***Техническая документация по генератору упражнений для русского языка.***

**Содержание**

[Используемые библиотеки: 2](#_heading=h.gjdgxs)

[Описание работы кода программы: 3](#_heading=h.30j0zll)

[1 этап: загрузка и обработка коллекции текстов (class TextProcessor) 3](#_heading=h.1fob9te)

[2 этап: генерация упражнений (class Generator) 5](#_heading=h.3znysh7)

[3 этап: сохранение упражнений и ответов в текстовые файлы формата .docx (MS Word) (Class Storage) 9](#_heading=h.2et92p0)

[Пример результата выполнения программы: 12](#_heading=h.tyjcwt)

Генератор упражнений по коллекции русских текстов – это функционал, способный принимать на вход определенную коллекцию текстов на русском языке, обрабатывать ее и автоматически генерировать упражнения в удобном для использования и печати формате .docx (MS Word). Подобный функционал может быть использован в обучающих целях, например, при изучении русского языка детьми в школах России, а также при обучении иностранцев русскому языку.

# **Используемые библиотеки:**

**1) Модуль re** – функционал, позволяющий обрабатывать регулярные выражения – специальную последовательность символов, которая помогает сопоставлять или находить строки Python с использованием специализированного синтаксиса, содержащегося в шаблоне.

* re.split(pattern, string) - делит строку (string) по появлению шаблона регулярного выражения pattern и возвращает список получившихся подстрок.

**2) Модуль random** - предоставляет функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

* random.shuffle(sequence) - перемешивает последовательность (изменяется сама последовательность).
* random.randint(A, B) – выбирает случайное целое число N, A ≤ N ≤ B.
* random.sample(population, k) - создает список длиной k из последовательности population.

**3) Модуль Docx** - предназначен для создания и обновления файлов с расширением .docx - Microsoft Word.

* document = Document() – создание документа
* doc = document.styles – задает стиль документа
* doc.front.name – задает шрифт документа
* doc.font.size – задает размер шрифта документа
* doc.add\_paragraph() – добавление текста в документ
* table = doc.add\_table(rows = , cols =) – добавление таблицы в документ
* table.cell – доступ к ячейке таблицы
* table.cells – доступ к строке таблицы
* table.cell.text – запись текста в ячейку таблицы
* row = table.add\_row – добавление строки к таблице
* doc.save() – сохранение текстового файла

**4) Библиотека Pymorphy2** - морфологический анализатор для русского языка, написанный на языке Python и использующий словари из [OpenCorpora](http://opencorpora.org/).

* В pymorphy2 для морфологического анализа слов (русских) есть класс MorphAnalyzer().

morph = pymorphy2.MorphAnalyzer()

* Метод MorphAnalyzer().parse() - принимает слово (обязательно в нижнем регистре) и возвращает все возможные разборы слова

morph.parse('стали')

[Parse(word='стали', tag=OpencorporaTag('VERB,perf,intr plur,past,indc'), normal\_form='стать', para\_id=879, idx=4, estimate=1.0),

* Атрибут normal\_form – соответствует нормальной форме слова.
* Tag - набор [*граммем*](https://pymorphy2.readthedocs.io/en/0.2/glossary.html#term-4), характеризующих данное слово. Например, тег 'VERB,perf,intr plur,past,indc' означает, что слово - глагол (VERB) совершенного вида (perf), непереходный (intr), множественного числа (plur), прошедшего времени (past), изъявительного наклонения (indc).

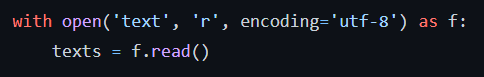
**5) Библиотека nltk** - предоставляет множество полезных функций для обработки текстов, включая токенизацию, выделение корней, теги, синтаксический анализ и многие другие вещи, необходимые для создания моделей машинного обучения для любого приложения обработки естественного языка.

* tokens = nltk.word\_tokenize(sentence) – токенизация предложения на слова.

