной линии обработки являются векторы слов, по которым можно определять их схожесть, как формальную, так и семантическую.

Для каждого языка доступно для загрузки 3 основных типа статистических моделей:

* Малая (например, “fi\_core\_news\_sm” для финского языка);
* Средняя (“fi\_core\_news\_md”);
* Большая (“fi\_core\_news\_lg”).

Отличаются они скоростью работы, размером словаря, количеством векторов и компонентами в линии обработки.

При вызове статистическая модель определяет Language-объект, который, в свою очередь, каждый раз получая строку как аргумент, возвращает Doc-объект, который и используется в дальнейшем для получения желаемых данных.

*Вызов статистической модели и определение Language-объекта приведены на рисунке 1.*

*Отображение компонентов линии обработки «fi\_core\_news\_sm» через атрибут pipe\_names, принадлежащий Language объекту приведено на рисунке 2.*

1.2. Vocab. Линия обработки SpaCy. Doc-объект

Ключевыми понятиями в принципе работы SpaCy, помимо Language-объекта являются его производные: Vocab и Doc-объект.

Vocab – словарь, общее хранилище лексем языка статистической модели. Оно определяется вместе с Language-объектом и к нему можно обратиться через nlp.vocab, где “nlp” – Language-объект. Элемент словаря – класс Lexeme – слово как часть языка, а не речи, а потому он не имеет информации о синтаксической роли и части речи (ср. “progress” в английском). Во время обработки текста программа анализирует каждую единицу текста, и чтобы определить, какое слово стоит за той или иной подстрокой, она сверяется с хранилищем лексем и находит совпадение подстроки с лексемой. Лексема содержит только ту информацию о слове, которая не зависит от контекста, например, его графическую оболочку и парадигму словоизменения.

Doc-объект – объект, хранящий результат обработки конкретного текста компонентами Language-объекта, за которым, как уже было упомянуто, стоит статистическая модель. Фактически это список токенов (см. «токенизация» в 1.3.) – строк с соответствующей им лингвистической информацией. Каждый компонент имеет определённое место в очереди компонентов. На рисунке 2 отображен список компонентов, обрабатывающих текст последовательно и передающих обновлённый Doc-объект следующему компоненту линии обработки. Чем выше индекс компонента в списке, тем позже он вступит в работу.

*Пользователь имеет доступ к редактированию очереди через методы “remove\_pipe”, “add\_pipe” объекта Language (см. рисунок 2 и 3).*

Неотъемлемый этап обработки в SpaCy – это конвертация каждого элемента текста в хэш-значение: каждой единице текста соответствует последовательность цифр, которая закреплена за ней вне зависимости от проекта или размера статистической модели. Это позволяет экономить место в памяти, так как в ином случае для каждой омонимичной единицы создавалась бы отдельная строка. За соответствие строки хэш значению отвечает “StringStore”, относящийся к объекту “Vocab”, являющемуся атрибутом класса “Language”. Чтобы получить по имени строки хэш значение, можно обратиться к StringStore через ***формулу (1)*:**

**nlp.vocab.strings[word] (1)**

где “word” – строка.

Кодированию в хэш-значения подвергается не только строка, но и лингвистическая аннотация, относимая к ней в результате работы того или иного компонента линии обработки. Например, атрибут “pos” произвольного токена содержит число, в то время как обращение к “pos\_” возвращает привычное обозначение части речи *(см. рисунок 4)*

1.3 Основные компоненты линии обработки

Компоненты, состоящие по умолчанию в обучаемых моделях, обладают широкими возможностями по добавлению морфологических, семантических и синтаксических аннотаций. Как уже было указано выше, они работают в определённом порядке, который доступен для редактирования пользователем. Далее будут перечислены ключевые и наиболее часто используемые из них.

1)Tokenizator. Неотъемлемая часть любой линии обработки, всегда следующая в самом начале. Этот компонент невозможно исключить из работы, потому он даже не отражается в списке компонентов на рис. 2.

