**СОПРОВОДИТЕЛЬНЫЙ ДОКУМЕНТ**

к программному средству

Оглавление

[Назначение 3](#_Toc91704142)

[Модули 3](#_Toc91704143)

[WSNews.py 3](#_Toc91704144)

[WebsiteShaker.py 3](#_Toc91704145)

[MyHTMLParser.py 5](#_Toc91704146)

[Примеры работы программы 6](#_Toc91704147)

[Пример 1. 6](#_Toc91704148)

[Пример 2. 6](#_Toc91704149)

[Направления дальнейшего улучшения/доработки 7](#_Toc91704150)

[1. Универсальность и унификация 7](#_Toc91704151)

[2. Расширение функционала и украшение 8](#_Toc91704152)

[3. Доработка тестового модуля 8](#_Toc91704153)

# Назначение

Анализ веб-страниц и получение «полезной» текстовой информации.

# Модули

## WSNews.py

Является главным модулем, реализующим обработку атрибутов утилиты командной строки.

## WebsiteShaker.py

Основной модуль, реализующий логику «парсинга» веб-страницы. Содержит класс *NewsObject*.

В качестве полей класса выступают объекты, предназначенные для хранения информации о сайте, откуда выбираются данные, и непосредственно полученного с сайта текста:

|  |  |
| --- | --- |
| Переменная | Назначение |
| url <str> | адрес страницы |
| title <str> | название статьи |
| text <list> | блоки текста, взятые со страницы |
| references <list> | список ссылок, найденных по тексту |
| settings <dict> | настройки программы |

Класс содержит 8 методов:

|  |  |
| --- | --- |
| Метод | Назначение |
| \_\_init\_\_() | конструктор |
| shake\_site(url) | public метод, запускающий обработку страницы, в качестве атрибута получает адрес |
| create\_raw\_html(url) | public метод, возвращает «сырой» HTML-код страницы |
| \_get\_text() | Находит и сохраняет в соответствующее поле класса блоки текста, название статьи и ссылки |
| \_format\_text() | Производит форматирование блоков текста, согласно требованиям |
| \_do\_the\_textfile() | Производит запись текста в файл |
| \_\_settings\_load() | Производит загрузку настоек из файла |
| settings\_change() | public метод, позволяет производить смену настроек путем вызова утилиты |

**Конструктор *\_\_init\_\_()***

Стандартный метод класса, инициализирует все поля «пустыми», за исключением *settings*, содержащий значения по умолчанию.

**Метод *shake\_site (url)***

Настоящий метод передает значение URL экземпляру класса и активирует процесс обработки страницы и генерации выходного файла.

Запускает последовательно следующие методы.

**Метод *\_settings\_load()***

Подгружает (десериализует) с помощью json словарь с настройками. Также отлавливает исключения. Поскольку конструктор класса проинициализировал настройки значениями по умолчанию, при обнаружении нарушений в структуре файла (например, после вмешательства вручную), позволяет их игнорировать. Сообщает пользователю о количестве ошибок и о том, что значения берутся по умолчанию.

**Метод  *\_get\_text ()***

Реализует логику поиска текста на сайте.

На данный момент алгоритм абсолютно точно работает на сайте lenta.ru.

Блоки текста находятся внутри тегов <p></p>. При этом для текущего сайта текстовые блоки имеют свой особый класс. Метод проверяет на соответствие класса блока классу основного теста – «topic-body\_\_content-text» или заголовка статьи – «topic-header\_\_title».

Внутри каждого текстового блока происходит проверка вхождения блока ссылки, заключенного в теги <a></a>. В случае обнаружения, в текст, взятый из блока <р>, после слова, содержащего ссылку, добавляется сноска [n] с порядковым номером ссылки, а адрес ссылки добавляется в список ссылок класса на соответствующую номеру позицию. Реализована система примечаний по той причине, что читать текст с длинной (порой в несколько строк) ссылкой посреди абзаца – неудобно.

**Метод *\_format\_text()***

Метод «подготавливает» полученные блоки текста к записи в файл, придавая им желаемый вид.

В данном случае играют роль настройки: «длина строки» и «считать пробелы при вычислении длины строки», значения которых берутся из словаря *settings* по соответствующим ключам. Второй параметр добавлен исключительно в «декоративных» целях и не является критическим.

Каждый блок текста разбивается на слова – функция *split()*. С помощью простого перебора в полученном списке слов вычисляется длина каждого слова, сумма длин слов, а сами слова добавляются к строке. Простыми словами говоря, суммируются длины слов и сами слова. Если суммарная длина слов превышает заданную, добавляем в строку символ переноса, обнуляем счетчик длины строки.

