Язык шаблонов Content Monster 3

Основное предназначение

1. Генерация уникального, 100% читабельного контента
2. Генерация недостающего контента на сайте для внутренней перелинковки
3. Создание разделов сайта для поднятия трафика

Краткий обзор возможностей

Рассмотрим следующий блок:

```
{{ ["Дата публикации","Дата"]|choice }}
```

Фильтром *choice* мы выводим 1 значение из списка. Тем самым мы добиваемся уникальности этой строки в пределах сайта. Обязательно расширяйте список!

Разберем еще один блок:

```
{{ ["новый","свежий","интересный","забавный","",""]|choice }}
{{ [ ["он лайн","онлайн","он-лайн","online",""]|choice, ["видео",""]|choice, "ролик" ]|s
```

Здесь нам интересен фильтр *shuffle*, который произвольно перемешивает список, при этом список в свою очередь порождается из случайно выбранных значений списка (вложенная функция *choice*). Далее список объединяется фильтром *join(" ")* через пробел.

На выходе могут быть такие значения:

```
ролик видео онлайн
он-лайн ролик видео
интересный online  ролик
забавный  ролик он-лайн
```

Таким образом, манипулируя темплейтом, можно получать уникальные тематические описания какого угодно размера и степени уникальности.

Синтаксис

Шаблонный язык для генерирования контента использует технологию Jinja2 [2].

Применяются два типа разделителей. `{% ... %}` и `{{ ... }}`. Первый, для блочных тегов, таких как `for` и `if`, второй, для переменных.

Переменные

Для доступа к атрибутам либо переменным используется точка (`.`). Кроме того возможно использованием квадратных скобок (`[]`). Следующие строки идентичны:

```
{{ foo.bar }}
{{ foo['bar'] }}
```

Важно знать, что фигурные скобки не являются частью переменной, и для доступа к переменным из тегов, их указывать не требуется.

Если переменная не определена, то при печати будет выведена пустая стока, а в цикле не будет выполнено ни одной итерации. Однако другие действия вызовут ошибку генерации шаблона.

> ***Примечание***
>
> При обработке `foo.bar`:
>
> - ищется атрибут *bar* в *foo*.
> - если не найдено, ищется как ключ `'bar'` в *foo*.
> - если не найдено, возвращается неопределённый объект.
>
> `foo['bar']` поиск работает в ином порядке:
>
> - ищется как ключ `'bar'` в *foo*.
> - если не найден, ищется атрибут *bar* в *foo*.
> - если не найдено, возвращается неопределённый объект.
>
> Порядок поиска важен в случае если объект имеет одинаковые значения ключа и атрибута.

Встроенные фильтры

abs(number)

Возвращает абсолютное значение (модуль) аргумента.

Пример:

```
{% set zlo = -40 %}
{{ zlo|abs }}
```

Результат:

```
40
```

capitalize(s)

Капитализирует переменную. Первый символ будет заглавным. Остальные символы - строчными.

Пример:

```
{% set zlo = "А вот кто на новенького? ВСЕХ ЗАБАНЮ!" %}
{{ zlo|capitalize }}
```

Результат:

```
А вот кто на новенького? всех забаню!
```

center(value, width=80)

Размещает переменную в поле указанной длины

Пример:

```
{% set zlo = "А вот кто на новенького? ВСЕХ ЗАРЕЖУ!" %}
{{ zlo|center(100) }}
{{ zlo|center (10)}}
```

Результат:

```
                           А вот кто на новенького? ВСЕХ ЗАРЕЖУ!
А вот кто на новенького? ВСЕХ ЗАРЕЖУ!
```

default(value, default_value=u'', boolean=false)

Если переменная не определена, возвращает значение по умолчанию

Пример:

```
{{ zlo|default('переменная не определена') }}
{% set zlo = "определяем переменную" %}
{{ zlo }}
```

Результат:

```
переменная не определена
определяем переменную
```

filesizeformat(value, binary=false)

Форматирует числовое значение размера файла в читабельный вид (например 13 kB, 4.1 MB, 102 Bytes, и т.д.).

first(seq)

Возвращает первое значение последовательности, в том числе первую букву строки.