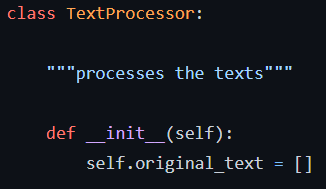
# **Описание работы кода программы:**

## **1 этап: загрузка и обработка коллекции текстов (class TextProcessor)**

1) Коллекция, которая представляет собой несколько текстов, разделенных между собой символом “#” и новой строкой, загружается с помощью метода with open (file).



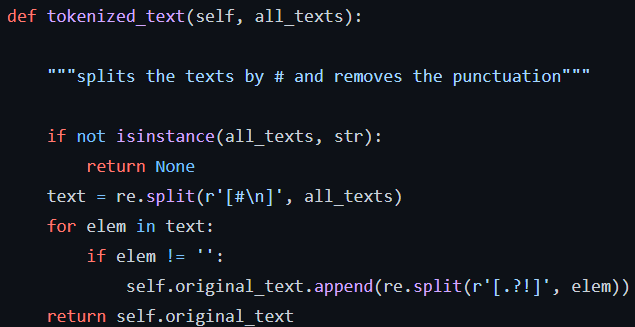
2) Класс TextProcessor - отвечает за предобработку коллекции текстов.



*Метод tokenized\_text (self, all\_texts):*

**Принимает на вход**: коллекция текстов на русском языке (type: str)  
**Выход**: список текстов, разбитых на предложения. (type: lst)

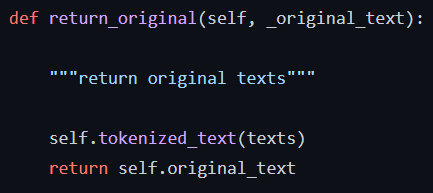
* Тексты разделяются с помощью использования регулярного выражения (re.split()) по символу “#”. Каждый текст начинается с новой строки.
* Далее каждый текст разбивается на предложения (по разделителям – символам окончания предложения, а именно точкам, восклицательным и вопросительным знакам).
* Тексты, разбитые на предложения, записываются в список, переданный в конструкторе класса (self.original\_text)



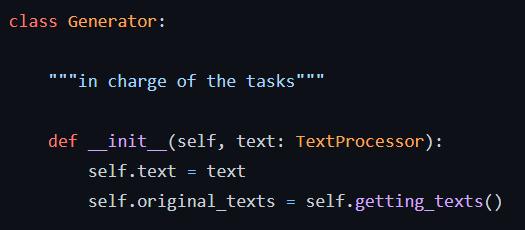
*Метод return\_original (self, \_original\_text):*

**Принимает на вход**: коллекция текстов на русском языке (type: str)  
**Выход**: список текстов, разбитых на предложения (type: lst)

* С помощью метода *tokenized\_text (self, all\_texts)* обрабатывает тексты, считанные из загруженного ранее текстового файла.

****

## **2 этап: генерация упражнений (class Generator)**

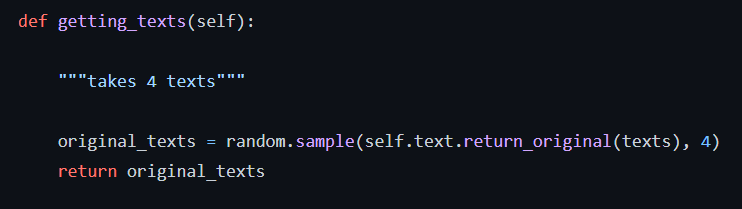
****

1) Выбор отдельного текста для каждого упражнения

*Метод getting\_texts (self):*

**Выход**: список из 4 случайно отобранных текстов (type: lst)

* Из списка с предобработанными текстами случайный образом выбираются 4 текста



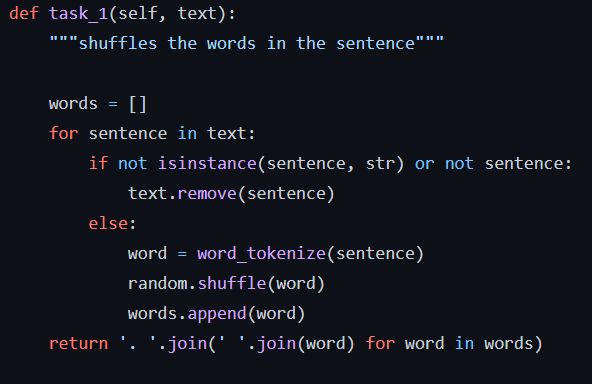
2) Упражнение №1.

