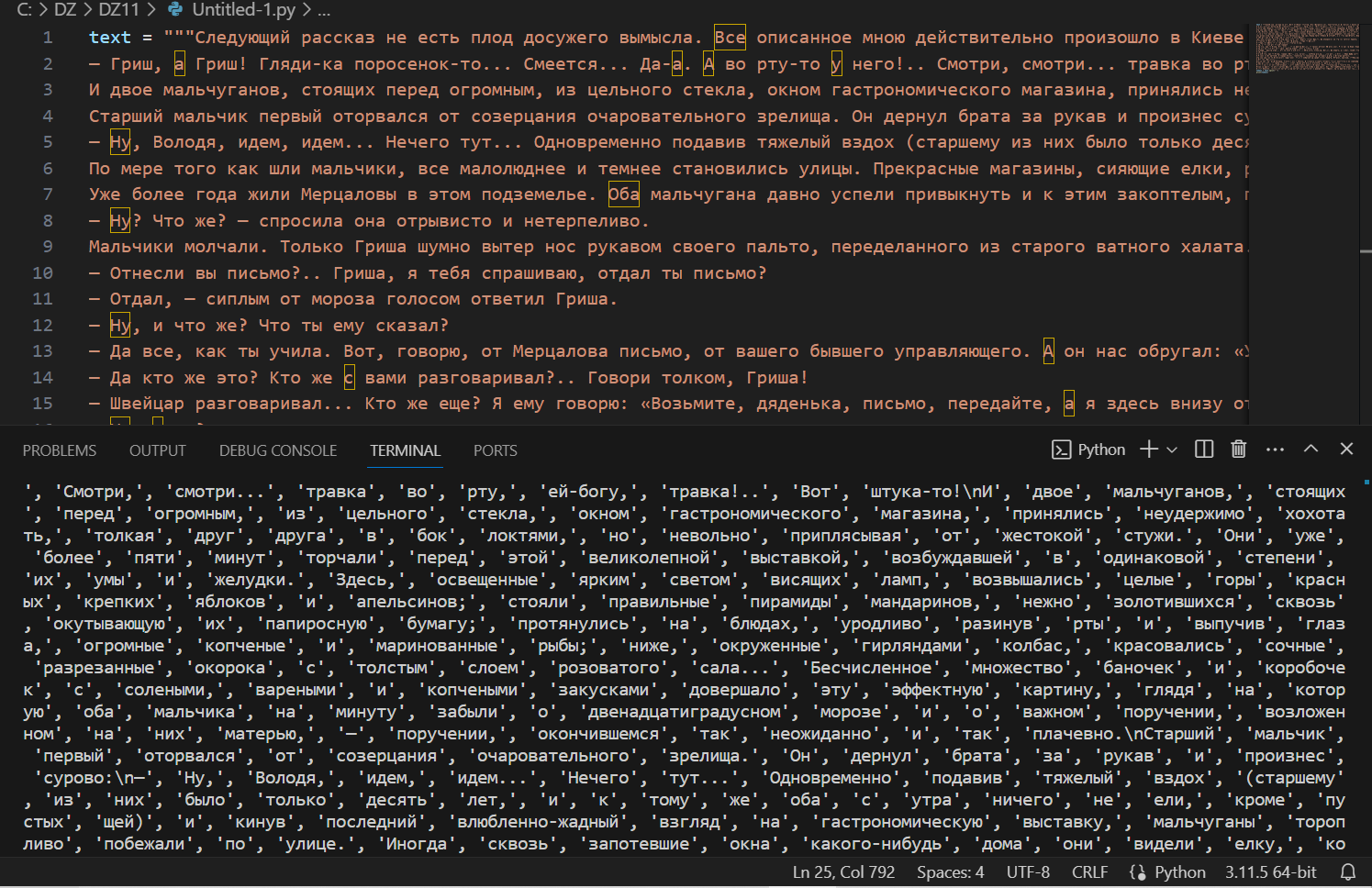
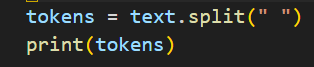
Лабораторная работа 13-14 (уровень 1)

*Задания уровня 1*

- провести токенизацию методом .split

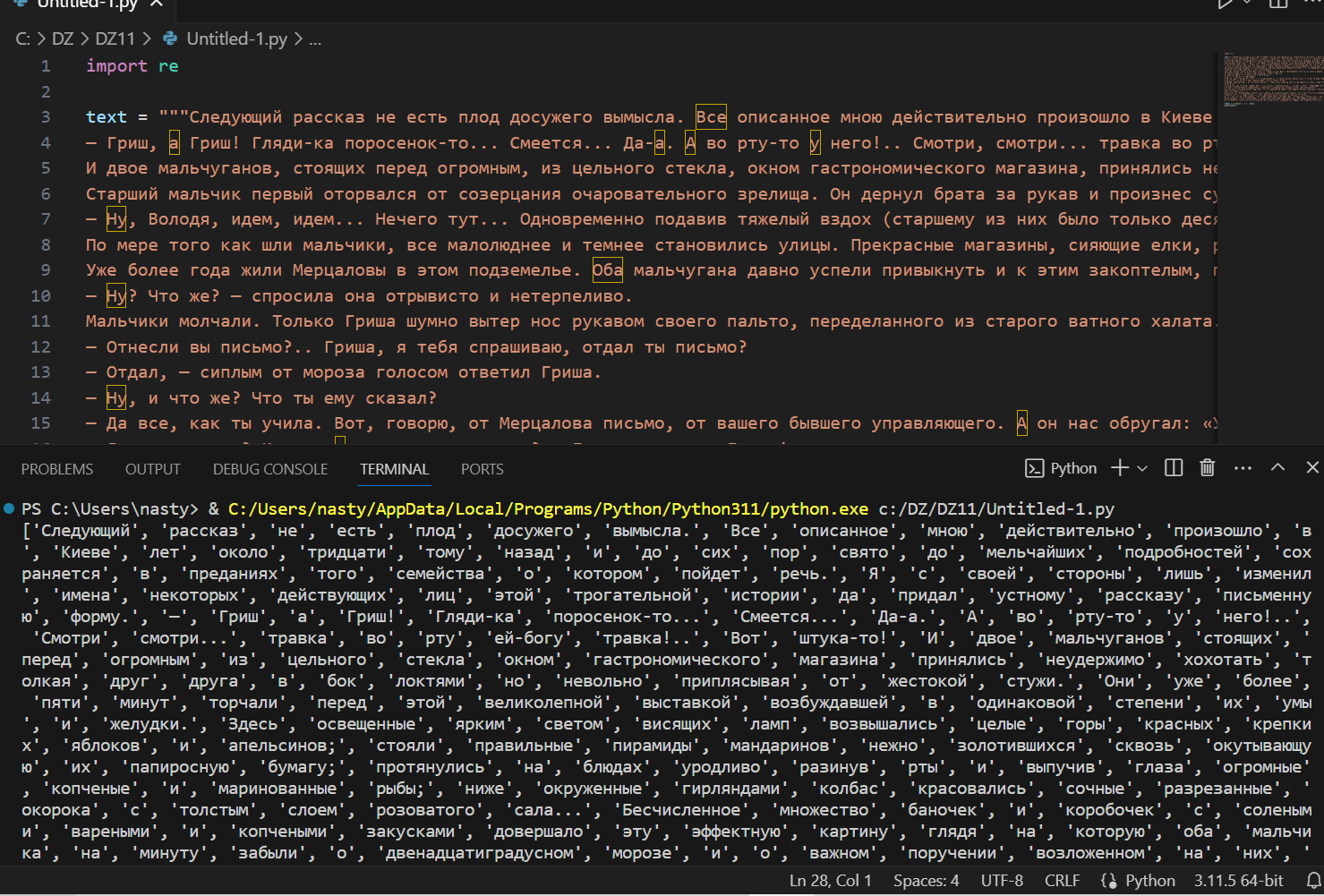
Отрывок текста Куприна «Чудесный доктор»