Пример 1:

```
{% set zlo = "3452" %}
{{ zlo|first }}
```

Результат:

```
3
```

Пример 2:

```
{{ [1,2,3]|first }}
```

Результат:

```
1
```

float(value, default=0.0)

Преобразует значение к числу с плавающей точкой. Если конвертирование не удается - возвращает 0.0

format(value, *args, **kwargs)

Форматирует строку в соответствии с правилами python:

```
{{ "%s - %s"|format("Hello?", "Foo!") }}
    -> Hello? - Foo!
```

groupby(value, attribute)

Группирует список значений по указанному атрибуту.

К примеру, если у вас есть список или словарь содержащий имя, фамилию и пол человека, посредством команды groupby этот список можно сгруппировать по полю *gender*, как в следующем примере:

```
<ul>
{% for group in persons|groupby('gender') %}
    <li>{{ group.grouper }}<ul>
    {% for person in group.list %}
        <li>{{ person.first_name }} {{ person.last_name }}</li>
    {% endfor %}</ul></li>
{% endfor %}
</ul>
```

Кроме того, кортеж можно сразу распаковать:

```
<ul>
{% for grouper, list in persons|groupby('gender') %}
    ...
{% endfor %}
</ul>
```

> ***Внимание!***
>
> Список перед группировкой должен быть уже отсортирован по полю группировки.

indent(s, width=4, indentfirst=false)

Возвращает текст где каждая строка, кроме первой, сдвинута вправо на указанное количество пробелов(по умолчанию 4). Если необходимо обработать все строки, укажите *indentfirst=true*:

```
{{ mytext|indent(2, true) }}
    indent by two spaces and indent the first line too.
```

int(value, default=0)

Преобразовывает переменную к целочисленному виду. Если преобразование не удалось, возвращает 0.

join(value, d=u'', attribute=None)

Возвращает строку, являющейся конкатенацией(сложением) строк в последовательности. По умолчанию, разделителем является пустая строка. Разделитель можно задать отдельно.

Пример:

```
{{ [1, 2, 3]|join('|') }}
```

Результат:

```
1|2|3
```

Пример 2:

```
{{ ["один", 2, 3]|join }}
```

Результат:

```
один23
```

last(seq)

Возвращает последний элемент из последовательности.

Пример:

```
{{ [1, 2, 3]|last }}
```

Результат:

```
3
```

length(object)

Возвращает число элементов в последовательности. В том числе можно определить длину строки.

list(value)

Преобразует переменную в список. Если переменная была строкой, возвращает список символов.

Пример:

```
{{ "злоба"|list }}
```

Результат:

```
["з","л","о","б","а"]
```

lower(s)

Преобразует переменную к строчному виду

Пример:

```
{{ "BLAP.RU"|lower }}
```

Результат:

```
blap.ru
```

pprint(value, verbose=false)

Выводит (печатает) переменную. Хорошо применять для отладки.

random(seq)

Возвращает случайный элемент из последовательности

replace(s, old, new, count=None)

Возвращает копию переменной, где все найденный подстроки заменяются на новые. Первая переменная - что ищем, вторая - на что меняем. Можно опционально указать число замен.

Пример:

```
{{ "Hello World"|replace("Hello", "Goodbye") }}
```

Результат:

```
Goodbye World
```

Пример:

```
{{ "аааа убили кенни"|replace("а", "они ", 2) }}
```

Результат:

```
они они аа убили кенни
```

reverse(value)

Возвращает последовательность, строку в обратном порядке.

round(value, precision=0, method='common')

Округляет число с заданной точностью. Первый параметр - точность округления, второй - метод округления.

common - применяется по умолчанию, округляет по правилам *ceil* - всегда округляет до максимального *floor* - всегда округляет до минимального

Пример:

```
{{ 42.55|round }}
```

Результат:

```
43.0
```

Пример:

```
{{ 42.55|round(1, 'floor') }}
```

Результат:

```
42.5
```

Обратите внимание, что если точность равна 0, то все равно будет выводится плавающая точка. Если нужно реально целое число - применяйте фильтр int

Пример:

```
{{ 42.55|round|int }}
```

Результат:

```
43
```

slice(value, slices, fill_with=None)

Нарезает последовательность на фрагменты. К примеру, если необходимо сформировать три столбца содержащие списки:

```
<div class="columwrapper">
  {%- for column in items|slice(3) %}
    <ul class="column-{{ loop.index }}">
    {%- for item in column %}
      <li>{{ item }}</li>
    {%- endfor %}
```

```
    </ul>
  {%- endfor %}
</div>
```

Если вы укажете параметр fill_with, то недостающие элементы будут содержать переданное значение.

sort(value, reverse=false, case_sensitive=false, attribute=None)

