장고

선적 서류 비치

내장 템플리트 태그 및 필터

이 문서는 Django에 내장 된 템플릿 태그와 필터를 설명합니다. 가능한 경우 자동 문서가 사용되는 것이 좋습니다. 여기 에는 설치된 모든 사용자 정의 태그 또는 필터에 대한 문서가 포함됩니다.

내장 태그 참조

autoescape

현재 자동 이스케이프 동작을 제어합니다. 이 태그는 하나 **on**또는 **off**인수를 취하며 자동 이스케이프가 블록 내부에서 적용되는지 여부를 결정합니다. 블록은 **endautoescape**종료 태그로 닫힙니다 .

자동 이스케이프가 실행 중일 때 결과를 출력에 배치하기 전에 (모든 필터가 적용된 후에) 모든 가변 내용에 HTML 이스케이프가 적용됩니다. 이는 **escape** 각 변수에 필터를 수동으로 적용하는 것과 같습니다 .

유일한 예외는 변수를 채우는 코드에 의해 **safe**또는 **escape**필터가 적용 되었기 때문에 이미 이스케이프에서 "안전"하다고 표시된 변수입니다 .

샘플 사용법 :

```
{% autoescape on %}
    {{ body }}
{% endautoescape %}
```

---

block

자식 템플릿에 의해 재정의 될 수있는 블록을 정의합니다. 자세한 정보는 템플리트 상속 을 참조하십시오.

---

comment

와 및 사이의 모든 것을 무시합니다 . 선택적 참고가 첫 번째 태그에 삽입 될 수 있습니다. 예를 들어, 이는 코드가 사용되지 않는 이유를 문서화하기위한 코드를 주석 처리할 때 유용합니다.**{% comment %}{% endcomment %}**

샘플 사용법 :

```
<p>Rendered text with {{ pub_date|date:"c" }}</p>
{% comment "Optional note" %}
    <p>Commented out text with {{ create_date|date:"c" }}</p>
{% endcomment %}
```

comment 태그는 중첩 될 수 없습니다.

---

csrf_token

이 태그는 교차 사이트 요청 위조 에 대한 설명서에 설명 된대로 CSRF 보호에 사용됩니다 .

언어 : **ko**

문서 버전 : **2.2**

---

cycle

이 태그가 발생할 때마다 인수 중 하나를 생성합니다. 첫 번째 인수는 첫 번째 발생에서 생성되고 두 번째 인수는 두 번째 발생에서 생성됩니다. 모든 인수가 모두 소모되면 태그는 첫 번째 인수로 순환하고 다시 인수를 생성합니다.

이 태그는 루프에서 특히 유용합니다.

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' 'row2' %}">
        ...
    </tr>
{% endfor %}
```

첫 번째 반복은 클래스를 참조하는 HTML을 생성 **row1**하고, 두 번째 **row2**, 세 번째 **row1**, 다시 세 번째 반복을 반복합니다.

변수도 사용할 수 있습니다. 예를 들어, 두 개의 템플릿 변수를 가지고있는 경우 **rowvalue1**와 **rowvalue2**, 당신은이 같은 자신의 가치를 교대로 할 수 있습니다 :

```
{% for o in some_list %}
    <tr class="{% cycle rowvalue1 rowvalue2 %}">
        ...
    </tr>
{% endfor %}
```

사이클에 포함 된 변수가 이스케이프됩니다. 다음을 사용하여 자동 이스케이프를 비활성화 할 수 있습니다.

```
{% for o in some_list %}
    <tr class="{% autoescape off %}{% cycle rowvalue1 rowvalue2 %}{% endautoescape %}">
        ...
    </tr>
{% endfor %}
```

변수와 문자열을 혼합 할 수 있습니다.