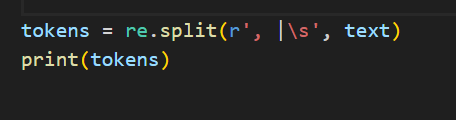


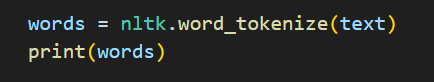
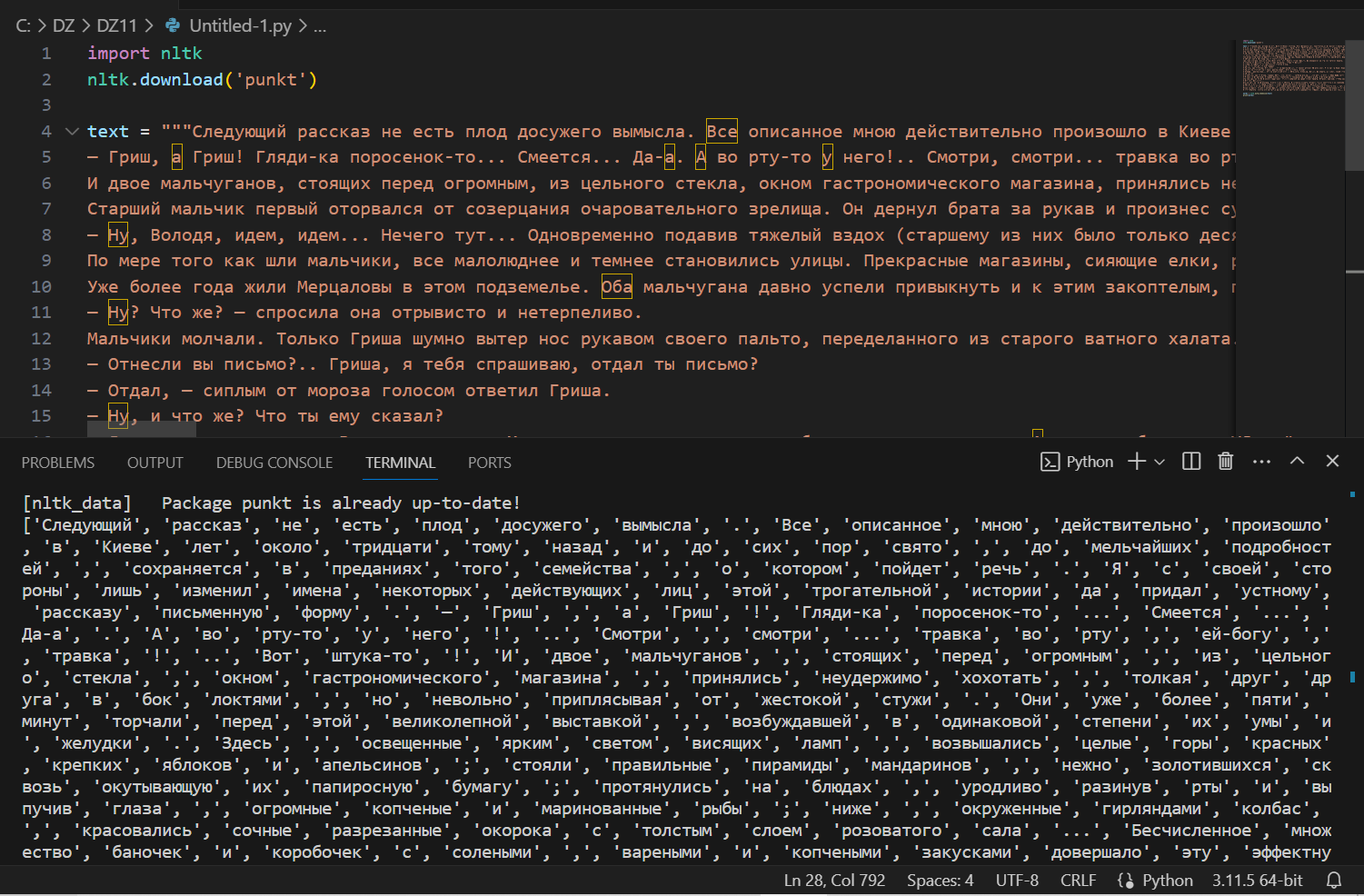


- провести токенизацию при помощи регулярных выражений

Токенезация по запятым и пробелам:





- провести токенизацию при помощи библиотеки NLTK

? в чем отличия разных способов токенизации?

1. Метод .split:

* Простой в использовании.
* Разделяет текст по заданному разделителю (обычно пробелу).
* Не учитывает пунктуацию и специальные символы.