Сортировка последовательности. По умолчанию, в порядке возрастания. Если сортируются строки, то возможно указать регистр символов.:

```
{% for item in iterable|sort %}
    ...
{% endfor %}
```

Кроме того, при сортировке последовательностей, возможно указать поле по которому будет производиться сравнение.:

```
{% for item in iterable|sort(attribute='date') %}
    ...
{% endfor %}
```

string(object)

Приводит строку к уникоду.

striptags(value)

Удаляет SGML/XML тэги и заменяет несколько пробелов одним.

sum(iterable, attribute=None, start=0)

Возвращает сумму элементов последовательности, прибавляя к ним *start*. Кроме того, можно рассчитать сумму указанных атрибутов элементов:

```
Total: {{ items|sum(attribute='price') }}
```

title(s)

Преобразовывает переменную таким образом, что первый символ слова будет Строчным, все остальные прописными

trim(value)

Удаляет пробелы впереди и сзади строки (переменной)

truncate(s, length=255, killwords=false, end='...')

Возвращает усеченную копию строки. Длина задается первым параметром (по умолчанию 255). Если второй параметр true, то будет усекать строку по заданной длине, в противном случае постарается обрезать строку по окончании слова. Если текст был усечен, то добавляет троеточие, которое можно настроить в третьей переменной.

upper(s)

Конвертирует в заглавные символы.

urlize(value, trim_url_limit=None, nofollow=false)

Преобразует url переданный в виде текста в активную ссылку. Вторая переменная сокращает адрес до заданного целого числа знаков. Третья переменная добавляет атрибут rel="nofollow"

wordcount(s)

Подсчитывает слова в строке

wordwrap(s, width=79, break_long_words=true)

Возвращает копию строки, нарезанную на части (в данном случае - после 70 символа). Если вы установите второй параметр в false, то слова не будут делиться на части, даже если строка длинней установленного числа символов.

xmlattr(d, autospace=true)

Создание SGML/XML аттрибутов строки на основании словаря.

Пример:

```
<ul{{ {'class': 'my_list', 'missing': none,
        'id': 'list-%d'|format(variable)}|xmlattr }}>
...
</ul>
```

Результат:

```
<ul class="my_list" id="list-42">
...
</ul>
```

Расширенные фильтры Content Monster 3

shuffle(список)

Перемешивание списка:

```
{{ ["раз","два","три"]|shuffle|join("-") }}
```

by_dict

Обработка встроенным словарем.

Пример:

```
{{ "Радость"|by_dict }} {{ "Урок"|by_dict }}
```

choice(список)

Выборка одного, случайного, значения из списка.

Пример вызова:

```
{{ ["раз","два","три"]|choice }}
```

in_words(число)

Число прописью.

Пример:

```
{{ "55.5"|in_words }}
```

split(разделитель)

Делит строку по заданному разделителю.

Пример:

```
{{ "1,2,3,4,5"|split(',') }}
```

Комментарии

Для комментирования используется следующая комбинация `{# ... #}`.

Пример:

```
{# алярм: этот фрагмент более не нужен
    {% for user in users %}
        ...
    {% endfor %}
#}
```

Управление переводами строк

По умолчанию каждая строка шаблона добавляет в конце перевод на следующую.

Если такое поведение неприемлимо, вы можете изменить его, добавив знак минус(`-`) в начале и конце блока (для примера тег for), примерно так:

```
{% for item in seq -%}
    {{ item }}
{%- endfor %}
```

Это выведет все значения от 1 до 9 без перевода строк, вот так 123456789.

Внимание!

Между знаком минуса и знаком процента не должно быть пробелов.

правильно:

```
{%- if foo -%}...{% endif %}
```

неправильно:

```
{% - if foo - %}...{% endif %}
```

Экранирование

В тех случаях, когда требуется вывести символы, использующиеся для обозначения тегов или блоков тегов, самым простым вариантом является заключение их в строку. Например, если требуется вывести {{ то это сделать возможно следующим образом:

```
{{ '{{' }}
```

Однако, для большого объема информации, это не всегда удобно. В таком случае применяется блочный тег *raw*. Пример:

```
{% raw %}
    <ul>
    {% for item in seq %}
        <li>{{ item }}</li>
    {% endfor %}
    </ul>
{% endraw %}
```

Блочные выражения

Управляющие структуры, на которых базируется логика работы шаблона(циклы, условия) Объявляются посредством блоков {% ... %}.