Полученные блоки текста готовы к записи в файл.

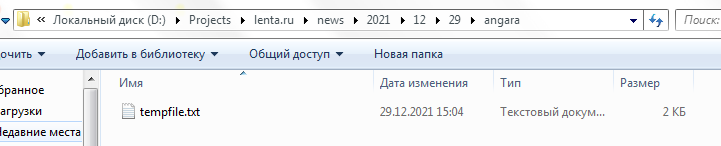
**Метод  *\_do \_the\_textfile()***

«Метод мне следовало назвать «\_create \_the\_textfile», ибо корректнее» – прим. автора

Указанный метод с использованием модуля OS формирует полный путь на основании URL страницы. Полный путь позволяет избежать некоторых возможных проблем с определением путей.

Выходной файл изначально создается в корневом каталоге. Записывается в файл title, затем блоки текста и в конце файла список ссылок по тексту с номерами примечаний. Также в данном методе производится минимальное «косметическое» форматирование выходного текста.

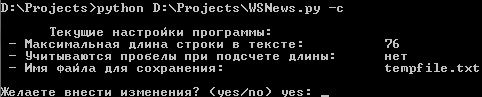
Готовый файл сохраняется и переносится в сформированную по URL директорию.



**Метод *settings\_change()***

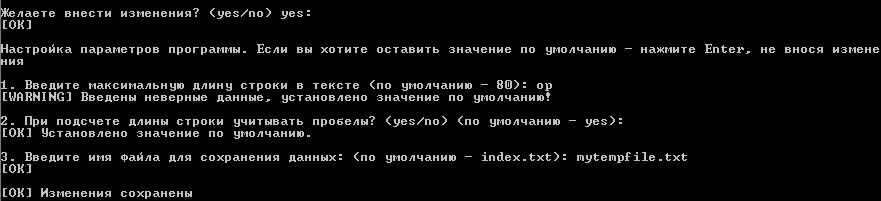
Метод разработан с целью оптимизации процесса ввода настроек для пользователя, позволяя хранить данные настроек в том виде, в котором программе «проще» их воспринимать, исключая необходимость для пользователя разбираться со структурой файла.

Также вызов метода отображает текущие настойки и позволяет пользователю оставить все, как есть.



Метод реализует логику проверки всех вводимых данных на соответствие типам данных, на предмет пустых значений и т.д., «отлавливая» основные исключения, как, например, ошибки открытия файла настроек или получения из него информации.

Метод отвечает на все действия пользователя, позволяя контролировать процесс ввода данных.



По завершению работы сохраняет словарь с настойками в файл *settings.cfg*.

**Метод *create\_raw\_html(url)***

Получает код страницы с использованием Beautiful Soup и requests для дальнейшей ее обработки.

## MyHTMLParser.py

Тестовый модуль. Содержит класс WebRageParser, наследующий класс HTMLParser, и включает перегруженные методы родительского класса.

Данный модуль – попытка решения поставленной задачи без применения сторонних модулей. Указанный супер-класс позволяет быстро находить блоки, ограниченные определенными тегами, определять открывающий и закрывающий тег, а также данные между ними. За счет перегрузки методов получается без труда находить блоки с текстом и ссылки.

Проблема данной реализации в том, что стандартный метод *handle\_data(self, data)* возвращает содержимое найденного блока только до следующего тега. То есть из найденной записи <p> Some text <a href=”…”>ref</a> another text </p> указанный метод вернет только «Some text», проигнорировав как «ref», так и «another text».

В целях экономии времени данную реализацию пришлось отложить.

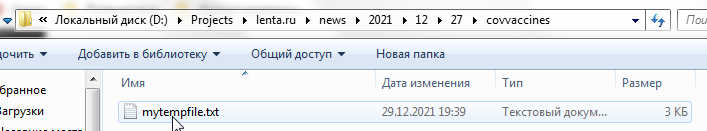
# Примеры работы программы

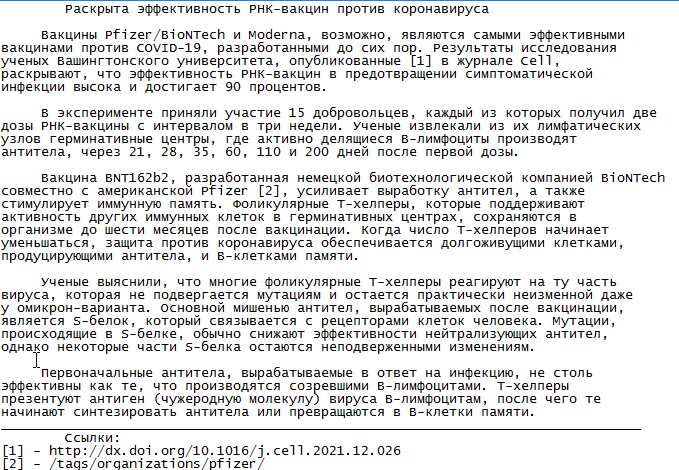
## Пример 1.