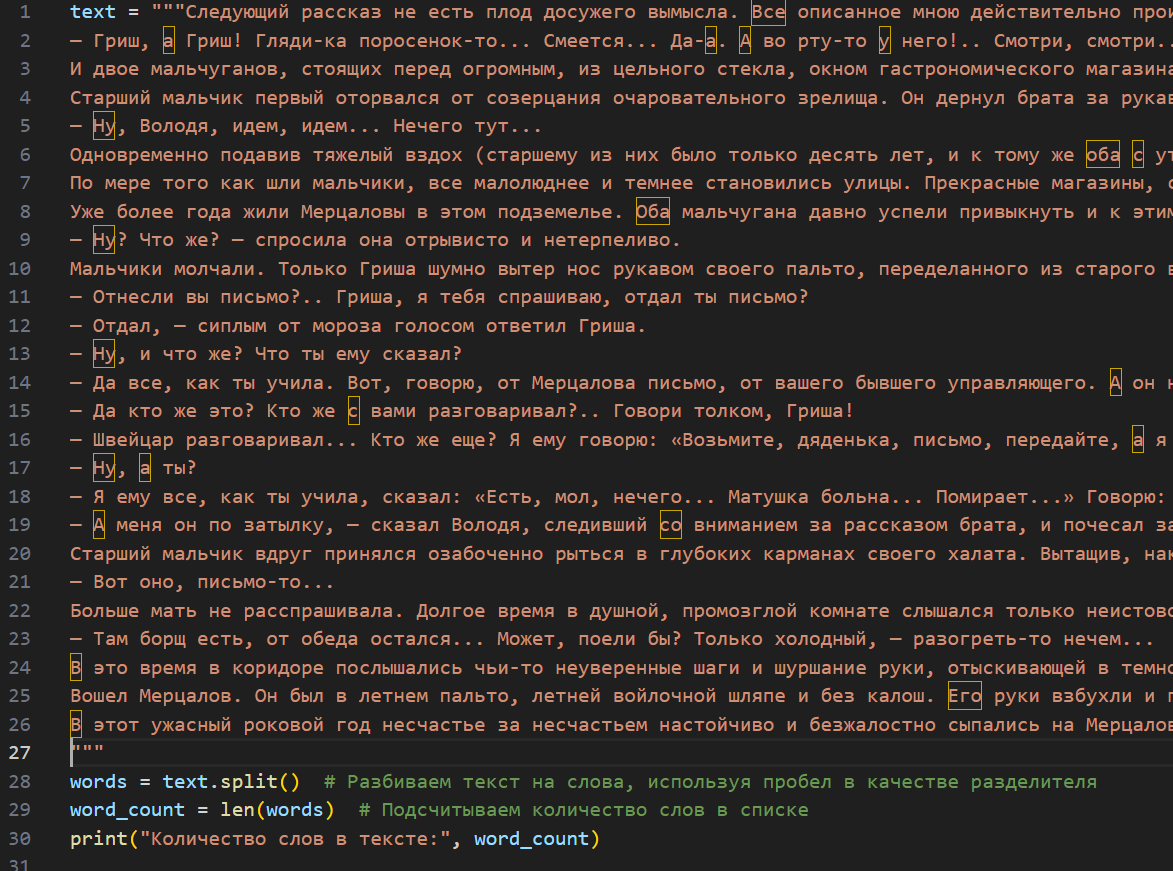
2. Токенизация при помощи регулярных выражений:

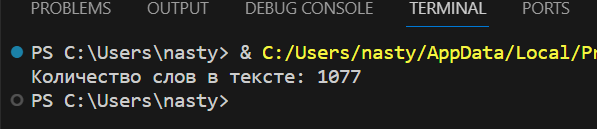
* Позволяет более гибко настраивать правила разделения текста.
* Учитывает пунктуацию и специальные символы.
* Может быть сложнее в использовании из-за необходимости работы с регулярными выражениями.

3. Токенизация при помощи библиотеки NLTK:

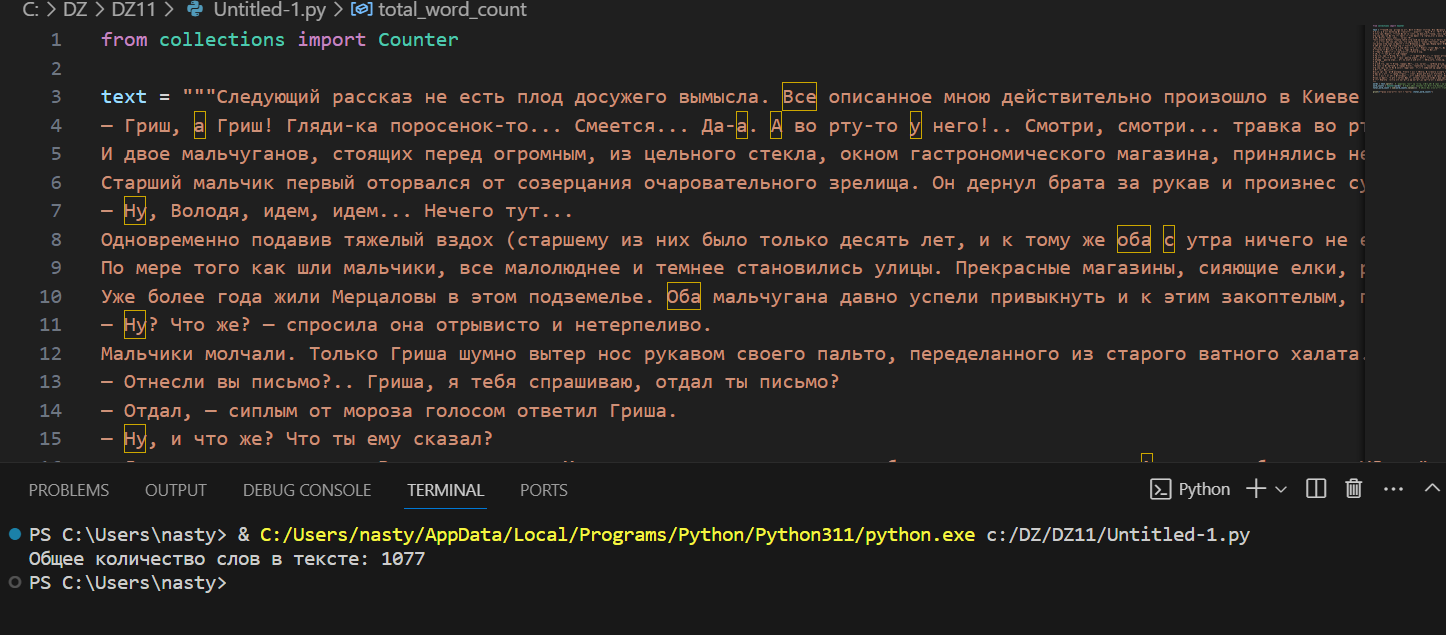
* Предоставляет широкий выбор методов токенизации для различных языков.
* Умеет обрабатывать особенности языка, такие как апострофы, дефисы и другие.
* Может требовать предварительной установки библиотеки.

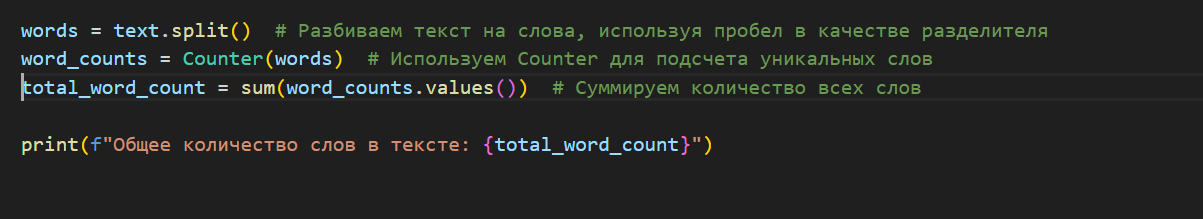
- посчитать количество слов в тексте при помощи len