For(цикл)

Для обхода последовательностей используется блочный тег *for*. Для примера, отображение всех user в списке *users*:

```
<h1>Пользователи</h1>
<ul>
{% for user in users %}
  <li>{{ user.username }}</li>
{% endfor %}
</ul>
```

Так-же, *for* возможно использовать для обхода словарей типа *dict*:

```
<dl>
{% for key, value in my_dict.iteritems() %}
    <dt>{{ key }}</dt>
    <dd>{{ value }}</dd>
{% endfor %}
</dl>
```

Обратите внимание, словари обычно не отсортированы в порядке добавления элементов.

Внутри блочного *for* доступны следующие переменные:

Переменная	Описание
loop.index	Текущая итерация. (начиная с 1)
loop.index0	Текущая итерация. (начиная с 0)
loop.revindex	Количество оставшихся итераций (начиная с 1)
loop.revindex0	Количество оставшихся итераций (начиная с 0)
loop.first	true если первая итерация (подходит для вывода заголовков таблицы).
loop.last	true если крайняя итерация.
loop.length	Количество элементов в последовательности.
loop.cycle	Вспомогательная переменная для перебора внутреннего списка. К примеру для выделения четных строк. Пример далее.

Внутри блока for возможно использование особой переменной *loop.cycle*:

```
{% for row in rows %}
    <li class="{{ loop.cycle('odd', 'even') }}">{{ row }}</li>
{% endfor %}
```

В результате все нечетные строки получат класс *odd*, а четные *even*.

Для фильтрации элементов вы можете использовать следующую *if* конструкцию:

```
{% for user in users if not user.hidden %}
    <li>{{ user.username }}</li>
{% endfor %}
```

Если список пуст, либо он стал таковым после фильтрации *if*, то будет вызван блок *else*:

```
<ul>
{% for user in users %}
    <li>{{ user.username }}</li>
{% else %}
    <li><em>Нет пользователей</em></li>
{% endfor %}
</ul>
```

Так-же возможно использовать циклы рекурсивно. К примеру, для создания карты сайта. Для этого вам необходимо добавить модификатор *recursive* в объявлении цикла и вызывать функцию *loop* для создания рекурсии.

Следующий пример показывает как это можно сделать:

```
<ul class="sitemap">
{%- for item in sitemap recursive %}
    <li><a href="{{ item.href }}">{{ item.title }}</a>
    {%- if item.children -%}
        <ul class="submenu">{{ loop(item.children) }}</ul>
    {%- endif %}</li>
{%- endfor %}
</ul>
```

Управление циклом возможно с помощью *break* and *continue* тегов. *break* прерывает цикл; *continue* вызывает переход на следующую итерацию.

В следующем примере пропускаются чётные строки:

```
{% for user in users %}
    {%- if loop.index is even %}{% continue %}{% endif %}
    ...
{% endfor %}
```

В этом примере происходит выход из цикла после десяти итераций:

```
{% for user in users %}
    {%- if loop.index >= 10 %}{% break %}{% endif %}
{%- endfor %}
```

If(условие)

Тег *if* случит для проверки на true, не 0, или на то, что список(кортеж) не пустой:

```
{% if users %}
<ul>
{% for user in users %}
    <li>{{ user.username }}</li>
{% endfor %}
</ul>
{% endif %}
```

Для ветвления возможно использовать *elif* и *else*. Вы можете и более сложные значения:

```
{% if kenny.sick %}
    Кенни болен.
{% elif kenny.dead %}
    Они убили Кенни! Сволочи!!!
{% else %}
    Кенни живой --- пока что
{% endif %}
```

Вы так-же можете использовать *if* как выражение для фильтрации в циклах.

Макросы

Макросы представляют из себя обычные функции и служат для вызова повторяющего кода.

Ниже пример вывода кода формы ввода:

```
{% macro input(name, value='', type='text', size=20) -%}
    <input type="{{ type }}" name="{{ name }}" value="{{
        value|e }}" size="{{ size }}">
{%- endmacro %}
```

Макрос может быть вызван следующим образом:

```
<p>{{ input('username') }}</p>
<p>{{ input('password', type='password') }}</p>
```

Внутри макросов доступны следующие переменные:

varargs

Если в макрос передано позиционных переменных более чем объявлено доступ к ним можно получить посредством специальной переменной *varargs* содержащей список этих значений.

kwargs

Служит для доступа к переданным по ключу необъявленным параметрам.

caller

Если макрос был вызван из блока *call<call>* то *caller* содержит данные этого блока.