**Задание: поставить слова в тексте в правильном порядке**

*Метод task\_1 (self, text):*

**Принимает на вход**: текст, токенизированный на предложения (type: lst)  
**Выход:** тот же текст, но слова в нем перемешаны (type: str)

* С помощью цикла программа проходится по предложениям выбранного для данного упражнения текста и токенизирует каждое предложение текста на слова (для этого используется метод word\_tokenize () библиотеки NLTK)
* Слова в каждом предложении перемешиваются с помощью метода random.shuffle() и добавляются в новый список words.
* C помощью метода ‘ ’.join() готовый список с перемешанными слова преобразовывается в строку и возвращается в методе task\_1 класса Generator.



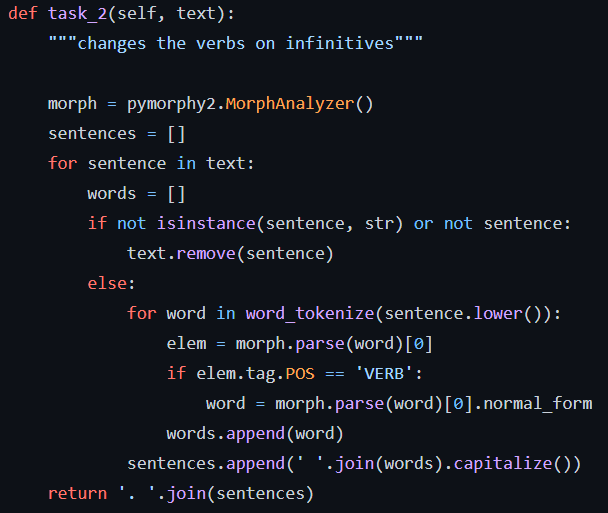
3) Упражнение №2

**Задание:** в заданном тексте поставить глаголу в подходящую грамматическую форму, расставить знаки препинания.

*Метод task\_2(self, text):*

**Принимает на вход**: текст, токенизированный на предложения (type: lst)  
**Выход:** тот же текст, но глаголы в нем стоят в форме инфинитива (type: str)

* С помощью цикла программа проходится по предложениям выбранного для данного упражнения текста и токенизирует каждое предложение текста на слова (для этого используется метод word\_tokenize () библиотеки nltk). Слова также приводятся в нижний регистр с помощью строкового метода s.lower().
* Для каждого слова в каждом предложении выполняется морфологический анализ (Для этого используется метод morph.parse() класса MorphAnalyzer() библиотеки pymorphy2).
* В результате разбора с помощью тэга (elem.tag.POS) – набора граммем, характеризующих слово, мы можем определить к какой части речи оно принадлежит.
* Если слово является глаголом (То есть первым элементов тэга является VERB), то в выбранном для данного упражнения тексте оно ставится в начальную форму (normal\_form).
* Слова и предложения преобразовываются в строку и итоговый текст возвращается данным методом.



