СОДЕРЖАНИЕ

[Описание алгоритмов в разработанном модуле Mini\_readability 2](#_Toc27073556)

[Класс Readability 3](#_Toc27073557)

[Исходный код скрипта. 4](#_Toc27073558)

[Список url для проверки 8](#_Toc27073559)

[Результаты проверки 8](#_Toc27073560)

[Направление дальнейшего развития 9](#_Toc27073561)

# Описание алгоритмов в разработанном модуле Mini\_readability

1. Получение DOM дерева html страницы по заданному url. Используя библиотеки requests для доступа по url и получение DOM дерева с помощью библиотеки lxml.
2. Обход дерева используя установленные теги. Проанализировав сайты, можно понять, что «основной» текст находиться внутри <body> в тегах <p>, <h1>, <h2>, <h3>, следовательно, нас интересуют именно эти теги. Но все сайты написаны по-разному и разными методами поэтому эти теги будут дополнены см. пункт 5.
3. Если внутри тега текст имеется длинный текст, грабим его. В полученном тексте выделить ссылки в [] скобки. Так как внутри «основного» текста могут находиться ссылки, а это тег <a>, следовательно, получаем их путем доступа к нижнему уровню элемента из библиотеки lxml и метода «.text».
4. Выполнение усложнения 1: из полученного url создается путь с файлом, в него записывается файл, полученный на предыдущем этапе. На этом этапе реализуется создание новых директорий, путем преобразования строк с исходным url методами \_\_get\_cur\_path(link) и \_\_get\_new\_filename(link) описание см.ниже. После происходит запись в файл методом In\_file(str).
5. Выполнение усложнения 2: в текстовом документе в папке вместе с исполняемым файлом хранить дополнительные теги, с которых тоже можно производить парсинг, хранить в виде списка через ‘,’. На этом этапе организован файл под названием more\_tag.txt если он есть, то полученные оттуда теги будут переданы в метод find\_text(el, more\_tag) и по данным тегам также будет проводиться парсинг.

# Класс Readability

Конструктор:

* Readability(url), где url – ссылка на сайт.
* \_\_main\_domain\_stat = url
* \_\_response = requests.get(self.\_\_main\_domain\_stat)
* root = html.parse(StringIO(self.\_\_response.text)).getroot()

Методы:

try\_response(url) – проверка url на ошибки страницы (true-успешно, false-ошибка доступа, либо стр не существует)

find\_text(el, more\_tag) – el – элемент типа lxml.html.HtmlElement, more\_tag – список дополнительных тегов. Возвращает текст со страницы.

In\_file(str) – записывает str – текст в файл и создает путь к нему из текущей директории путем преобразования \_\_main\_domain\_stat.

\_\_internal\_link(main\_link, internal\_tag) – получение абсолютной ссылки, так как на некоторых сайтах используются не полные ссылки. Делает слияние main\_link и internal\_tag и возвращает строку.

\_\_get\_cur\_path(link) – обрабатывает входной link и возвращает строку - новый путь к создаваемой директории.

\_\_get\_new\_filename(link) – обрабатывает входной link и возвращает строку с именем файла (заменяет .html или другие расширения на .txt и сокращает только до короткого имени) возвращает строку с именем файла.