- посчитать количество слов в тексте при помощи Counter

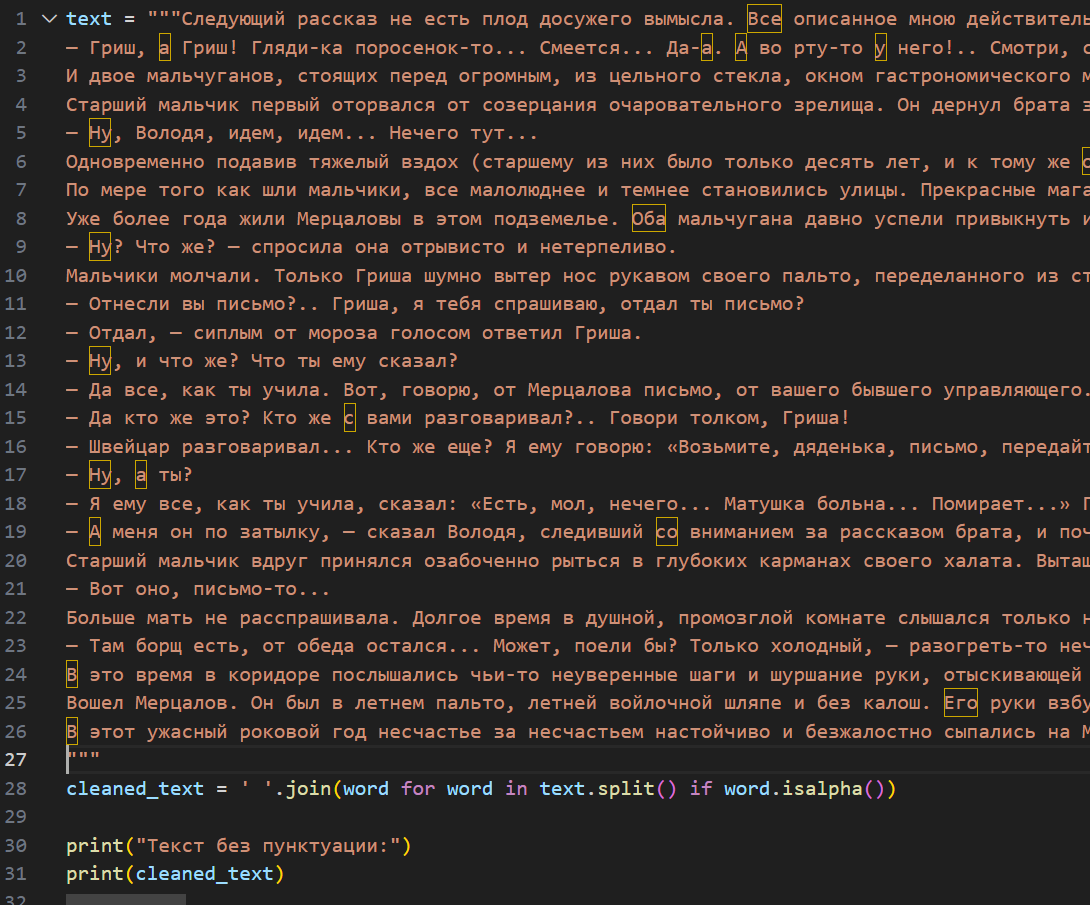




? есть ли отличия в результате?

Отличий в результате нет

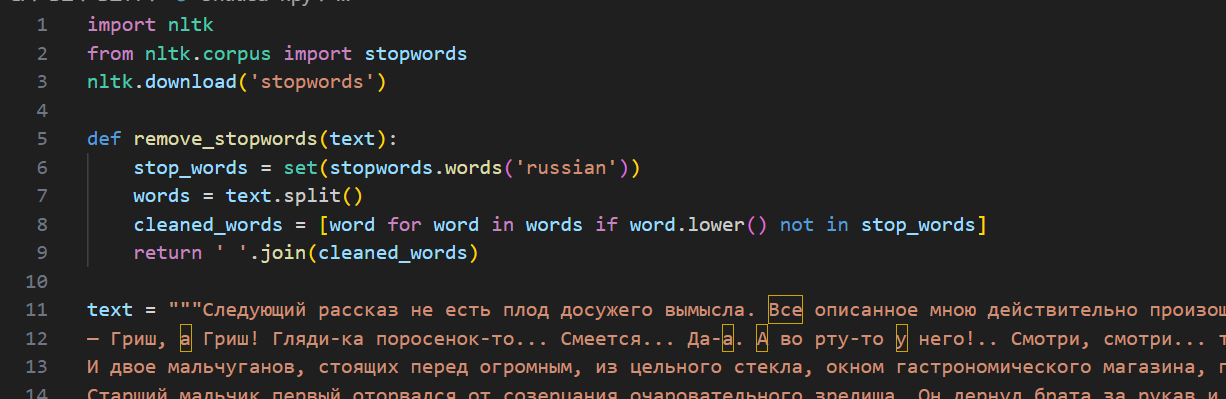
- очистить от пунктуации при помощи isalpha

