Модуль python-markdown в Python

1. Сторонние пакеты и модули Python3.
2. Модуль python-markdown в Python

Преобразование разметки Markdown в HTML

Модуль `markdown` представляет собой синтаксический анализатор разметки Markdown, написанный на языке Python. Он почти полностью соответствует эталонной реализация Markdown Джона Грубера, хотя есть несколько очень незначительных отличий. Правила синтаксиса смотрите в документации "Синтаксис Markdown".

Установка модуля `markdown` в виртуальное окружение:

```
# создаем виртуальное окружение, если нет
$ python3 -m venv .venv --prompt VirtualEnv
# активируем виртуальное окружение
$ source .venv/bin/activate
# ставим модуль markdown
(VirtualEnv) Idea@Centre:~$ python -m pip install -U markdown
```

Прежде всего, проект `Python-Markdown` задуман как модуль библиотеки Python, используемый различными проектами для преобразования синтаксиса Markdown в HTML.

Пример простого преобразования разметки Markdown в HTML:

```
>>> import markdown
>>> text = """`Python-Markdown` предоставляет *две* общедоступные
функции _`markdown.markdown()`_ и **`markdown.markdownFromFile()`**"""
>>> html = markdown.markdown(text)
>>> html
# '<p><code>Python-Markdown</code> предоставляет <em>две</em>
# общедоступные функции <em><code>markdown.markdown()</code></em>
# и <strong><code>markdown.markdownFromFile()</code></strong></p>'
```

API модуля `markdown`.

Проект `Python-Markdown` предоставляет две общедоступные функции `markdown.markdown()` и `markdown.markdownFromFile()`](#markdown.markdownFromFile), обе из которых обертывают открытый класс `markdown.Markdown()`.

Если необходимо обрабатывать один документ за раз, то эти функции будут соответствовать вашим потребностям. Однако, если нужно обработать несколько документов за раз, то может быть полезно создать один экземпляр класса `markdown.Markdown()` и обрабатывать через него несколько документов. Если в работе используется один экземпляр, то не забудьте правильно вызвать метод очистки.

Содержание:

***markdown.markdown(text [, **kwargs])*:**

Функция `markdown.markdown()` преобразует разметку Markdown в разметку HTML.

Аргумент `text` - это исходная строка *Юникода*.

Важно. Python-Markdown ожидает ввода строки Unicode и возвращает вывод в виде строки Unicode. Модуль не принимает в качестве исходной строки байтовые строки! Пользователь несет ответственность за ее декодирование. Например:

```
import markdown
with open("some_file.txt", "r", encoding="utf-8") as fp:
    text = fp.read()
html = markdown.markdown(text)
```

Если необходимо записать вывод на диск, то нужно самим его перевести в требуемую кодировку :

```
import markdown
with open("some_file.html", "w", encoding="utf-8",
errors="xmlcharrefreplace") as fp:
    fp.write(html)
```

Аргумент `extensions` представляет собой список расширений модуля `markdown`.

Проект `Python-Markdown` предоставляет API для сторонних разработчиков для написания расширений в парсер, добавляя собственные дополнения или изменения в синтаксис. Несколько часто используемых расширений поставляются с модулем `markdown`.

Список расширений может содержать экземпляры расширений и/или строки имен расширений.

```
extensions=[MyExtClass(), 'myext', 'path.to.my.ext:MyExtClass']
```

Примечание. Предпочтительный метод - передать экземпляр расширения. Строки следует использовать только тогда, когда невозможно напрямую импортировать класс расширения.

При передаче экземпляров расширения каждый экземпляр класса должен быть подклассом `markdown.extensions.Extension`, и любые параметры конфигурации должны быть определены при инициализации экземпляра класса, а не с использованием ключевого слова `extension_configs`. Например:

```
from markdown.extensions import Extension
class MyExtClass(Extension):
    # Определите свое расширение здесь...

markdown.markdown(text, extensions=[MyExtClass(option='value')])
```

Если имя расширения предоставляется в виде строки, то строка должна быть либо зарегистрированной точкой входа любого установленного расширения, либо импортируемым путем с использованием точечной нотации Python.