```
{% for o in some_list %}
    <tr class="{% cycle 'row1' rowvalue2 'row3' %}">
        ...
    </tr>
{% endfor %}
```

어떤 경우에는 다음 값으로 진행하지 않고 사이클의 현재 값을 참조 할 수 있습니다. 이렇게하려면 태그에 "as"를 사용하여 다음과 같이 이름을 지정하면됩니다.**{% cycle %}**

```
{% cycle 'row1' 'row2' as rowcolors %}
```

그런 다음주기 이름을 컨텍스트 변수로 참조하여 템플릿에 원하는 모든 사이클의 현재 값을 삽입 할 수 있습니다. 원래 **cycle**태그와 독립적으로 다음 값으로 사이클을 이동 하려면 다른 **cycle**태그를 사용하고 변수의 이름을 지정할 수 있습니다 . 그래서, 다음 템플릿 :

```
<tr>
    <td class="{% cycle 'row1' 'row2' as rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
<tr>
    <td class="{% cycle rowcolors %}">...</td>
    <td class="{{ rowcolors }}">...</td>
</tr>
```

출력 할 것이다 :

```
<tr>
    <td class="row1">...</td>
    <td class="row1">...</td>
</tr>
<tr>
    <td class="row2">...</td>
    <td class="row2">...</td>
</tr>
```

cycle공백으로 구분 하여 태그 에 원하는 수의 값을 사용할 수 있습니다 . 작은 따옴표 (**'**) 나 큰 따옴표 (**"**)로 묶인 값은 문자열 리터럴로 취급되며 따옴표가없는 값은 템플릿 변수로 취급됩니다.

기본적으로 **as**키워드를 싸이클 태그와 함께 사용하면 해당 싸이클 태그를 사용 하면 싸이클 자체가 싸이클의 첫 번째 값을 생성합니다. 중첩 루프 또는 포함 된 템플릿에서 값을 사용하려는 경우 문제가 될 수 있습니다. 주기를 선언하고 첫 번째 값을 생성하지 않으려는 경우 키워드를 태그의 마지막 키워드로 추가 할 수 있습니다 . 예 :**{% cycle %}silent**

```
{% for obj in some_list %}
    {% cycle 'row1' 'row2' as rowcolors silent %}
    <tr class="{{ rowcolors }}">{% include "subtemplate.html" %}</tr>
{% endfor %}
```

이 뜻 출력리스트 **<tr>**와 요소 **class** 사이에 교대 **row1**하고 **row2**. 하위 템플릿은 **rowcolors**해당 컨텍스트에서 액세스 할 수 있으며 값은 해당 템플릿을 포함 하는 클래스와 일치 **<tr>**합니다. 경우] **silent**키워드 있었다 생략하고, **row1**및 **row2**외부 정상적인 텍스트로 방출된다 **<tr>**소자.

사일런트 키워드가 싸이클 정의에 사용되면 자동 사일런스는 해당 싸이클 태그의 모든 후속 용도에 자동으로 적용됩니다. 두 번째 호출이 지정하지 않아도 다음 템플릿은 *아무 것도* 출력 하지 않습니다 .**{% cycle %}silent**

```
{% cycle 'row1' 'row2' as rowcolors silent %}
{% cycle rowcolors %}
```

resetcycle태그를 사용하여 다음 번에 태그가 처음 나타날 때 태그를 다시 시작할 수 있습니다 .**{% cycle %}**

---

debug

현재 컨텍스트 및 가져온 모듈을 포함하여 전체 디버깅 정보로드를 출력합니다.

---

extends

이 템플릿이 상위 템플릿을 확장한다는 신호입니다.

이 태그는 두 가지 방법으로 사용할 수 있습니다.

- **{% extends "base.html" %}**(따옴표 사용)은 리터럴 값 **"base.html"**을 확장 할 상위 템플릿의 이름으로 사용합니다.

언어 : **ko**

문서 버전 : **2.2**

{% extends variable %}의 값을 사용합니다 **variable**. 변수가 문자열로 평가되면 Django는 그 문자열을 부모 템플릿의 이름으로 사용합니다. 변수가 **Template**객체로 평가 되면 Django는 그 객체를 부모 템플릿으로 사용합니다.

자세한 정보는 템플리트 상속 을 참조하십시오.

일반적으로 템플리트 이름은 템플리트 로더의 루트 디렉토리에 상대적입니다. 문자열 인수는 **./**또는 로 시작하는 상대 경로 일 수도 있습니다 **../**. 예를 들어, 다음 디렉토리 구조를 가정합니다.