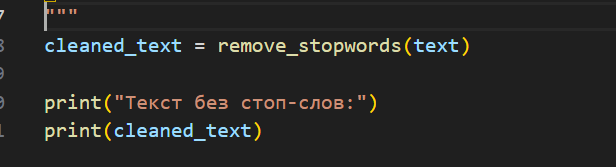


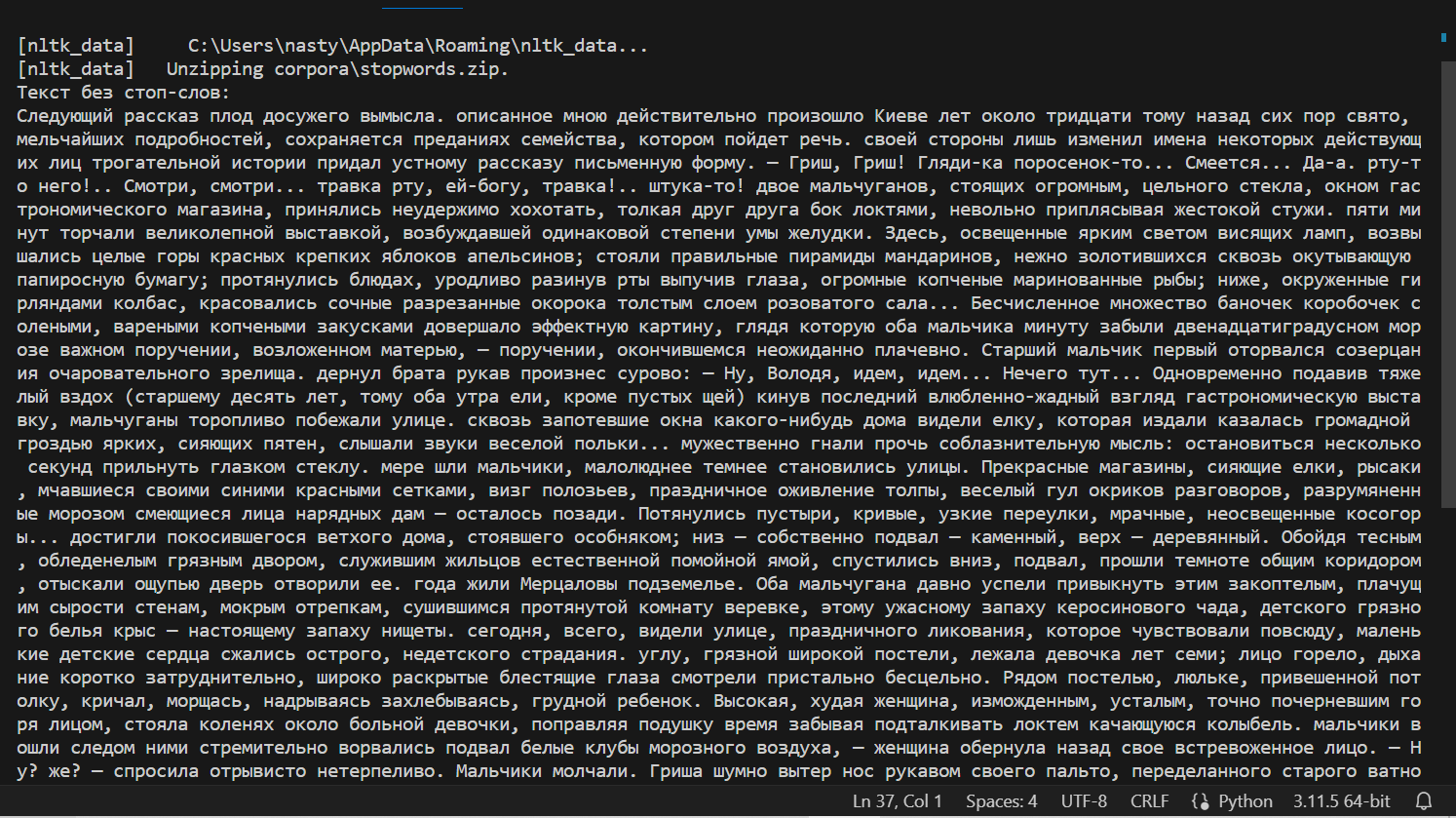


Этот код разобьет текст на отдельные слова, а затем пройдется по каждому слову, проверяя, состоит ли оно только из букв (без пунктуации). Затем он объединит эти слова обратно в строку, разделяя их пробелами, и выведет текст без пунктуации.