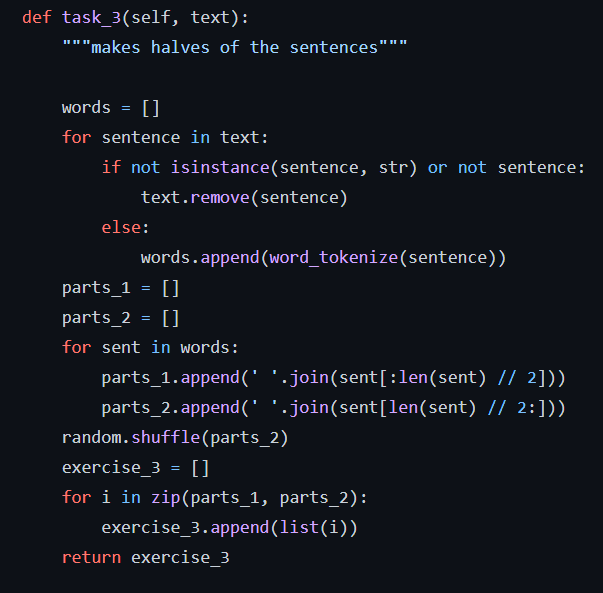
4) Упражнение №3.

**Задание:** правильно соединить половины предложения, исходя из грамматических показателей и контекста.

*Метод task\_3(self, text):*

**Принимает на вход:** текст, токенизированный на предложения (type: lst) **Выход:** список кортежей, где первый элемент кортежа – первая половина предложения, второй элемент – вторая половина предложения) (type: lst)

* С помощью цикла программа проходится по предложениям выбранного для данного упражнения текста и токенизирует каждое предложение текста на слова (для этого используется метод word\_tokenize () библиотеки nltk).
* Каждое предложение делится напополам (берется срез каждого списка, содержащего токенизированные предложения текста). Первые половины добавляются в список parts\_1, вторые – в список parts\_2.
* Элементы списка parts\_2 перемешиваются (random.shuffle()). Элементы списка parts\_1 остаются в правильном порядке
* В итоговый список exercise\_3 добавляются собранные в кортежи элементы списков parts\_1 и parts\_2 (zip).
* Для удобства кортежи преобразовываются в списки.



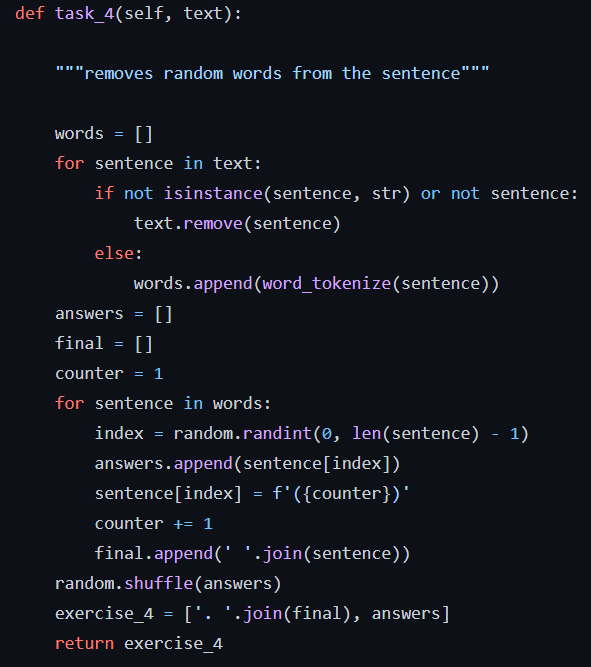
5) Упражнение №4.

**Задание:** правильно вставить пропущенные слова в предложения, основываясь на грамматических показателях и контексте.

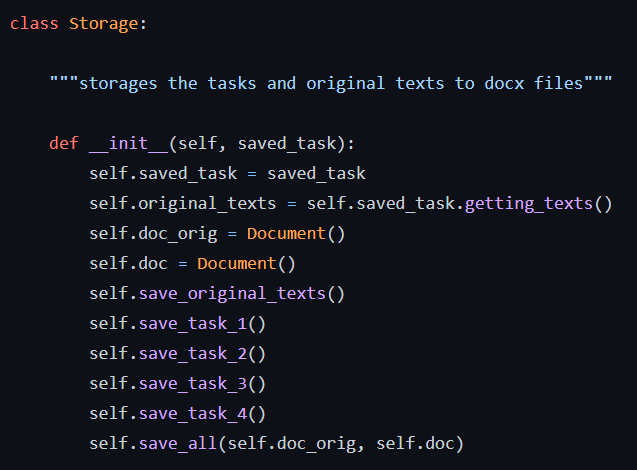
*Метод task\_4(self, text):*

**Принимает на вход:** текст, токенизированный на предложения (type: lst) **Выход:** список, содержащий тот же текст, но в каждом предложении пропущено по одному слову (type: str), каждый пропуск заменен на его порядковый номер, и также содержащий список с удаленными словами (ответами). (type: lst)