```
dir1/
    template.html
    base2.html
    my/
        base3.html
base1.html
```

에서 **template.html**, 다음 경로는 유효 할 것이다 :

```
{% extends "./base2.html" %}
{% extends "../base1.html" %}
{% extends "./my/base3.html" %}
```

---

filter

하나 이상의 필터를 통해 블록의 내용을 필터링합니다. 여러 개의 필터를 파이프와 함께 지정할 수 있으며 필터는 변수 구문과 마찬가지로 인수를 가질 수 있습니다.

블록에는 및 태그 사이의 *모든* 텍스트가 포함됩니다 .**filterendfilter**

샘플 사용법 :

```
{% filter force_escape|lower %}
    This text will be HTML-escaped, and will appear in all lowercase.
{% endfilter %}
```

노트

escape및 **safe**필터는 허용 인수하지 않습니다. 대신 **autoescape**태그를 사용하여 템플릿 코드 블록에 대한 자동 이스케이프를 관리하십시오.

---

firstof

첫 번째 인수가 아닌 변수를 출력합니다 **False**. 전달 된 변수가 모두있는 경우 아무 것도 출력하지 않습니다 **False**.

샘플 사용법 :

```
{% firstof var1 var2 var3 %}
```

이것은 다음과 같습니다.

```
{% if var1 %}
    {{ var1 }}
{% elif var2 %}
    {{ var2 }}
{% elif var3 %}
    {{ var3 }}
{% endif %}
```

전달 된 모든 변수가 False 인 경우 리터럴 문자열을 대체 값으로 사용할 수도 있습니다.

```
{% firstof var1 var2 var3 "fallback value" %}
```

이 태그는 변수 값을 자동으로 이스케이프합니다. 다음을 사용하여 자동 이스케이프를 비활성화 할 수 있습니다.

```
{% autoescape off %}
    {% firstof var1 var2 var3 "<strong>fallback value</strong>" %}
{% endautoescape %}
```

또는 일부 변수 만 이스케이프해야하는 경우 다음을 사용할 수 있습니다.

```
{% firstof var1 var2|safe var3 "<strong>fallback value</strong>"|safe %}
```

구문 을 사용하여 출력을 변수에 저장할 수 있습니다 .**{% firstof var1 var2 var3 as value %}**

---

for

배열의 각 항목을 루프하여 항목을 컨텍스트 변수에서 사용할 수 있도록합니다. 예를 들어, 제공 선수의 목록을 표시한다 **athlete_list**:

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

를 사용하여 목록을 역순으로 반복 할 수 있습니다 .**{% for obj in list reversed %}**

목록 목록을 반복해야하는 경우 각 하위 목록의 값을 개별 변수로 압축 해제 할 수 있습니다. 예를 들어 컨텍스트에 호출 된 (x, y) 좌표 목록이 포함되어있는 **points**경우 다음을 사용하여 점 목록을 출력 할 수 있습니다.

```
{% for x, y in points %}
    There is a point at {{ x }},{{ y }}
{% endfor %}
```

사전의 항목에 액세스해야하는 경우에도 유용 할 수 있습니다. 예를 들어 컨텍스트에 사전이 포함 된 **data**경우 다음은 사전 의 키와 값을 표시합니다.