Кроме того макрос содержит некоторые внутренние поля:

name

Имя макроса. `{{ input.name }}` отобразит `input`.

arguments

Кортеж имен аргументов принимаемых макросом.

defaults

Кортеж значений по умолчанию.

catch_kwargs

Возвращает *true*, если переданы дополнительные параметры по ключу.

catch_varargs

Возвращает *true*, если переданы дополнительные позиционные параметры.

caller

Возвращает *true* если макрос вызван специальным блоком *caller*.

Цепочки макросов

Иногда возникает необходимость в передаче одного макроса другому. Для этих целей используется специальный блок *call*. Следующий пример показывается использование такого рода функционала:

```
{% macro render_dialog(title, class='dialog') -%}
    <div class="{{ class }}">
        <h2>{{ title }}</h2>
        <div class="contents">
            {{ caller() }}
        </div>
    </div>
{%- endmacro %}

{% call render_dialog('Hello World') %}
    Этот текст выводиться в блок content макроса render_dialog.
{% endcall %}
```

Кроме того, существует возможность передать данные в вызывающий макрос. В таком случае цепочки макросов можно применять вместо циклов. В общем случае вызов блока работает как макрос не имеющий имени.

Следующий пример отображает список всех пользователей из list_of_user:

```
{% macro dump_users(users) -%}
    <ul>
    {%- for user in users %}
        <li><p>{{ user.username|e }}</p>{{ caller(user) }}</li>
    {%- endfor %}
    </ul>
{%- endmacro %}

{% call(user) dump_users(list_of_user) %}
    <dl>
        <dl>Имя</dl>
        <dd>{{ user.realname|e }}</dd>
        <dl>Характеристика на члена НСДАП</dl>
        <dd>{{ user.description }}</dd>
    </dl>
{% endcall %}
```

Блочные фильтры

Все фильтры возможно применять для обработки фрагментов. Просто оборачивайте их специальным блоком filter:

```
{% filter upper %}
    Этот текст выведется в верхнем регистре.
{% endfilter %}
```

Присваивание

Для присваивания значений используется тег *set*:

```
{% set navigation = [('index.html', 'Главная'), ('about.html', 'О нас')] %}
{% set key, value = call_something() %}
```

Выражения

Jinja поддерживание базовые выражения во всех блоках.

Литералы

Литералы представляют объекты Python, такие как списки, строки, цифры:

"Привет Мир":

Всё, что находиться между одинарными или двойными кавычками - строка.

42 / 42.23:

Все что записано цифрами - есть либо целое число, либо с число плавающей запятой.

['список', 'всех', 'объектов']:

Все, между квадратными скобками - список. Он может содержать последовательность различных данных. Например кортежи из ссылки и заголовка:

```
<ul>
{% for href, caption in [('index.html', 'Главная'), ('about.html', 'О нас'),
                         ('downloads.html', 'Файлы')] %}
    <li><a href="{{ href }}">{{ caption }}</a></li>
{% endfor %}
</ul>
```

('a', 'это ', 'кортеж'):

Кортеж является неизменяемым списком. При определении кортежа состоящего из одного элемента, необходимо после него добавлять запятую, для отличия от вызова функции.

{'ключ1': 'значение1', 'ключ2': 'значение2', 'ключ3': 'значение3'}:

Словарь задается парами ключ-значение. Ключи должен быть уникальными.

true / false:

true всегда истинно и false всегда ложно.

> ***Внимание!***
>
> Специальные константы *true*, *false* и *none* необходимо задавать в нижнем регистре.

Вычисление

Jinja позволяет выполнять математические операции с переменными:

+

Сложение двух переменных. Если переменная - строка, то объединит строки. Однако для сложения строк лучше использовать оператор ~ .

Пример:

```
{{ 1 + 1 }}
```

Результат:

```
2
```

-

Вычитание второй переменной из первой.

Пример:

```
{{ 3 - 2 }}
```

Результат:

```
1
```

/

Деление 2-х чисел. Результат возвращается в виде числа с плавающей точкой.

Пример:

```
{{ 1 / 2 }}
```

Результат:

```
0.5
```

//

Деление нацело. Возвращает целую часть операции деления.