Входные данные: url = <https://lenta.ru/news/2021/12/27/covvaccines/>

Результат:





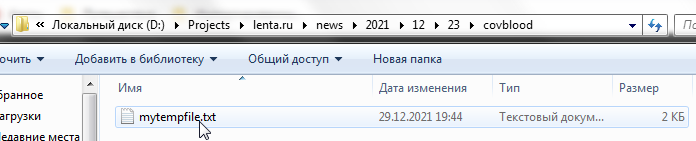


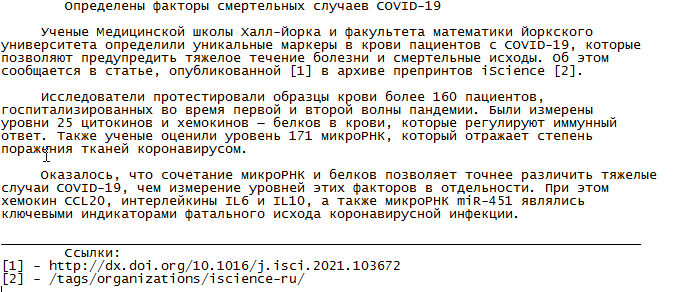
## Пример 2.

Входные данные: url = <https://lenta.ru/news/2021/12/23/covblood/>

Результат:







# Направления дальнейшего улучшения/доработки

## 1. Универсальность и унификация

Указанная реализация работает на сайте lenta.ru. Анализ страниц других сайтов, в том числе новостного сайта gazeta.ru показал, что текстовая информация содержится в блоках <p></p>. Однако заголовки статей имею различные теги, а текст – классы. Для сравнения:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Lenta.ru** | | **Gazeta.ru** | |
|  | **тег** | **класс** | **тег** | **класс** |
| Заголовок статьи | <span> | topic-header\_\_title | <h1> | headline |
| Текст | <p> | topic-body\_\_content-text | <p> | - |

Также у «Газеты» имеются подзаголовки, ограниченные тегом <h2>, которых нет у «Ленты». И так далее.

Все указанные факторы усложняют процесс унификации атрибутов сайтов различных интернет ресурсов, так как даже получение, казалось бы, текста из унифицированного тега <p> может быть связано с трудностями, поскольку данным тегом выделен на странице не только информативный текст, но и иные элементы страницы. Поэтому использование наименования класса в случае с «Лентой» абсолютно оправдано.

Однако по указанному тегу непосредственно текст получить можно, все-таки, с большинства сайтов. Данный алгоритм будет реализован в будущем.

Из возможных вариантов могу выделить следующие:

1. Жестко привязывать логику обработки к конкретному ресурсу. То есть в настройках задавать обрабатываемый ресурс («Лента», «Газета» и т.д.) и из заранее прописанных алгоритмов обработки выбирать соответствующий. Или же позволить программе выбирать самостоятельно (сопоставляя наименование ресурса и вшитые в нее алгоритмы)
2. Создать универсальный алгоритм, который:

- будет получать весть текст <p> со страницы и проводить его семантический разбор. В первую очередь, проверять его «родительский» тег. При этом стоит учесть, что если та же «Газета» строит на нем только информативные блоки, то «Лента», например, может им злоупотреблять.

## 2. Расширение функционала и украшение

Представление «выгруженной» статьи в .doc файле с соответствующим форматированием, гиперссылками на интернет ресурсы, а также со вставкой изображений, относящихся к статье.

Так же саму статью в файл можно записывать с использованием либо самого HTML форматирования, либо XML.

Добавление графической оболочки с непосредственным выбором параметров парснга.

## 3. Доработка тестового модуля

Немного о тестовом модуле. Обойти проблему с HTMLParser’ом для обеспечения абсолютной «нативности». Практической ценности не имеет, так как с помощью готовых сторонних модулей гораздо проще и быстрее решается задача. Исключительно спортивный интерес.