* С помощью цикла программа проходится по предложениям выбранного для данного упражнения текста и токенизирует каждое предложение текста на слова (для этого используется метод word\_tokenize () библиотеки nltk).
* В пределах каждого предложения случайным образом (random.randint()) выбирается число, не большее, чем [количество слов в предложении -1]. Элемент (слово) с индексом, равным этому числу, удаляется из предложения и добавляется в список с ответам (answers). На месте этого слова в предложении ставится порядковый номер, который увеличивается на один с каждым предложением.
* Ответы в списке answers перемешиваются (random.shuffle())
* В итоговый список exercise\_4 добавляются текст, с замененными на порядковый номер пропущенными словами и список с ответами.



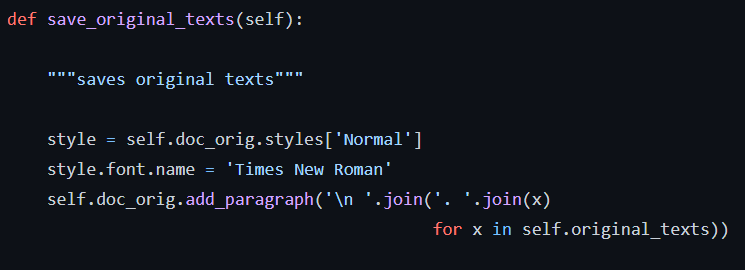
## **3 этап: сохранение упражнений и ответов в текстовые файлы формата .docx (MS Word) (Class Storage)**



1) Сохранение оригинальных текстов

*Метод save\_original\_texts(self):*

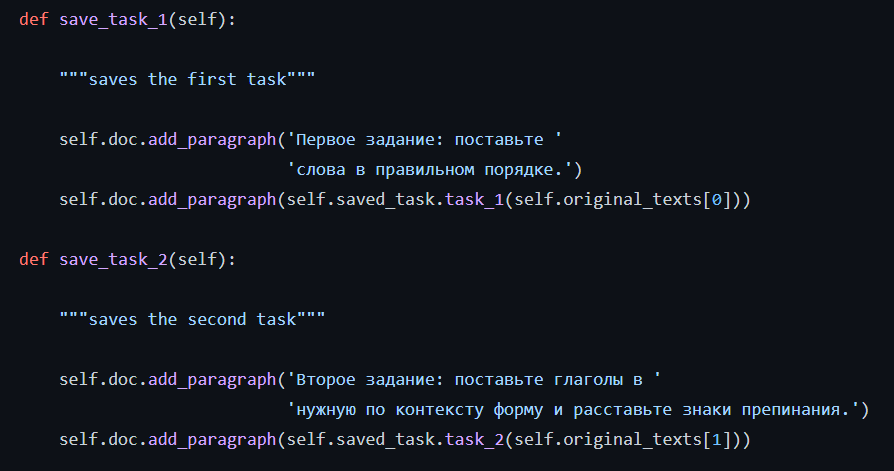
* Добавляет в документ формата .docx все тексты, которые использовались при генерации упражнений.
* Устанавливается шрифт текстов (Times New Roman) и стиль.



2) Сохранение упражнений №1 и №2.