```
{% for key, value in data.items %}
    {{ key }}: {{ value }}
{% endfor %}
```

Keep in mind that for the dot operator, dictionary key lookup takes precedence over method lookup. Therefore if the **data** dictionary contains a key named **'items'**, **data.items** will return **data['items']** instead of **data.items()**. Avoid adding keys that are named like dictionary methods if you want to use those methods in a template (**items**, **values**, **keys**, etc.). Read more about the lookup order of the dot operator in the documentation of template variables.

The for loop sets a number of variables available within the loop:

Variable	Description
forloop.counter	The current iteration of the loop (1-indexed)
forloop.counter0	The current iteration of the loop (0-indexed)
forloop.revcounter	The number of iterations from the end of the loop (1-indexed)
forloop.revcounter0	The number of iterations from the end of the loop (0-indexed)
forloop.first	True if this is the first time through the loop
forloop.last	True if this is the last time through the loop
forloop.parentloop	For nested loops, this is the loop surrounding the current one

for … empty

The **for** tag can take an optional **{% empty %}** clause whose text is displayed if the given array is empty or could not be found:

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% empty %}
    <li>Sorry, no athletes in this list.</li>
{% endfor %}
</ul>
```

The above is equivalent to – but shorter, cleaner, and possibly faster than – the following:

```
<ul>
  {% if athlete_list %}
    {% for athlete in athlete_list %}
      <li>{{ athlete.name }}</li>
    {% endfor %}
  {% else %}
    <li>Sorry, no athletes in this list.</li>
  {% endif %}
</ul>
```

if

The **{% if %}** tag evaluates a variable, and if that variable is “true” (i.e. exists, is not empty, and is not a false boolean value) the contents of the block are output:

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|length }}
{% elif athlete_in_locker_room_list %}
    Athletes should be out of the locker room soon!
{% else %}
    No athletes.
{% endif %}
```

In the above, if **athlete_list** is not empty, the number of athletes will be displayed by the **{{ athlete_list|length }}** variable.

보시다시피 **if**태그 에는 하나 이상의 절이있을 수 있으며 모든 이전 조건이 실패 할 경우 표시 되는 절이있을 수 있습니다. 이 절은 선택 사항입니다.**{% elif %}{% else %}**

부울 연산자

if태그를 사용할 수 있습니다 **and**, **or**또는 **not**변수의 수를 테스트하거나 특정 변수를 부정 :

```
{% if athlete_list and coach_list %}
    Both athletes and coaches are available.
{% endif %}

{% if not athlete_list %}
    There are no athletes.
{% endif %}

{% if athlete_list or coach_list %}
    There are some athletes or some coaches.
{% endif %}

{% if not athlete_list or coach_list %}
    There are no athletes or there are some coaches.
{% endif %}

{% if athlete_list and not coach_list %}
    There are some athletes and absolutely no coaches.
{% endif %}
```

동일한 태그 내에서 both **and**및 **or**clause를 사용할 수 있으며 예를 들어 다음 **and**보다 우선 순위가 높습니다 **or**.

```
{% if athlete_list and coach_list or cheerleader_list %}
```

다음과 같이 해석 될 것이다 :

```
if (athlete_list and coach_list) or cheerleader_list
```

if태그 에서 실제 괄호를 사용하면 유효하지 않은 구문입니다. 우선 순위를 나타내는 데 필요하면 중첩 된 **if**태그를 사용해야 합니다.

if태그는 또한 사업자 사용할 수 있습니다 **==**, **!=**, **<**, **>**, **<=**, **>=**, **in**, , , 그리고 다음과 같이 어떤 일을 :**not inisis not**

==연산자

평등. 예:

```
{% if somevar == "x" %}
  This appears if variable somevar equals the string "x"
{% endif %}
```

!=연산자

불평등. 예:

```
{% if somevar != "x" %}
  This appears if variable somevar does not equal the string "x",
  or if somevar is not found in the context
{% endif %}
```

<연산자

미만. 예:

```
{% if somevar < 100 %}
  This appears if variable somevar is less than 100.
{% endif %}
```

>연산자

보다 큰. 예:

```
{% if somevar > 0 %}
  This appears if variable somevar is greater than 0.
{% endif %}
```

<=연산자

보다 작거나 같음 예:

```
{% if somevar <= 100 %}
  This appears if variable somevar is less than 100 or equal to 100.
{% endif %}
```

>=연산자

크거나 같음 예:

```
{% if somevar >= 1 %}
  This appears if variable somevar is greater than 1 or equal to 1.
{% endif %}
```

언어 : **ko**

문서 버전 : **2.2**

in연산자

안에 들어 있습니다. 이 연산자는 많은 Python 컨테이너에서 지원되어 주어진 값이 컨테이너에 있는지 테스트합니다. 다음은 해석 방법에 대한 몇 가지 예입니다 .**x in y**

```
{% if "bc" in "abcdef" %}
  This appears since "bc" is a substring of "abcdef"
{% endif %}

{% if "hello" in greetings %}
  If greetings is a list or set, one element of which is the string
  "hello", this will appear.
{% endif %}

{% if user in users %}
  If users is a QuerySet, this will appear if user is an
  instance that belongs to the QuerySet.
{% endif %}
```

---

not in연산자

안에 포함되어 있지 않습니다. 이것은 **in**운영자 의 부정입니다 .

---

is연산자

객체 식별. 2 개의 값이 같은 오브젝트인가 어떤가를 판정합니다. 예:

```
{% if somevar is True %}
  This appears if and only if somevar is True.
{% endif %}

{% if somevar is None %}
  This appears if somevar is None, or if somevar is not found in the context.
{% endif %}
```

---

is not연산자

부인 된 개체 신원. 2 개의 값이 같은 오브젝트가 아닌지를 판정합니다. 이것은 **is**운영자 의 부정입니다 . 예:

```
{% if somevar is not True %}
  This appears if somevar is not True, or if somevar is not found in the
  context.
{% endif %}

{% if somevar is not None %}
  This appears if and only if somevar is not None.
{% endif %}
```

필터

if표현식에 필터를 사용할 수도 있습니다 . 예 :

```
{% if messages|length >= 100 %}
   You have lots of messages today!
{% endif %}
```

복잡한 표현식

위의 모든 것을 결합하여 복잡한 표현식을 구성 할 수 있습니다. 그러한 표현식의 경우, 표현식이 평가 될 때 연산자가 그룹화되는 방식, 즉 선행 규칙을 알아야합니다. 연산자의 우선 순위는 가장 낮은 것에서 가장 높은 것까지 다음과 같습니다.

- **or**
- **and**
- **not**
- **in**
- **==**, **!=**, **<**, **>**, **<=**,**>=**

(이것은 정확히 파이썬을 따른다). 예를 들어 다음과 같은 복잡한 **if**태그가 있습니다.

```
{% if a == b or c == d and e %}
```

...는 다음과 같이 해석됩니다.

```
(a == b) or ((c == d) and e)
```

다른 우선 순위가 필요한 경우에는 중첩 된 **if**태그 를 사용해야 합니다. 우선 순위 규칙을 모르는 사람들을 위해 어쨌든 명확성을 위해 더 나은 경우가 있습니다.

비교 연산자는 파이썬이나 수학 표기법과 같이 '연쇄'될 수 없습니다. 예를 들어 다음을 사용하는 대신

```
{% if a > b > c %}  (WRONG)
```

다음을 사용해야합니다.

```
{% if a > b and b > c %}
```

---

ifequal및 ifnotequal

{% ifequal a b %} ... {% endifequal %}쓸모없는 방법 입니다. 마찬가지로, 에 의해 대체됩니다 . 와 태그는 향후 릴리스에서 더 이상 사용되지 않습니다.**{% if a == b %} ... {% endif %}{% ifnotequal a b %} ... {% endifnotequal %}{% if a != b %} ... {% endif %}ifequalifnotequal**

---

ifchanged

값이 루프의 마지막 반복에서 변경되었는지 확인하십시오.

블록 태그 루프에서 사용된다. 두 가지 용도가 있습니다.**{% ifchanged %}**

1. 렌더링 된 내용을 이전 상태와 대조하여 내용이 변경된 경우에만 내용을 표시합니다. 예를 들어 일 목록 만 표시되며 변경된 경우 해당 월 만 표시합니다.

언어 : **ko**

문서 버전 : **2.2**

```
<h1>Archive for {{ year }}</h1>

{% for date in days %}
    {% ifchanged %}<h3>{{ date|date:"F" }}</h3>{% endifchanged %}
    <a href="{{ date|date:"M/d"|lower }}/">{{ date|date:"j" }}</a>
{% endfor %}
```

2. 하나 이상의 변수가 주어지면 변수가 변경되었는지 확인하십시오. 예를 들어, 다음은 변경 될 때마다 날짜를 표시하고 시간 또는 날짜가 변경된 경우 시간을 표시합니다.