Процесс токенизации делит текст на токены, которые соотносятся с объектами Lexeme из Vocab, обретая тем самым смысл и переставая быть просто строками для программы. Токенизация похожа на строковый метод text.split(‘ ‘) из Python, однако помимо словоформ и знаков пунктуации она добавляет ещё и пробельные символы в список токенов. К тому же токенизация способна быть многоступенчатой: на каждом шаге программа проверяет, можно ли от подстроки отделить знак препинания, получив при этом два отдельных токена. Также подстрока сверяется со списком исключений, которые нельзя делить дальше, несмотря на наличие знаков препинания внутри слова.

Важно, что с самого своего появления токен не тождественен строке, по которой он был создан, он иерархически выше её. Для получения строки по токену к ней обращаются ***по формуле (2)***:

**token.text (2)**

Таким образом, токенизатор создает изначальную версию Doc-объекта. На этом этапе за каждой словоформой стоит лишь соответствие со значением слова. Чтобы токены «обросли» актуальными для текста данными, в работу вступают остальные компоненты линии обработки.

2)Morphologizer. Этот компонент ответственен за определение части речи по набору Universal Part Of Speech Tag Set (UPOS), а также грамматических свойств словоформы: в финском у существительных это число и падеж, у глаголов – залог, число, время и т.д.

Атрибуты компонента, передаваемые классу Token:

* “pos” (возвращает хэш-значение, то есть число);
* “pos\_” (возвращает строку);
* “morph” (возвращает грамматические признаки).

Обращение ко всем трём атрибутам отображено на рисунке 4.

3)Tagger. Этот компонент определяет за точное определение части речи по многим наборам тегов частей речи, в отличие от Morphologizer.

Атрибуты компонента, передаваемые классу Token:

* “tag” (возвращает хэш-значение);
* “tag\_” (возвращает строку).

*Обращение к атрибутам отображено на рисунке 5.*

4)Dependency Parser. Устанавливает границы предложений, а также синтаксические зависимости слов в предложении, определяя в этих зависимостях главные и побочные слова. Основано на алгоритме Arc-eager на основе переходов.

В границах предложений выделяются именные группы (noun chunks - словосочетания с существительным в главной роли), которые синтаксически зависимы от глагола, определяемого как центр предложения.

Атрибуты компонента, передаваемые классу Token:

* “dep” (тип синтаксической связи, возвращает хэш-значение);
* “dep\_” (тип синтаксической связи);
* “head” (главное слово в синтаксической связи, в которой задействован токен);
* “is\_sent\_start” (True или False, в зависимости от того, начинается ли предложение с данного токена).

Атрибут, передаваемый классу Doc:

* “sents” (доступ к итератору, в котором перебираются предложения doc-объекта).

*На рисунке 6 для каждого слова в предложении охарактеризован тип синтаксической связи.*

5)Lemmatizer. Используется для соотнесения токена с его основной словарной формой. Из-за возможной омонимии требует привязки к части речи у токена.

Атрибуты, передаваемые классу Token:

* “lemma” (основная форма слова, возвращает хэш-значение);
* “lemma\_” (основная форма слова).

*На рисунке 7 выведено отображение начальной формы финского существительного.*

6)Named Entity Recognizer (NER). Компонент выделяет из находящихся рядом токенов те слова или неразрывные коллокации, которые являются названиями реальных явлений действительности: людей, мест, компаний и т.д.

Пересечение названий, удовлетворяющих условиям, образует список Span-объектов (срезов Doc-объекта), не пересекающихся друг с другом.

Атрибуты, передаваемые классу Token:

* “ent\_type” (категория названия, например “GPE” для географических названий, “ORG” для организаций; возвращает хэш-значение);
* “ent\_type\_” (категория названия).
* Атрибут, передаваемый классу Doc:
* “ents” (множество срезов, являющихся названиями).

7)Attribute Ruler. Используется для ручного переназначения атрибутов для токенов. Токены определяются при помощи Token Matcher.