*Методы save\_task\_1(self) и save\_task\_2(self):*

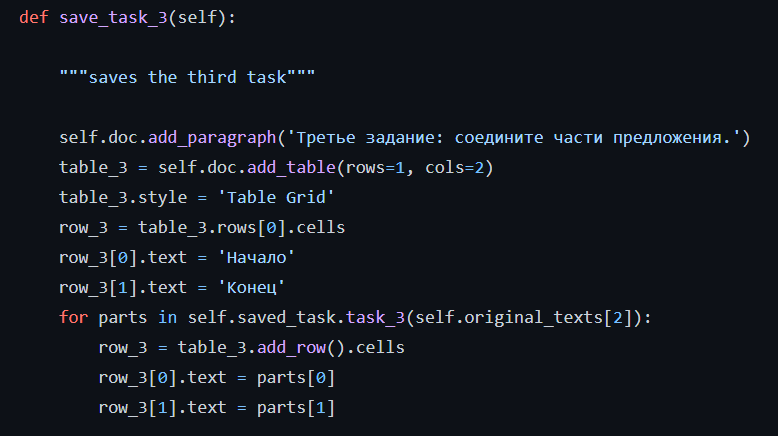
* Добавляет в итоговый документ формата .docx первые два упражнения.
* Прописывается само задание, которое нужно выполнить тестируемому.
* В качестве первого упражнения записывается результат выполнения метода класса Generator task\_1, второго – task\_2 соответственно.



3) Сохранение упражнения №3.

*Метод save\_task\_3(self):*

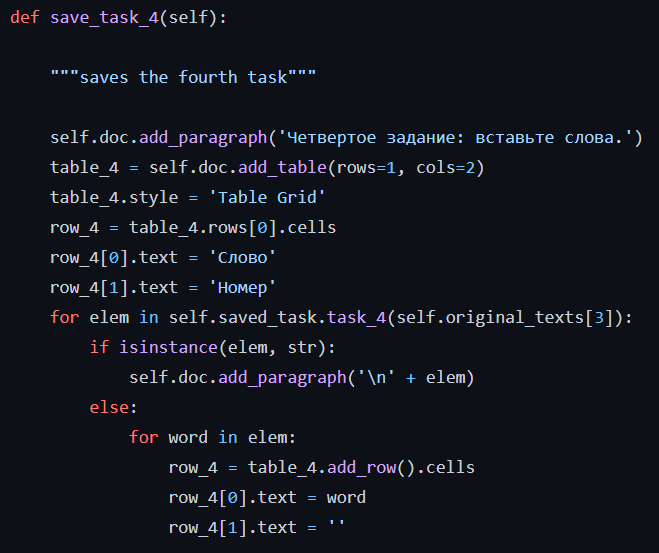
* Добавляет в итоговый документ формата .docx третье упражнение.
* Создаются таблица (add\_table), содержащая 2 столбца и одну первую строку, устанавливается ее стиль (style)
* В первую строку добавляются наименования столбцов (Начало, Конец)
* В созданную таблицу добавляются новые строки (add\_row), которые заполняются данными результат выполнения метода класса Generator task\_3. В первый столбец каждой строки записываются первые половины предложения (элемент кортежа parts с индексом 0, так как первым элементом каждого кортежа являются элементы списка с первыми половинами (parts\_1)), во второй – вторые (элемент кортежа parts с индексом 1)



4) Cохранение упражнения №4.

*Метод save\_task\_4(self):*

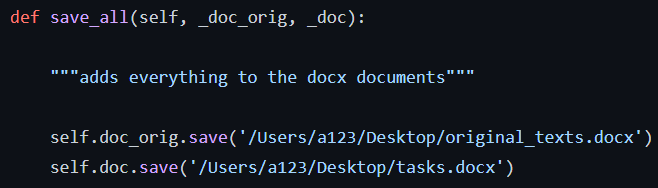
* Добавляет в итоговый документ формата .docx четвертое упражнение.
* Создаются таблица (add\_table), содержащая 2 столбца и одну первую строку, устанавливается ее стиль (style)
* В первую строку добавляются наименования столбцов (Слово, Номер)
* В созданную таблицу добавляются новые строки (add\_row), которые заполняются данными результат выполнения метода класса Generator task\_4. В первый столбец каждой строки слова из списка с ответами answers. Второй столбец остается пустым (в него тестируемый должен вставить правильный порядковый номер)