Пример:

```
{{ 20 // 7 }}
```

Результат:

```
2
```

%

Рассчитывает остаток целочисленного деления

Пример:

```
{{ 11 % 7 }}
```

Результат:

```
4
```

*****

Умножает левый операнд на правый

Пример:

```
{{ 2 * 2 }}
```

Результат:

```
4
```

Может быть использован для генерации повторяющейся строки

Пример:

```
{{ '=' * 10 }}
```

Результат:

```
==========
```

******

Возводит левый операнд в степень правого

Пример:

```
{{ 2**3 }}
```

Результат:

```
8
```

Сравнение

==

Сравнивает два объекта на равенство

!=

Сравнивает два объекта на неравенство

>

истина, когда левая часть сравнения больше правой

>=

истина, когда левая часть сравнения больше или равно правой

<

истина, когда левая часть сравнения меньше правой

<=

истина, когда левая часть сравнения меньше или равно правой

Логические

Полезные операторы при задании сложных условий

and

Возвращает true если левый и правый операнд установлены в true

or

Возвращает true если левый или правый операнд установлены в true

not

отрицает заявленное

(expr)

группировать выражение

> ***Внимание!***
>
> При использовании операторов `is` и `in` при отрицании, применяйте `foo is not bar` и `foo not in bar` вместо `not foo is bar` и `not foo in bar`. Все остальные выражения требуют префиксной нотации: `not (foo and bar)`.

Другие операторы

in

Проверяет последовательность на вхождение элемента. Возвращает истину, если правый операнд содержит левый.

Пример:

```
{{ 1 in [1, 2, 3] }}
```

Результат:

```
true
```

is

Выполняет проверку на истину.

|

Применяет фильтр.

~

Преобразовывает все операнды в строку и складывает (объединяет) их. Пример:

```
{{ "Hello " ~ name ~ "!" }}
```

возвращает строку (при установленном name в 'John') Hello John!.

()

Вызывает callable: {{ post.render() }}. Внутри скобки можно использовать позиционные аргументы и ключевые аргументы, как в Python {{ post.render(user, full=true) }}.

. / []

Получает аттрибуты объекта

Условные выражения

Тег *if* возможно использовать как условное выражение. Базовый синтаксис следующий:

```
``<выполнить что-то> if <что-то истинно> else <выполнить что-то другое>``.
```

Фрагмент *else* необязателен. Как пример, отобразить значение переменной page.title, если она определена:

```
{{ '[%s]' % page.title if page.title }}
```

Глобальные функции

range([start,] stop[, step])

Возвращает список содержащий арифметическую прогрессию. range(i, j) возвращает [i, i+1, i+2, ..., j-1]; start по умолчанию 0:

```
<ul>
{% for user in users %}
    <li>{{ user.username }}</li>
{% endfor %}
{% for number in range(10 - users|count) %}
    <li class="empty"><span>...</span></li>
{% endfor %}
</ul>
```

lipsum(n=5, html=true, min=20, max=100)

Генерирует "рыбу"(lorem ipsum..). По умолчанию пять параграфов содержащие от 20 до 100 слов. Если html истинно, добавляет html тег параграфа.

dict(**items)

Альтернатива фигурным скобкам. `{'foo': 'bar'}` то-же самое что `dict(foo='bar')`.

cycler(*items)

Похоже на loop.cycle только вне цикла.

Следующий пример показывает вариант использования *cycler*:

```
{% set row_class = cycler('odd', 'even') %}
<ul class="browser">
{% for folder in folders %}
  <li class="folder {{ row_class.next() }}">{{ folder|e }}</li>
{% endfor %}
{% for filename in files %}
  <li class="file {{ row_class.next() }}">{{ filename|e }}</li>
{% endfor %}
</ul>
```

Cycler имеет следующие атрибуты и методы:

```
reset()

    Выбор первого элемента в последовательности.

next()

    Возвращает текущий элемент и переходит на следующий.

current

    Возвращает текущий элемент.
```

joiner(sep=', ')

Вспомогательная функция для объединения нескольких секций данных. Возвращает переданную строку, за исключением первого вызова:

```
{% set pipe = joiner("|") %}
{% if categories %} {{ pipe() }}
    Категории: {{ categories|join(", ") }}
{% endif %}
{% if author %} {{ pipe() }}
    Авторы: {{ author() }}
{% endif %}
{% if can_edit %} {{ pipe() }}
    <a href="?action=edit">Редактировать</a>
{% endif %}
```

---

1 Официальный сайт Content Monster 3.
2 Официальный сайт Jinja2.