***Формула 3***. Добавление элемента в Attribute Ruler:

**ar.add([patterns], attrs, index) (3)**

Где **ar = nlp.get\_pipe(“attribute\_ruler”);**

“patterns” – список словарей, в котором каждому словарю в формате {“Критерий”: “Требование”} соответствует токен, то есть “patterns” обозначает последовательность токенов, на которое будет распространяться изменение атрибутов;

“attrs” – словарь, в котором пары организованы по принципу {“Атрибут токена”: “Новое значение”};

“index” – номер токена из коллокации, совпавшей с “patterns”, над которым будет осуществляться замена атрибутов.

*На рис. 8 осуществлена привязка атрибута “POS” = “NOUN” вместо “PROPN” к токену “ENCEn”.*

8)Tok2vec. Это компонент, используемый для хранения обобщённых семантических характеристик. Векторное представление используют вышеописанные компоненты для оценки вероятности правильности предсказаний. Векторы в разных статистических моделях (“sm”, “md”, “lg”) отличаются размерностью

*Визуальное представление векторов слов приведено на рис. 9.*

В практической части данной работы напрямую векторы использовались только один раз – при обращении к методу “similarity()”, обозначющему смысловую схожесть токенов, Span- или Doc-объектов.

Это все компоненты линии обработки, которые были задействованы в практической части исследования.

*Схема линии обработки приведена на рисунке 10.*

Можно сделать вывод, что для многосторонней характеристики объектов любого текста их достаточно в силу имеющихся заранее предоставленного числа слов-векторов и натренированности на примерах. К тому же, отношения классов объектов в SpaCy довольно интуитивно представимы и роль их в функционировании линии обработки довольно легко постичь. При этом не была затронута гораздо более труднодоступная тема обучения собственных компонентов на базе имеющихся, от инициализации до наполнения примерами и тренировки. Тем не менее, цель этой курсовой работы состоит лишь в знакомстве с общими приёмами NLP, в их вызове и использовании, что было реализовано в результате написания скрипта.

**ГЛАВА 2. ОПИСАНИЕ СКРИПТА, ИСПОЛЬЗУЮЩЕГО NLP-МЕТОДЫ SPACY**

Итак, по мере закрепления кратко описанных ранее методов в состоянии написания находилась программа, позволяющая пользователю определять, какие функции обработки выполнить.

Условно программу можно представить в 3 этапа:

1)Инициализация Language- и Doc-объекта;

2)Цикл определения команд;

3)Выполнение функций, занесённых в список команд.

На этих этапах возникло несколько трудностей, связанных с особенностями классов SpaCy, о которых будет сказано ниже.

Далее следует подробное описание процессов, происходящих в рамках этапов скрипта.

2.1. Инициализация Language- и создание Doc-объекта

После импортирования модуля и загрузки модели “fi\_core\_news\_sm” должно быть определение Doc-объекта, требующего текст.

Загрузка текста осуществляется через конструкцию “with open … as…”, для предоставления пользователю возможности работать с желаемым ему текстом. Оставив поле ввода пустым, пользователь дает сигнал программе о загрузке текстового файла “finnish.txt”, который был использован при составлении программы по умолчанию.

За этим следует создание Doc-объекта и выводится список компонентов статистической модели SpaCy, по умолчанию поставляемых в “fi\_core\_news\_sm”.

2.2. Цикл определения команд

Весь этап построен на условных операторах и бесконечном цикле, прерываемом командой “break” при прописывании сигнала о переходе к следующему этапу – команде “done”.

Цель этого этапа – предоставление пользователю выбора, какие результаты обработки должны быть выведены, и, соответственно, какие функции, отвечающие за их вывод, должны быть задействованы.

Далее будут перечислены команды, использованные в скрипте, и их назначение.