5) Сохранение всей работы в единый итоговый документ формата .docx

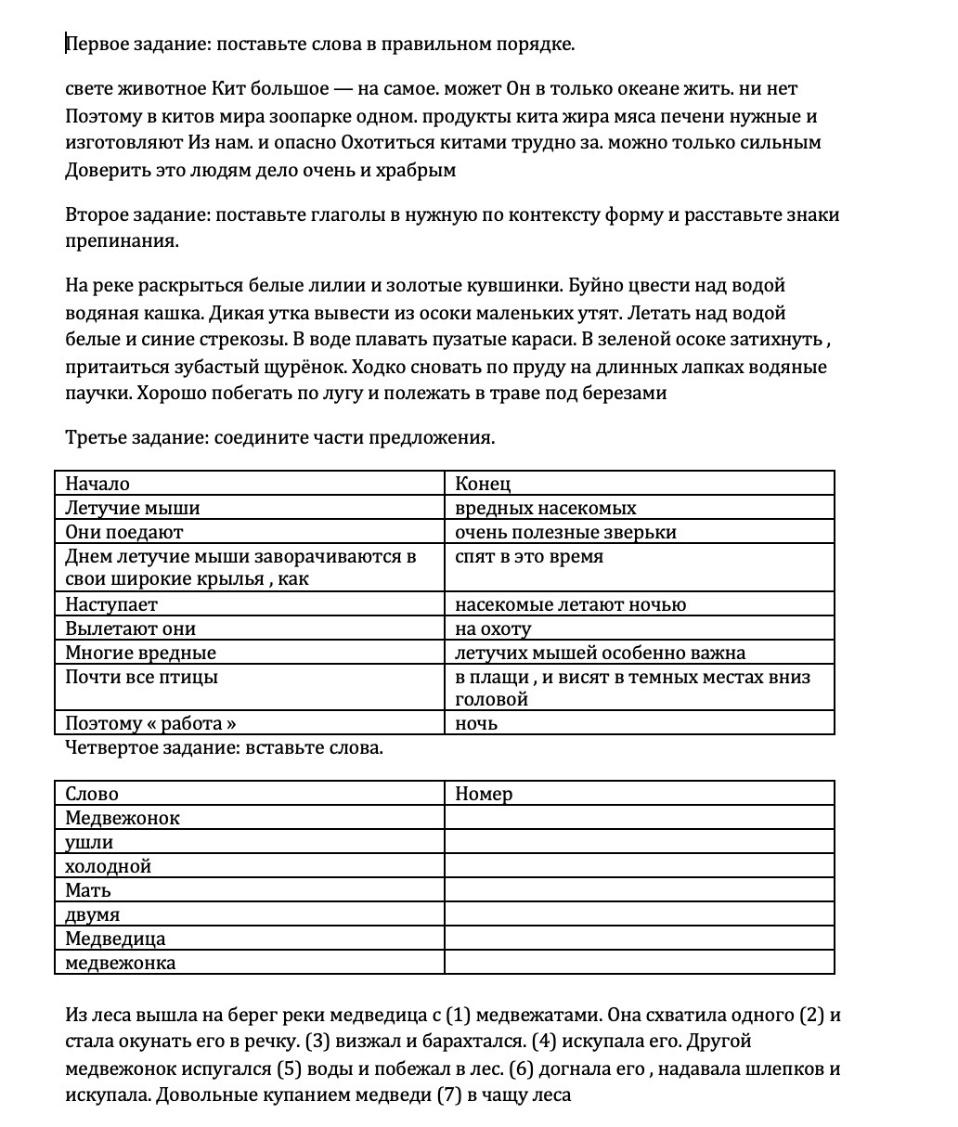
*Метод save\_all(self, \_doc\_orig, \_doc):*

* Создает в указанной директории два файла: один с оригинальными текстами, использованными в генерации упражнений, другой – с самими упражнениями.



# **Пример результата выполнения программы:**

1. Упражнения



1. Тексты