# Исходный код скрипта.

|  |
| --- |
| import requests  from lxml import html  from io import StringIO  import textwrap  import os  class Readability:  \_\_STANDART\_TAG = ['p', 'h1', 'h2', 'h3']  def \_\_init\_\_(self, url):  self.\_\_main\_domain\_stat = url  self.\_\_response = requests.get(self.\_\_main\_domain\_stat)  self.root = html.parse(StringIO(self.\_\_response.text)).getroot()  def try\_response(url):  RES = requests.get(url)  if RES:  print('Response OK')  return True  else:  print('Response Failed:', RES.status\_code)  return False  def \_\_internal\_link(self, main\_link, internal\_tag):  if internal\_tag.find("http") != -1:  return main\_link  FIRST\_IN = main\_link.find('.ru') + 3  if FIRST\_IN == 0:  FIRST\_IN = main\_link.find('.com') + 4  OPTIMAL\_LINK = "%s%s" % (main\_link[0:FIRST\_IN], internal\_tag)  return OPTIMAL\_LINK  def find\_text(self, el, more\_tag):  if el.tag in ['footer', 'header'] or el.attrib == {'class': 'right\_preview\_text'}:  return ""  ALL\_TEXT = ""  if el.text is not None:  if len(el.text) > 25 and el.tag in self.\_\_STANDART\_TAG+more\_tag:  TMP = el.text\_content()  if len(el) > 0:  for i in range(len(el)):  if el[i].tag == 'a':  if el[i].text is None:  return ""  LINK\_AND\_TXT = "%s[%s]" % (el[i].text, self.\_\_internal\_link(self.\_\_main\_domain\_stat, el[i].get('href')))  LEN\_TXT = len(el[i].text\_content())  FIRST\_IN = TMP.find(el[i].text)  TMP = "%s%s%s" % (TMP[0:FIRST\_IN], LINK\_AND\_TXT, TMP[FIRST\_IN + LEN\_TXT:len(TMP)])  # if el[i].tag =='br':  TMP = textwrap.fill(TMP, width=80)  TMP += "\n\n"  return TMP  for child in el:  ALL\_TEXT += self.find\_text(child, more\_tag)  return ALL\_TEXT  def \_\_get\_cur\_path(self, link):  EXEC\_PATH = os.getcwd()  if link.find('http://') != -1:  FIRST\_IN = 6  elif link.find('https://') != -1:  FIRST\_IN = 7  else:  FIRST\_IN = 0  LAST\_SLASH\_IN = link.rfind('/')  TMP = os.path.normpath(link[FIRST\_IN:LAST\_SLASH\_IN])  NEW\_DIR\_PATH = "%s%s" % (EXEC\_PATH, TMP)  return NEW\_DIR\_PATH  def \_\_get\_new\_filename(self, link):  LAST\_SLASH\_IN = link.rfind('/') + 1  if link.count('.') > 1:  return link[LAST\_SLASH\_IN:link.rfind('.')] + '.txt'  else:  return 'index.txt'  def in\_file(self, str):  NEW\_PATH = self.\_\_get\_cur\_path(self.\_\_main\_domain\_stat)  FILENAME = self.\_\_get\_new\_filename(self.\_\_main\_domain\_stat)  if not os.path.exists(NEW\_PATH):  os.makedirs(NEW\_PATH)  FILE = open(os.path.join(NEW\_PATH, FILENAME), 'w')  FILE.write(str)  FILE.close()  def main():  while True:  URL = input('input url:')  try:  if Readability.try\_response(URL):  break  print('Please, enter correct url again')  except requests.exceptions.MissingSchema:  print('Please, enter correct url again')  except requests.exceptions.ConnectionError:  print('Seems like dns lookup failed..')  MORE\_TAG = []  try:  FILE = open('more\_tag.txt', 'r')  MORE\_TAG = FILE.readline().split(',')  print('you use file more\_tag.txt')  FILE.close()  except FileNotFoundError:  print('you do not use file with more tags (more\_tag.txt)')  RDA = Readability(URL)  TEXT = RDA.find\_text(RDA.root, MORE\_TAG)  RDA.in\_file(TEXT)  print('compleat')  if \_\_name\_\_ == "\_\_main\_\_":  main() |

# Список url для проверки

https://lenta.ru/articles/2019/12/12/promfest/ (запуск без файла more\_tag.txt просто его не должно быть в текущей папке с исходным файлом)

https://lenta.ru/news/2019/12/12/pojar\_vse/ (запуск без файла more\_tag.txt)

https://www.gazeta.ru/politics/2019/12/12\_a\_12860060.shtml?updated (использовался файл more\_tag.txt с тегами <i>, <b>, <span>, <div>)

https://www.gazeta.ru/science/2019/10/30\_a\_12786044.shtml (использовался файл more\_tag.txt с тегами <i>, <b>, <span>, <div>)

# Результаты проверки

1. Запустить файл Mini\_readability.ext
2. Ввести url нажать «enter»

В результате выполнения исполняемого файла получаем в текущей папке директории, представленные на рисунке 1.

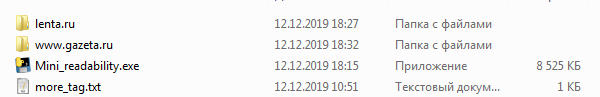


Рисунок 1 – «Результат выполнения»

Если пройдем в первую директорию по пути [CUR DIR] lenta.ru\news\2019\12\12\pojar\_vse будет находиться файл index.txt со следующим содержимым продемонстрированным на рисунке 2.

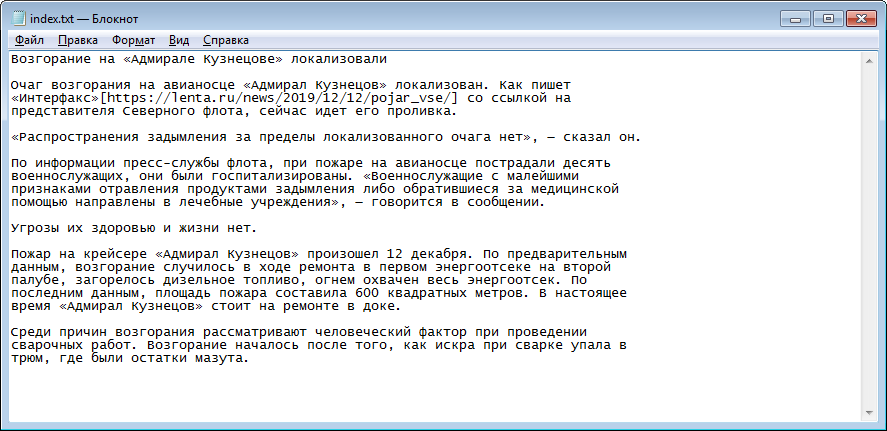


Рисунок 2 – «Содержимое index.txt»

# Направление дальнейшего развития

Реализовать возможность запрещать грабить данные с определенных тегов, которые укажет пользователь, а также реализовать выбор нужных тегов не только по тегу, а также и по атрибутам тега, организуя тем самым более тонкую настройку под любой сайт