* “punctuation” – вывод всех знаков препинания из текста;
* “numbers” – вывод всех чисел из текста;
* “measure length” – вывод длины Doc-объекта в токенах наряду с длиной отфильтрованного текста (без стоп-слов[[1]](#footnote-1));
* “stop words” – вывод стоп-слов из текста;
* “filter” – по усмотрению пользователя:

1)Вывод токенов отфильтрованного (без стоп-слов и знаков препинания) Doc-объекта;

2)Выборочная фильтрация Doc-объекта (с конфигурацией отсеиваний стоп-слов и знаков препинания) с возможностью дальнейшего вызова команд в цикле для отфильтрованного теста;

* “hash values” – вывод токенов Doc-объекта с соответствующими им хэш-значениями;
* “lemmatize” – вывод токенов Doc-объекта с соответствующими им начальными формами;
* “uppercase tokens” – вывод токенов Doc-объекта, состоящих из символов верхнего регистра;
* “help” – вывод общего описания работы программы
* “tag POS” – вывод токенов Doc-объекта с соответствующими им обозначениями части речи;
* “visualize” – создать визуализацию синтаксических отношений для каждого предложения в Doc-объекте;
* “display names” – вывести наименования (“named entities”, не ‘tokens”) вместе с соответствующими типами наименования;
* “match” – вывести совпадения с шаблоном, задаваемым пользователем, из текста;
* “phrase match” – вывести совпадения с фразами (строками), задаваемыми пользователем, из текста;
* “compare” – вывести степень схожести двух токенов, определяемых:

1)вводом строки от пользователя;

2)вводом индекса от пользователя.

Таким образом, можно определить, какие атрибуты токенов/компоненты будут использоваться в каждой из следующих за командами функциях.

* В функциях первых команд данного списка будут использоваться собственные атрибуты токенов (напр. булевые “is\_stop”, is\_punct”);
* В функции по команде “hash values” используется связь линии обработки с StringStore в Vocab;
* “lemmatize” использует атрибут, полученный токеном от Lemmatiser;
* “tag POS” использует атрибут “pos\_” от Morphologizer и “tag\_” от Tagger;
* “visualize” использует модуль “Displacy”;
* “display names” использует хранилище наименований, создаваемое в результате работы Named Entity Recognizer;
* “match” и “phrase match” используют функции семейства “Match”, принимающие шаблоны и данные о лексике и возвращающие Match-объекты, через которые можно итерировать, получая Span-объекты (срезы токенов в Doc);
* Функция, стоящая за “compare”, обращается к векторам токенов и показывает индекс смысловой соотнесённости двух токенов.

Таким образом, в этом скрипте было задействовано большинство компонентов малой финской статистической модели.

2.3. Выполнение функций, занесённых в список команд

В этой части кода описаны функции, соотносящиеся с командами, а также через оператор “if” обусловлено их выполнение.

Большинство функций реализованы перебором токенов и постановкой конкретных условий через их атрибуты.

* Так, функция “punctuation” перебирает токены и при условии того, что токен является знаком препинания, выводит его и его индекс (через собственный атрибут).
* “numbers” работает аналогично, но вместо одного булевого атрибута стоит другой, определяющий, является ли токен числом.
* “measure\_length” определяет длину двух Doc-объектов (в токенах).

Для создания версии Doc-объекта без стоп-слов и знаков препинания производится удобный однострочный перебор исходных токенов с двумя условиями (на стоп-слово и на знак препинания).

В итоге создаётся список из токенов, а не Doc-объект, но их длина измеряется аналогично.

* Принцип работы “stop\_words” аналогичен первым двум функциям.
* Команда “filter” может вести к прерыванию цикла и выводу отфильтрованного списка токенов, получаемого таким же образом, как и в “measure\_length”.

При выборе фильтрации Doc-объекта возникает серьёзная трудность: при описании предыдущих функций хватало помещения токенов в список, так как этот список использовался впоследствии только для вывода токенов через итерацию.

При фильтрации стоит задача сохранить статус Doc-объекта, так как у него после фильтрации предстоит работа с атрибутами, эксклюзивными для Doc-объекта (например обращение к StringStore в Vocab). Если этого не сделать, выдаётся AttributeError.

Единственный обнаруженный способ заключается в том, чтобы после фильтрации и при создании списка токенов преобразовывать список в строку, а затем с помощью метода “sub” в регулярных выражениях избавиться от присущих списку квадратных скобок и запятых. Полученная строка, лишённая облика списка, может быть превращена в Doc-объект снова.