```
{% for date in days %}
    {% ifchanged date.date %} {{ date.date }} {% endifchanged %}
    {% ifchanged date.hour date.date %}
        {{ date.hour }}
    {% endifchanged %}
{% endfor %}
```

ifchanged태그는 옵션이 걸릴 수 있습니다 값이 변경되지 않은 경우 표시됩니다 절을 :**{% else %}**

```
{% for match in matches %}
    <div style="background-color:
        {% ifchanged match.ballot_id %}
            {% cycle "red" "blue" %}
        {% else %}
            gray
        {% endifchanged %}
    ">{{ match }}</div>
{% endfor %}
```

---

include

템플릿을로드하고 현재 컨텍스트로 렌더링합니다. 이것은 템플릿 내에 다른 템플릿을 "포함"하는 방법입니다.

템플릿 이름은 작은 따옴표 나 큰 따옴표로 변수 또는 하드 코딩 된 (인용 된) 문자열 일 수 있습니다.

이 예제에는 템플릿의 내용이 포함되어 있습니다 **"foo/bar.html"**.

```
{% include "foo/bar.html" %}
```

Normally the template name is relative to the template loader's root directory. A string argument may also be a relative path starting with **./** or **../** as described in the **extends** tag.

This example includes the contents of the template whose name is contained in the variable **template_name**:

```
{% include template_name %}
```

The variable may also be any object with a **render()** method that accepts a context. This allows you to reference a compiled **Template** in your context.

An included template is rendered within the context of the template that includes it. This example produces the output **"Hello, John!"**:

- Context: variable **person** is set to **"John"** and variable **greeting** is set to **"Hello"**.
- Template:

```
{% include "name_snippet.html" %}
```

- The **name_snippet.html** template:

```
{{ greeting }}, {{ person|default:"friend" }}!
```

You can pass additional context to the template using keyword arguments:

```
{% include "name_snippet.html" with person="Jane" greeting="Hello" %}
```

If you want to render the context only with the variables provided (or even no variables at all), use the **only** option. No other variables are available to the included template:

```
{% include "name_snippet.html" with greeting="Hi" only %}
```

Note

The **include** tag should be considered as an implementation of "render this subtemplate and include the HTML", not as "parse this subtemplate and include its contents as if it were part of the parent". This means that there is no shared state between included templates – each include is a completely independent rendering process.

Blocks are evaluated *before* they are included. This means that a template that includes blocks from another will contain blocks that have *already been evaluated and rendered* - not blocks that can be overridden by, for example, an extending template.

---

load

Loads a custom template tag set.

For example, the following template would load all the tags and filters registered in **somelibrary** and **otherlibrary** located in package **package**:

```
{% load somelibrary package.otherlibrary %}
```

You can also selectively load individual filters or tags from a library, using the **from** argument. In this example, the template tags/filters named **foo** and **bar** will be loaded from **somelibrary**:

```
{% load foo bar from somelibrary %}
```

See Custom tag and filter libraries for more information.

---

lorem

Displays random "lorem ipsum" Latin text. This is useful for providing sample data in templates.

Usage:

언어 : **ko**

문서 버전 : **2.2**

```
{% lorem [count] [method] [random] %}
```

The **{% lorem %}** tag can be used with zero, one, two or three arguments. The arguments are:

Argument	Description
count	A number (or variable) containing the number of paragraphs or words to generate (default is 1).
method	Either **w** for words, **p** for HTML paragraphs or **b** for plain-text paragraph blocks (default is **b**).
random	The word **random**, which if given, does not use the common paragraph (“Lorem ipsum dolor sit amet…”) when generating text.

Examples:

- **{% lorem %}** will output the common “lorem ipsum” paragraph.
- **{% lorem 3 p %}** will output the common “lorem ipsum” paragraph and two random paragraphs each wrapped in HTML **<p>** tags.
- **{% lorem 2 w random %}** will output two random Latin words.

---

now

Displays the current date and/or time, using a format according to the given string. Such string can contain format specifiers characters as described in the **date** filter section.

Example:

```
It is {% now "jS F Y H:i" %}
```

Note that you can backslash-escape a format string if you want to use the “raw” value. In this example, both “o” and “f” are backslash-escaped, because otherwise each is a format string that displays the year and the time, respectively:

```
It is the {% now "jS \o\f F" %}
```

This would display as “It is the 4th of September”.

Note

The format passed can also be one of the predefined ones **DATE_FORMAT**, **DATETIME_FORMAT**, **SHORT_DATE_FORMAT** or **SHORT_DATETIME_FORMAT**. The predefined formats may vary depending on the current locale and if Format localization is enabled, e.g.:

```
It is {% now "SHORT_DATETIME_FORMAT" %}
```

You can also use the syntax **{% now "Y" as current_year %}** to store the output (as a string) inside a variable. This is useful if you want to use **{% now %}** inside a template tag like **blocktrans** for example:

```
{% now "Y" as current_year %}
{% blocktrans %}Copyright {{ current_year }}{% endblocktrans %}
```

언어 : **ko**

문서 버전 : **2.2**

regroup

Regroups a list of alike objects by a common attribute.

This complex tag is best illustrated by way of an example: say that **`cities`** is a list of cities represented by dictionaries containing **`"name"`**, **`"population"`**, and **`"country"`** keys:

```
cities = [
    {'name': 'Mumbai', 'population': '19,000,000', 'country': 'India'},
    {'name': 'Calcutta', 'population': '15,000,000', 'country': 'India'},
    {'name': 'New York', 'population': '20,000,000', 'country': 'USA'},
    {'name': 'Chicago', 'population': '7,000,000', 'country': 'USA'},
    {'name': 'Tokyo', 'population': '33,000,000', 'country': 'Japan'},
]
```

...and you'd like to display a hierarchical list that is ordered by country, like this:

- India
  - Mumbai: 19,000,000
  - Calcutta: 15,000,000
- USA
  - New York: 20,000,000
  - Chicago: 7,000,000
- Japan
  - Tokyo: 33,000,000

You can use the **`{% regroup %}`** tag to group the list of cities by country. The following snippet of template code would accomplish this:

```
{% regroup cities by country as country_list %}

<ul>
{% for country in country_list %}
    <li>{{ country.grouper }}
    <ul>
        {% for city in country.list %}
          <li>{{ city.name }}: {{ city.population }}</li>
        {% endfor %}
    </ul>
    </li>
{% endfor %}
</ul>
```

Let's walk through this example. **`{% regroup %}`** takes three arguments: the list you want to regroup, the attribute to group by, and the name of the resulting list. Here, we're regrouping the **`cities`** list by the **`country`** attribute and calling the result **`country_list`**.

`{% regroup %}` produces a list (in this case, **`country_list`**) of **group objects**. Group objects are instances of **`namedtuple()`** with two fields:

- **`grouper`** – the item that was grouped by (e.g., the string "India" or "Japan").
- **`list`** – a list of all items in this group (e.g., a list of all cities with country='India').

Because **`{% regroup %}`** produces **`namedtuple()`** objects, you can also write the previous example as:

```
{% regroup cities by country as country_list %}

<ul>
{% for country, local_cities in country_list %}
    <li>{{ country }}
    <ul>
        {% for city in local_cities %}
          <li>{{ city.name }}: {{ city.population }}</li>
        {% endfor %}
    </ul>
    </li>
{% endfor %}
</ul>
```

Note that **{% regroup %}** does not order its input! Our example relies on the fact that the **cities** list was ordered by **country** in the first place. If the **cities** list did *not* order its members by **country**, the regrouping would naively display more than one group for a single country. For example, say the **cities** list was set to this (note that the countries are not grouped together):