- очистить текст от стоп-слов при помощи библиотеки NLTK

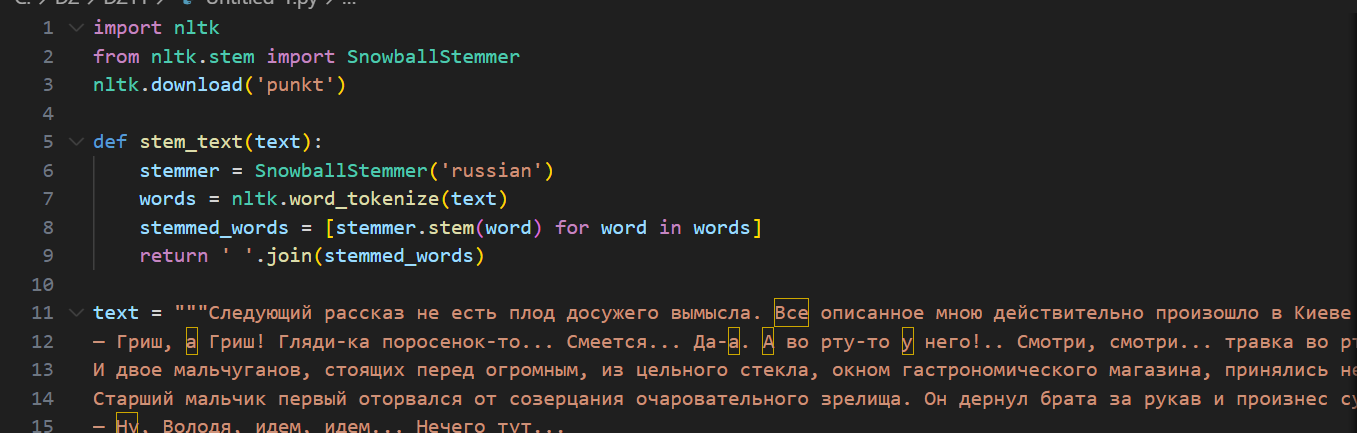


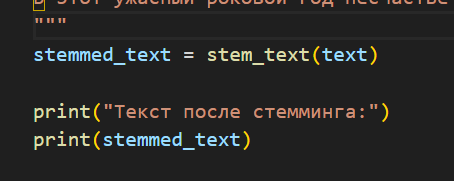


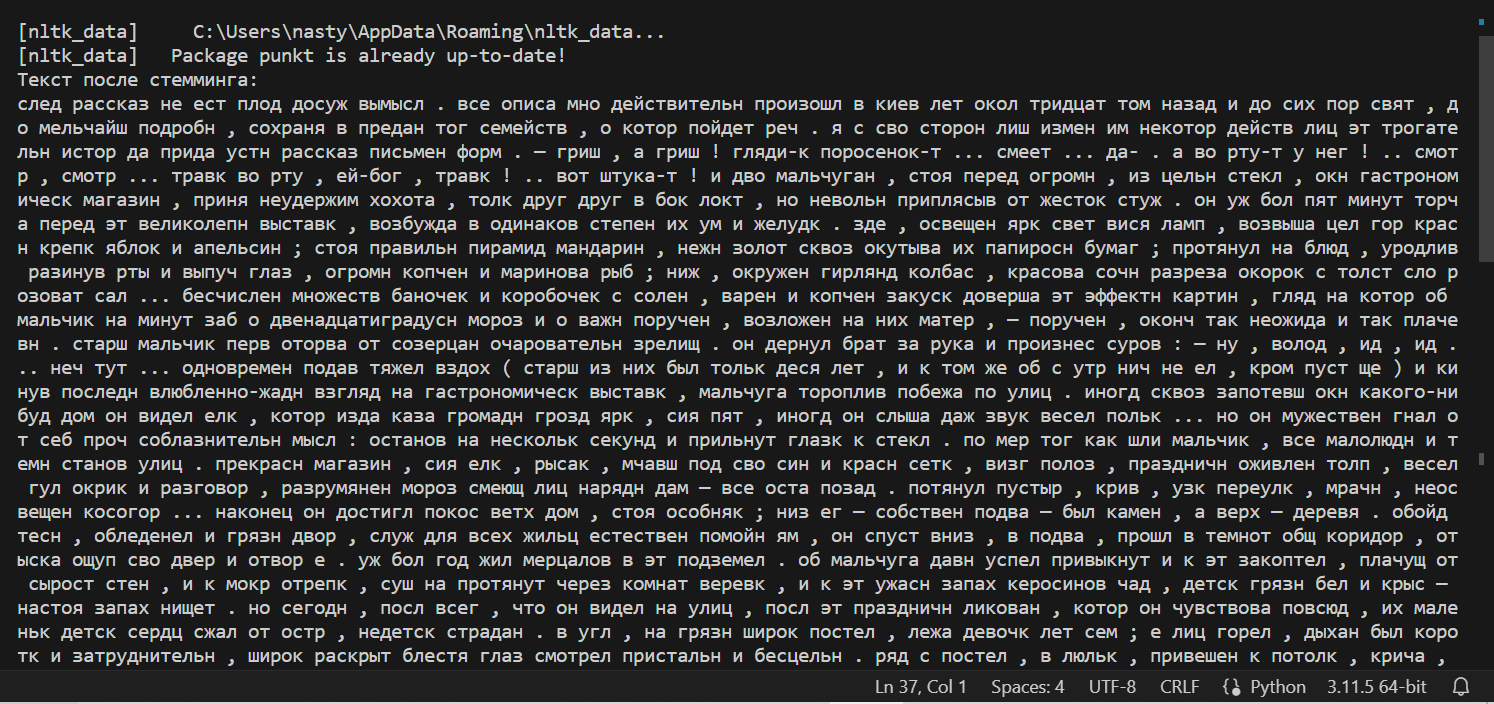


В этом коде мы используем модуль stopwords из NLTK для получения списка стоп-слов на русском языке. Затем мы разбиваем текст на отдельные слова и проверяем, не является ли каждое слово стоп-словом. Если слово не является стоп-словом, мы добавляем его в список очищенных слов. Наконец, мы объединяем очищенные слова обратно в строку и выводим результат.

- провести стемминг при помощи библиотеки NLTK







В этом коде мы используем стеммер SnowballStemmer из NLTK для русского языка. Мы разбиваем текст на отдельные слова с помощью nltk.word\_tokenize, затем применяем стеммер к каждому слову и сохраняем результат. Наконец, мы объединяем стеммированные слова обратно в строку и выводим итоговый текст после стемминга.

- провести лемматизацию при помощи pymorphy2



Не удалось лемматизировать текст☹

? в чем отличия результатов после стемминга и лемматизации?

Стемминг и лемматизация — это два различных метода нормализации текста в обработке естественного языка, имеют различия в том, как они обрабатывают слова.

1. Стемминг:

- Цель: усечение слов до их основы (стема) путем удаления аффиксов.

- Применение: стеммеры работают по жестким правилам и обычно быстрее лемматизации, но могут приводить к потере смысла из-за усечения слов до корня.

2. Лемматизация:

- Цель: приведение слов к их нормальной (словарной) форме.

- Применение: лемматизация использует словарь и грамматический анализ для приведения слов к их базовой форме, что помогает сохранить смысл текста.

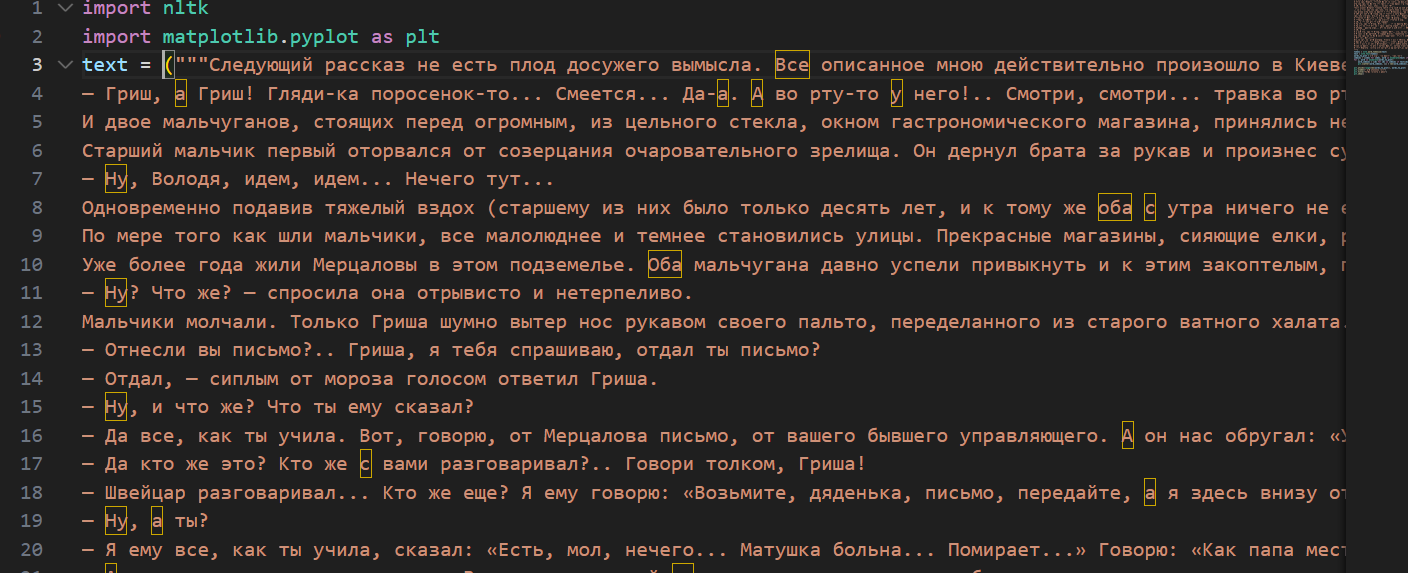
Отличия:

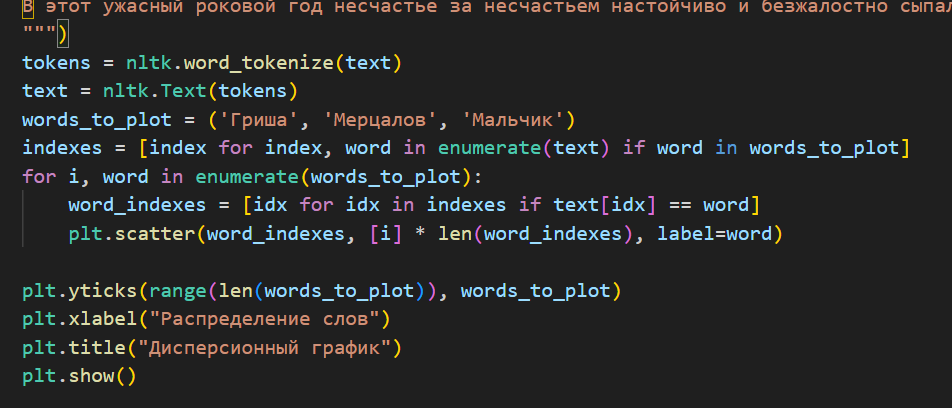
* Стемминг может привести к несуществующим или некорректным словам, так как он просто обрезает аффиксы, в то время как лемматизация приводит к реальным словам из словаря.
* Лемматизация более сложный и точный процесс, чем стемминг, так как она учитывает грамматические правила языка.
* Лемматизация обычно требует больше времени и ресурсов, чем стемминг, из-за более сложного алгоритма.