Итак, имеется три варианта фильтрации:

1)Избавиться только от стоп-слов, но не от знаков препинания. Самый сложный случай, так как он требует сохранения исходных запятых. После однострочного отбора итератором всех токенов, не являющихся стоп-словами, при помощи “sub” заменяются квадратные скобки на “”, а также группы “, “ на “ “. После эта строка конвертируется в Doc-объект.

2)Избавиться только от знаков препинания, что осуществимо в рамках одного “sub”. Все эти знаки, включая квадратные скобки, заменяются на “”.

3)Избавиться и от того, и от другого. Предполагается тот же вариант, что и в 2), но с дополнительным условием в начале при отсеивании токенов.

* “hash\_values” стандартным перебором токенов и обращением к StringStore получает хэш-значение.
* “lemmatise” стандартным перебором выводит токены с соответствующими начальными формами.
* “uppercase\_tokens” считывает атрибут “is\_upper” и в зависимости от него выводит текст токена, также учитывая условие с отсутствием элементов, состоящих из символов верхнего регистра.
* “tag\_pos” стандартным перебором выводит текст токена и обозначения его части речи по версии Morphologizer и Tagger.
* “visualize”, считывая предложения из Doc-объекта, для каждого из них должен вывести схему синтаксических отношений его участников. Согласно документации, “displacy.serve()” должен создавать частный сервер, отображаемый через браузер.
* “display\_named\_entities” итерирует через хранилище “ents”, созданное Named Entity Recognizer. Выводит текст наименования в соответствии с его типом.
* “match” вызывает Matcher, инициализирующийся через Vocab языковой модели. После этого пользователь создаёт “pattern” – список из словарей, в котором каждый словарь соответствует токену.

Каждый ключ словаря – наименование атрибута токена, а значение по ключу – необходимое для удовлетворения условию совпадения значение. Пользователь сам решает, сколько токенов задавать в паттерне. Притом заведомо нестроковые значения должны быть конвертированы в соответствующие значения по атрибутам.

После паттерн добавляется в Matcher, и Matcher применяется на Doc-объекте. Это возвращает серию Span-объектов (срезов в Doc), которые можно перебирать и выводить.

* “phrase match” работает похожим образом. Вместо одного паттерна, по которому происходит сравнение, предполагается добавить несколько строк, каждая из которых конвертируется в Doc-объектов. Совпадение с любым из паттернов при использовании Phrase Matcher на Doc добавит Span в серию срезов.
* “compare\_tokens” применяет векторную репрезентацию токенов для определения их семантической схожести при помощи метода “similarity()”. Если пользователь будет вводить индексы токенов в тексте, то для нахождения токенов используется перебор и сравнение индексов.

Для написания программы было достаточно поверхностных знаний касательно онтологии SpaCy. Не был полностью реализован раздел “help” в функции “match”, а также были описаны не все возможные команды, осуществляющие элементарный перебор токенов. Это обусловлено избыточным числом атрибутов токена, а также тем, что не все из них могут представлять какой-либо прикладной интерес.

**ЗАКЛЮЧЕНИЕ**

В ходе исследования были выполнены задачи, описанные во вступлении.

Было составлено общее представление о взаимоотношениях классов SpaCy, описаны Language-, Vocab-, Doc- и Token-объекты, а также охарактеризована линия обработки текстов в виде списка её компонентов, а также методов, позволяющих изменить расстановку компонентов обработки.

Были раскрыты атрибуты, передаваемые токенам основных компонентов и указаны примеры обращения к ним.

Полученные знания были закреплены на практике в программе, реализующую некоторые методы анализа текстов. Была описана структура самой программы, а также проведено объяснение процессов на каждом из её этапов.

Указана связь функций внутри программы с результатами работы конкретных компонентов линии обработки, и описаны сами функции. Упомянуты также трудности при написании данного скрипта, связанные с особенностями объектов использованной библиотеки.