Просто включите определенное имя в виде строки в список расширений. Например, если расширению присвоено имя `myext` и оно правильно установлено, выполните следующие действия:

```
markdown.markdown(text, extensions=['myext'])
```

Если у расширения нет зарегистрированной точки входа, вместо него можно использовать точечную нотацию Python. Расширение должно быть установлено как модуль Python. Как правило, в названии должен быть указан класс. Класс должен стоять в конце имени и отделяться от модуля двоеточием.

```
markdown.markdown(text, extensions=['path.to.module:MyExtClass'])
```

Если в модуле определено только одно расширение и модуль включает функцию `makeExtension`, которая возвращает экземпляр расширения, то тогда имя класса не требуется. Например, в этом случае можно сделать `extension=['path.to.module']`.

При загрузке расширения по имени (в виде строки) можно передать параметры конфигурации расширению только с помощью ключевого слова `extension_configs`.

Аргумент `extension_configs` представляет собой словарь параметров конфигурации для расширений.

Любые параметры конфигурации будут передаваться только расширениям, загруженным по имени (в виде строки). При загрузке расширений как экземпляров класса передайте параметры конфигурации непосредственно классу при его инициализации.

Примечание. Предпочтительный метод - передать экземпляр расширения, который вообще не требует использования аргумента `extension_configs`.

Словарь параметров конфигурации должен быть в следующем формате:

```
extension_configs = {
    'extension_name_1': {
        'option_1': 'value_1',
        'option_2': 'value_2'
    },
    'extension_name_2': {
        'option_1': 'value_1'
    }
}
```

При указании имени расширения обязательно используйте ту же строку, которая используется в аргументе `extension` для загрузки расширения. В противном случае настройки конфигурации не будут применены к расширению. Другими словами, если не можете использовать точку входа на одном месте и точечную нотацию Python на другом. Хотя оба могут быть действительными для данного расширения, модуль не распознает их как одно и то же.

Смотрите документацию, относящуюся к расширению, которое используете, чтобы узнать, как указать параметры конфигурации для этого расширения.

Аргумент `output_format` представляет собой строку формат вывода. Значения могут быть в нижнем или верхнем регистре.

Поддерживаемые форматы:

- 'xhtml': выводит теги стиля XHTML. Дефолт.
- 'html5': выводит теги стиля HTML.

Аргумент `tab_length` представляет собой длину отступов в исходном тексте для интерпретации этого текста как кода. По умолчанию: 4

`markdown.markdownFromFile(**kwargs)`:

Функция `markdown.markdownFromFile()` читает текстовый файл с разметкой Markdown, преобразует ее в HTML, который записывается в выходной файл.

Функция принимает те же параметры, что и `markdown.markdown`, за исключением аргумента `text` - текстовую строку (или строку Unicode).

Функция `markdown.markdownFromFile()` следующие обязательные параметры:

Аргумент `input` - строка с исходным текстовым файлом и может быть установлен в один из трех вариантов:

- строка, содержащая путь к исходному файлу в файловой системе,
- файловый объект,
- `None` (по умолчанию), при этом функция будет читать поток из стандартного ввода.

Аргумент `output` - целевой объект, в который записывается вывод и может быть установлен в один из трех вариантов:

- строка, содержащая путь к исходному файлу в файловой системе,
- файловый объект,
- `None` (по умолчанию), при этом функция будет читать поток из стандартного ввода.

Аргумент `encoding` - кодировка исходного текстового файла. По умолчанию "utf-8". Для ввода и вывода всегда будет использоваться одна и та же кодировка. При кодировании вывода используется обработчик ошибок `xmlcharrefreplace`.

Примечание. Это единственное место, где происходит декодирование и кодирование Unicode.

`markdown.Markdown([**kwargs])`.

При инициализации класс `markdown.Markdown()` принимает те же параметры как и функция `markdown.markdown()`, за исключением того, что класс не принимает исходную текстовую строку при инициализации. Исходная текстовая строка `source` должна быть передана методу экземпляра `Markdown.convert(source)`.

Предупреждение. Экземпляр `markdown.Markdown()` является потокобезопасным только в том потоке, в котором он был создан. Один экземпляр не должен быть доступен из нескольких потоков.

Метод *`Markdown.convert(source)`*:

Метод `Markdown.convert()` преобразует разметку Markdown в HTML.