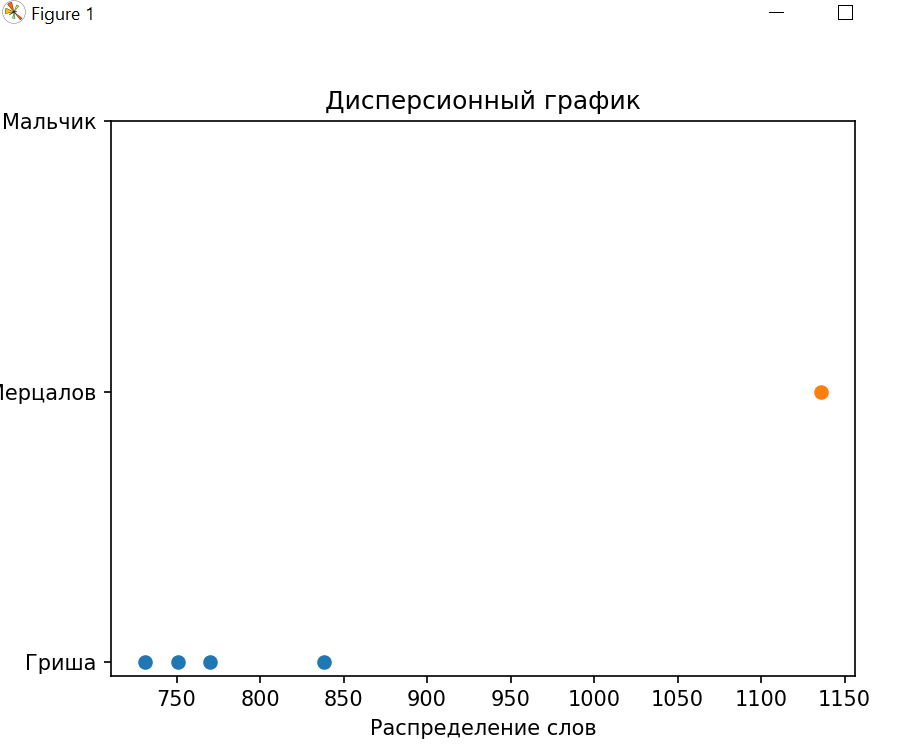
- построить график наиболее частотных (лемматизированных) слов текста при помощи matplotlib

Из - за того, что не удаётся лемматизировать текст, у меня не получается выполнить это задание.

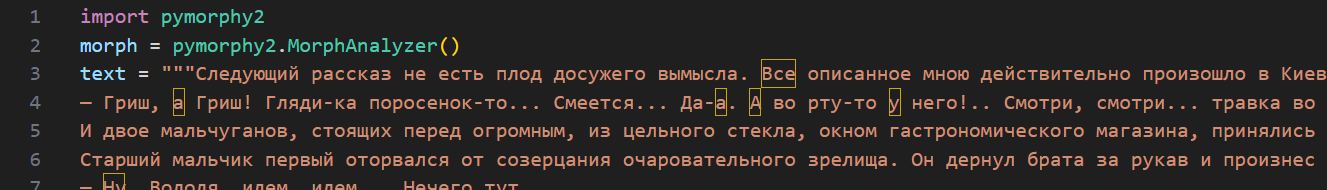
- построить dispersion plot при помощи одноименного метода из библиотеки NLTK

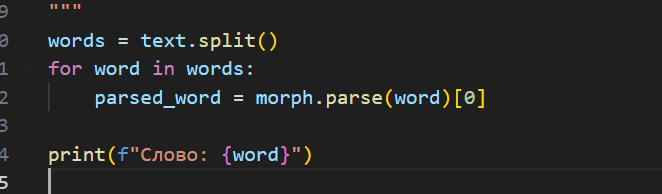


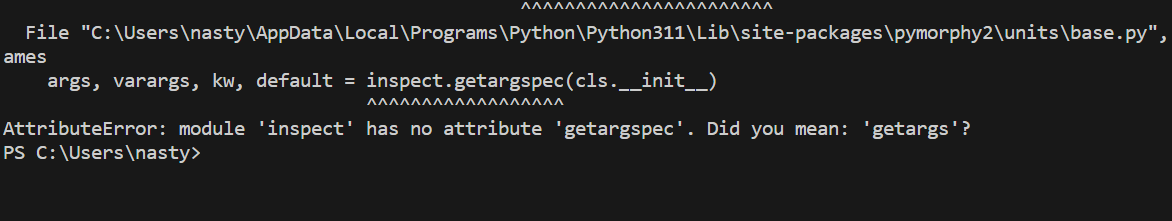


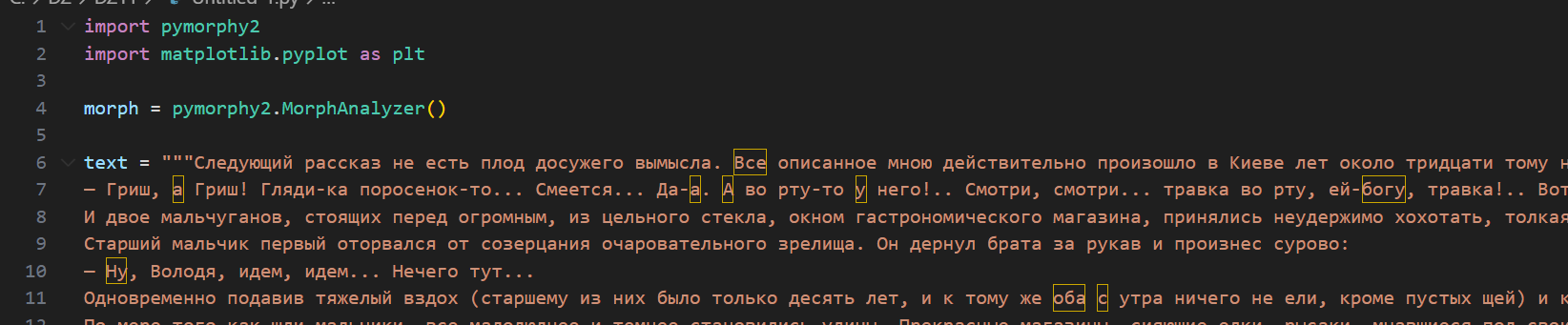


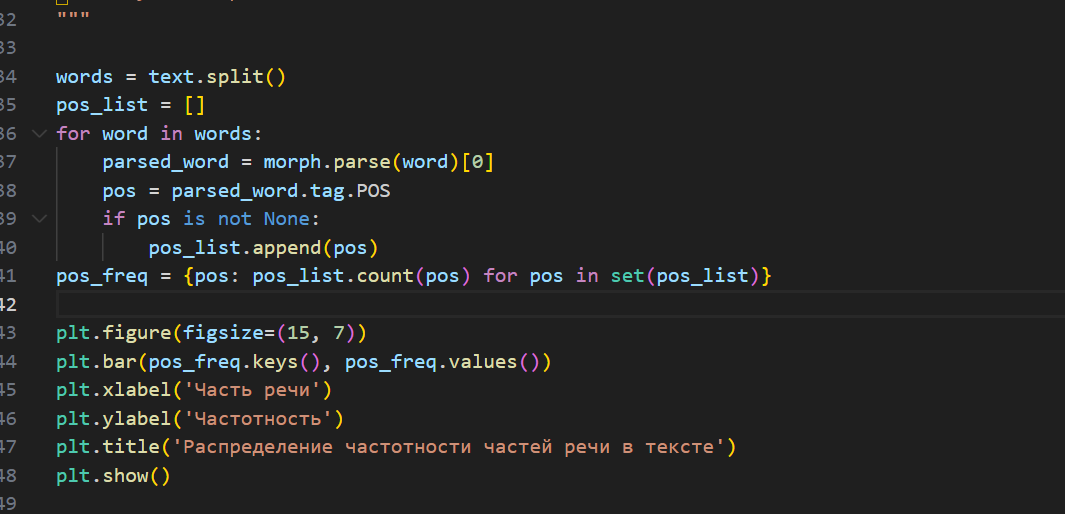
- определить грамматические характеристики слов при помощи pymorphy2