Подводя итоги, код, являющийся результатом работы, может быть использован для основных манипуляций с токенами, а исследование в целом может представлять пользу для начинающих освоение NLP.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Matthew Honnibal, Mark Johnson. An Improved Non-monotonic Transition System for Dependency Parsing. URL: <https://www.aclweb.org/anthology/D15-1162/> (дата обращения: 10.04.2025)
2. Daniel Jurafsky, James H. Martin. Speech and Language Processing. // Sequence Labeling for Parts of Speech and Named Entities. URL: <https://web.stanford.edu/~jurafsky/slp3/17.pdf> (дата обращения: 17.04.2025)
3. SpaCy Tutorial – Complete Writeup. URL: <https://www.machinelearningplus.com/spacy-tutorial-nlp/> (дата обращения: 15.03.2025)
4. SpaCy 101: Everything you need to know. URL: <https://spacy.io/usage/spacy-101/> (дата обращения: 18.03.2025)
5. SpaCy Linguistic Features. URL: <https://spacy.io/usage/linguistic-features> (дата обращения: 08.04.2025)
6. SpaCy API. URL: <https://spacy.io/api/> (дата обращения: 18.03.2025)
7. SpaCy Visualizers. URL: <https://spacy.io/usage/visualizers/> (дата обращения: 24.04.2025)
8. SpaCy Installation. URL: <https://spacy.io/usage> (дата обращения: 11.02.2025)
9. SpaCy Trained Models & Pipelines. URL: <https://spacy.io/models> (дата обращения: 29.01.2025)
10. Universal POS tags. URL: <https://universaldependencies.org/u/pos/index.html> (дата обращения: 03.04.2025)