Аргумент `source` должен соответствовать тем же требованиям, что и аргумент `text` функции `markdown.markdown`.

Этот метод необходимо использовать, если надо обрабатывать несколько строк без создания нового экземпляра класса для каждой строки.

```
import markdown
md = markdown.Markdown()
html1 = md.convert(text1)
html2 = md.convert(text2)
```

Метод *`Markdown.reset()`*:

В зависимости от того, какие параметры и/или расширения используются, синтаксическому анализатору может потребоваться сброс состояния `Markdown.reset()` между каждым вызовом преобразования.

```
html1 = md.convert(text1)
md.reset()
html2 = md.convert(text2)
```

```
# или

html3 = md.reset().convert(text3)
```

Метод *`Markdown.convertFile(**kwargs)`*:

Аргументы этого метода идентичны одноименным аргументам функции `markdown.markdownFromFile()`: `input`, `output`, и `encoding`.

Как и метод `Markdown.convert()`, этот метод следует использовать для обработки нескольких файлов без создания нового экземпляра класса для каждого документа.

Может потребоваться сброс состояния между каждым вызовом `Markdown.convertFile()`, как в случае с `Markdown.convert()`.

Расширения модуля `markdown`.

Модуль `markdown` предлагает гибкий механизм расширения, который позволяет изменять и/или расширять поведение анализатора без необходимости редактировать исходный код.

Чтобы использовать расширение, передайте его в `markdown` с ключевым аргументом `extensions`.

Синтаксис:

```
import markdown

html = markdown.markdown(markdown_text,
                extensions=[MyExtClass(), 'myext',
'path.to.my.ext:MyExtClass'])
```

В командной строке укажите расширение с параметром `-x`.

```
python -m markdown -x myext -x path.to.module:MyExtClass input.txt >
output.html
```

Официально поддерживаемые расширения.

Перечисленные ниже расширения включены в самый последний выпуск и официально поддерживаются проектом `Python-Markdown`. Эти расширения доступны для использования сразу после типичной установки модуля `markdown`.

- `markdown.extensions.abbr`: сокращения, HTML тег `<abbr>`;
- `markdown.extensions.attr_list`: списки атрибутов;
- `markdown.extensions.def_list`: списки определений;

- `markdown.extensions.fenced_code`: защищенные блоки кода;
- `markdown.extensions.footnotes`: сноски;
- `markdown.extensions.md_in_html`: Markdown в HTML;
- `markdown.extensions.tables`: таблицы;
- `markdown.extensions.admonition`: добавляет `rST`-стиль;
- `markdown.extensions.codehilite`: CodeHilite;
- `markdown.extensions.legacy_attrs`: наследованные атрибуты;
- `markdown.extensions.legacy_em`: legacy Emphasis;
- `markdown.extensions.meta`: метаданные;
- `markdown.extensions.nl2br`: `<br>` для переноса строки;
- `markdown.extensions.sane_lists`: вменяемые списки;
- `markdown.extensions.toc`: содержание;
- `markdown.extensions.wikilinks`: WikiLinks.

Так же этот модуль имеет много сторонних расширений. **Внимание**, эти расширения были разработаны разными людьми, которые сделали их общедоступными. Команда `Python-Markdown` не одобряет и не предлагает никакой поддержки для этих расширений. За поддержкой обращайтесь к индивидуальному разработчику каждого расширения.

Использование модуля `markdown` в качестве командной строки.

Хотя проект `Python-Markdown` - это в первую очередь библиотека Python, в нее также включен сценарий командной строки. Для запуска сценария командной строки, как правило, необходимо, чтобы библиотека Markdown была полностью установлена в вашей системе

Сценарий командной строки `Python-Markdown` использует флаг -m интерпретатора Python. Используйте следующий формат для сценарий командной строки:

```
python -m markdown [options] [args]
```

Это запустит модуль как сценарий с предоставленными параметрами и аргументами.

В самом простом случае можно просто передать имя файла в качестве единственного аргумента:

```
python -m markdown input_file.txt
```

Также полностью поддерживается ввод и вывод на `STDIN` и `STDOUT`. Например:

```
echo "Some **Markdown** text." | python -m markdown > output.html
```

Используйте опцию `--help` для вывода списка всех доступных опций и аргументов:

```
python -m markdown --help
```