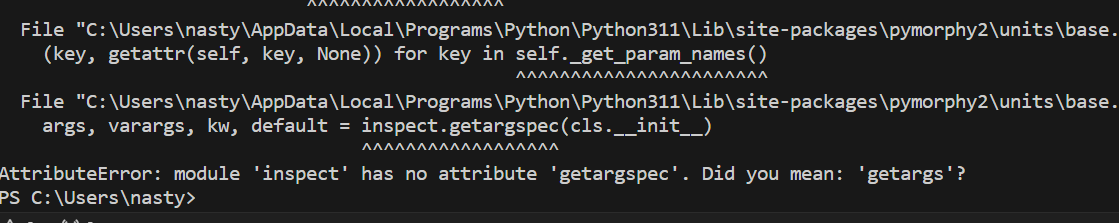




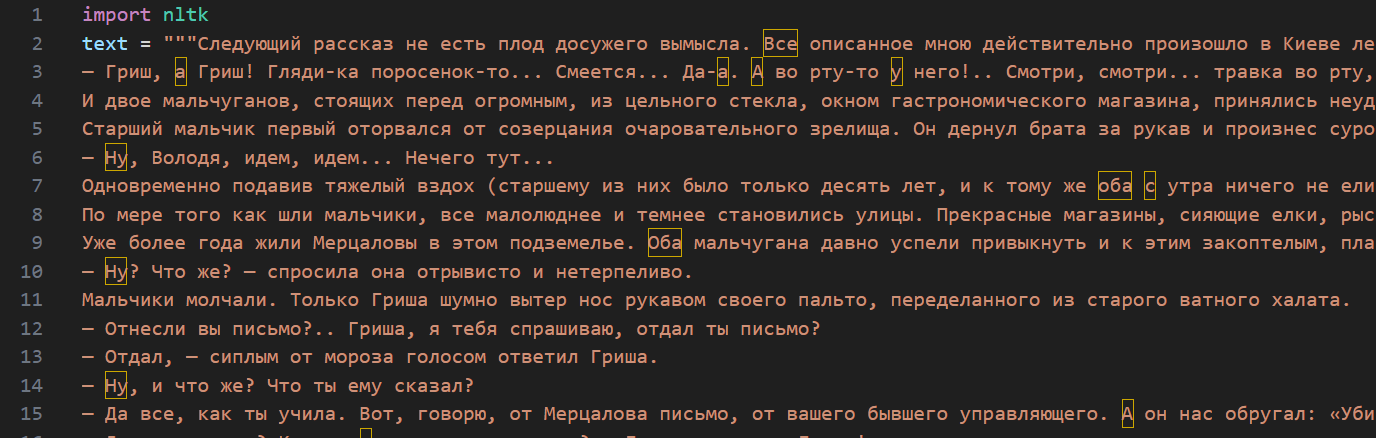
  
- построить график, отражающий распределение частотности частей речи в тексте, при помощи matplotlib

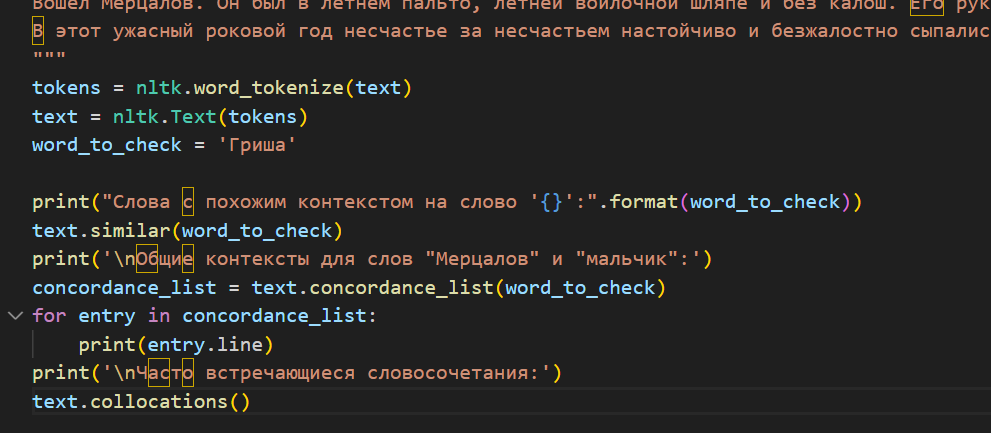


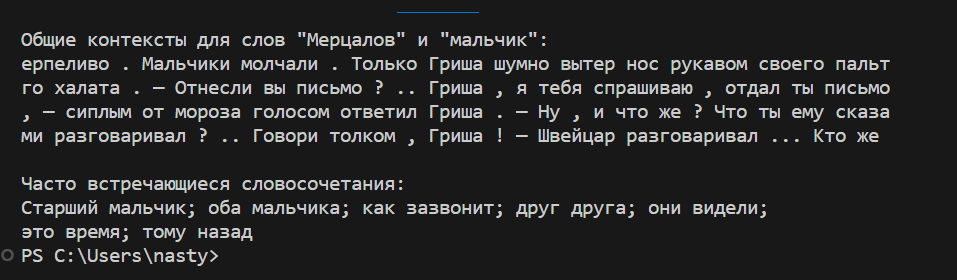




- протестировать методы similar, common\_contexts и collocations из библиотеки NLTK







? для чего эти методы можно использовать?

1. Метод similar(): этот метод позволяет найти слова, которые встречаются в похожих контекстах в тексте. Например, если вы используете метод similar('слово'), он вернет список слов, которые встречаются в сходных контекстах с указанным словом.

2. Метод common\_contexts(): этот метод позволяет найти общие контексты для двух или более слов. Он выводит контексты, в которых указанные слова встречаются вместе. Например, common\_contexts(['слово1', 'слово2']) вернет контексты, в которых оба слова появляются.

3. Метод collocations(): этот метод позволяет найти коллокации в тексте, то есть пары слов, которые часто встречаются вместе. Например, он может найти такие пары, как "быстро бежать" или "горячий чай".