**ПРИЛОЖЕНИЕ А**

**Иллюстративный материал**



Рисунок 1. Импортирование модуля и инициализация Language-объекта

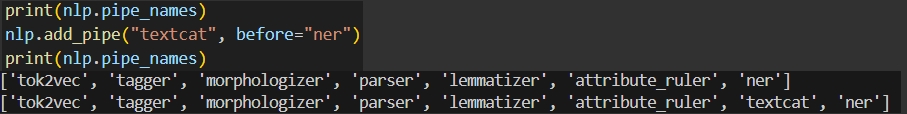


Рисунок 2. Добавление Named Entity Recognizer в линию обработки

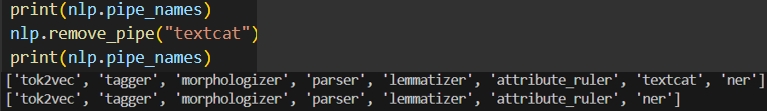


Рисунок 3. Удаление Text Categorizer из линии обработки

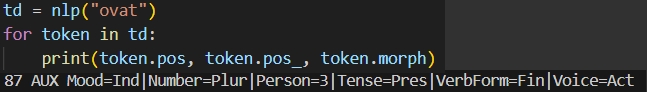


Рисунок 4. Атрибуты Morphologizer

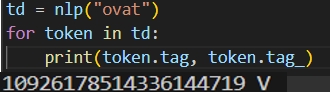


Рисунок 5. Атрибуты Tagger

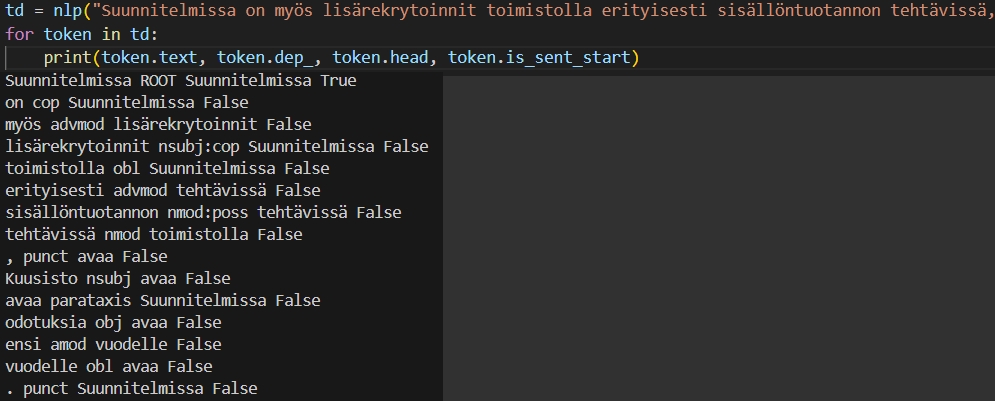


Рисунок 6. Характеристика синтаксических связей слов

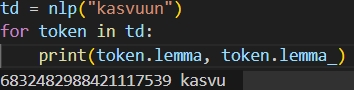


Рисунок 7. Атрибуты Lemmatizer

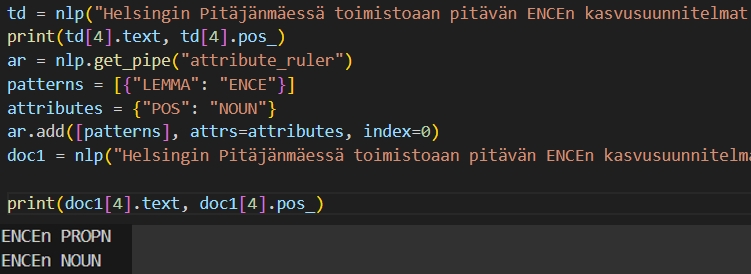


Рисунок 8. Использование Attribute Ruler

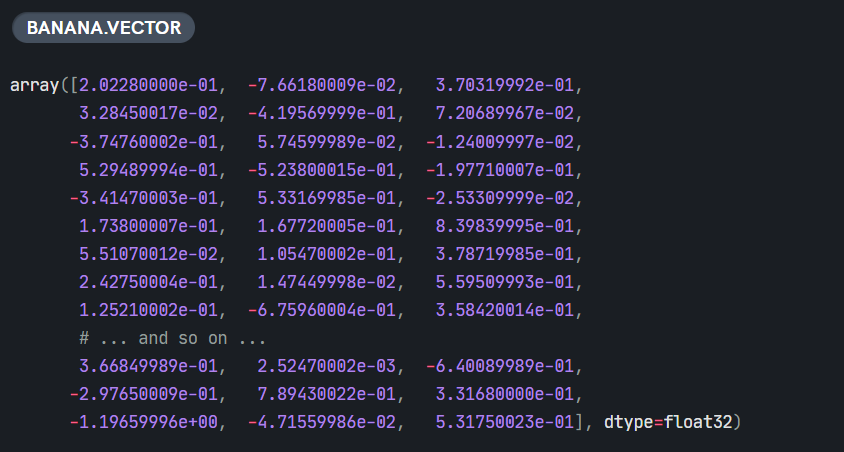


Рисунок 9. Вид вектора, полученного из слова. Источник: SpaCy 101 [4]

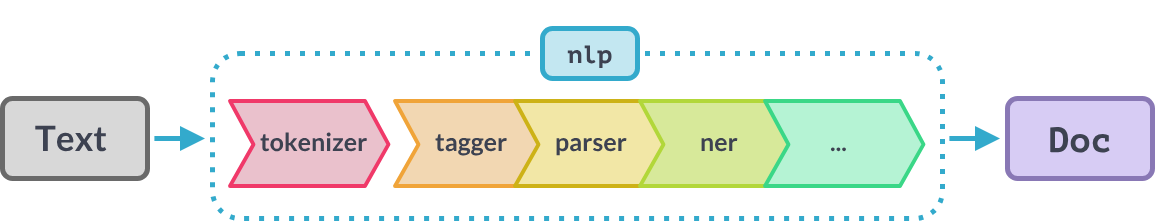


Рисунок 10. Схема линии обработки. Источник: SpaCy 101 [4]

1. Здесь и далее: стоп-слова – фразы или слова, не имеющие смысловой нагрузки, а потому удаляемые из текста [↑](#footnote-ref-1)