```
cities = [
    {'name': 'Mumbai', 'population': '19,000,000', 'country': 'India'},
    {'name': 'New York', 'population': '20,000,000', 'country': 'USA'},
    {'name': 'Calcutta', 'population': '15,000,000', 'country': 'India'},
    {'name': 'Chicago', 'population': '7,000,000', 'country': 'USA'},
    {'name': 'Tokyo', 'population': '33,000,000', 'country': 'Japan'},
]
```

With this input for **cities**, the example **{% regroup %}** template code above would result in the following output:

- India
  - Mumbai: 19,000,000
- USA
  - New York: 20,000,000
- India
  - Calcutta: 15,000,000
- USA
  - Chicago: 7,000,000
- Japan
  - Tokyo: 33,000,000

The easiest solution to this gotcha is to make sure in your view code that the data is ordered according to how you want to display it.

Another solution is to sort the data in the template using the **dictsort** filter, if your data is in a list of dictionaries:

```
{% regroup cities|dictsort:"country" by country as country_list %}
```

Grouping on other properties

Any valid template lookup is a legal grouping attribute for the regroup tag, including methods, attributes, dictionary keys and list items. For example, if the “country” field is a foreign key to a class with an attribute “description,” you could use:

```
{% regroup cities by country.description as country_list %}
```

언어 : **ko**

문서 버전 : **2.2**

Or, if the field is a field with **choices**, it will have a **get_FOO_display()** method available as an attribute, allowing you to group on the display string rather than the **choices** key:

```
{% regroup cities by get_country_display as country_list %}
```

{{ country.grouper }} will now display the value fields from the **choices** set rather than the keys.

---

resetcycle

Resets a previous cycle so that it restarts from its first item at its next encounter. Without arguments, **{% resetcycle %}** will reset the last **{% cycle %}** defined in the template.

Example usage:

```
{% for coach in coach_list %}
    <h1>{{ coach.name }}</h1>
    {% for athlete in coach.athlete_set.all %}
        <p class="{% cycle 'odd' 'even' %}">{{ athlete.name }}</p>
    {% endfor %}
    {% resetcycle %}
{% endfor %}
```

This example would return this HTML:

```
<h1>José Mourinho</h1>
<p class="odd">Thibaut Courtois</p>
<p class="even">John Terry</p>
<p class="odd">Eden Hazard</p>

<h1>Carlo Ancelotti</h1>
<p class="odd">Manuel Neuer</p>
<p class="even">Thomas Müller</p>
```

Notice how the first block ends with **class="odd"** and the new one starts with **class="odd"**. Without the **{% resetcycle %}** tag, the second block would start with **class="even"**.

You can also reset named cycle tags:

```
{% for item in list %}
    <p class="{% cycle 'odd' 'even' as stripe %} {% cycle 'major' 'minor' 'minor' 'minor' 'minor' as tick %}">
        {{ item.data }}
    </p>
    {% ifchanged item.category %}
        <h1>{{ item.category }}</h1>
        {% if not forloop.first %}{% resetcycle tick %}{% endif %}
    {% endifchanged %}
{% endfor %}
```

In this example, we have both the alternating odd/even rows and a “major” row every fifth row. Only the five-row cycle is reset when a category changes.

언어 : **ko**

문서 버전 : **2.2**

---

spaceless

Removes whitespace between HTML tags. This includes tab characters and newlines.

Example usage:

```
{% spaceless %}
    <p>
        <a href="foo/">Foo</a>
    </p>
{% endspaceless %}
```

This example would return this HTML:

```
<p><a href="foo/">Foo</a></p>
```

Only space between *tags* is removed – not space between tags and text. In this example, the space around **`Hello`** won't be stripped:

```
{% spaceless %}
    <strong>
        Hello
    </strong>
{% endspaceless %}
```

---

templatetag

Outputs one of the syntax characters used to compose template tags.

Since the template system has no concept of "escaping", to display one of the bits used in template tags, you must use the **`{% templatetag %}`** tag.

The argument tells which template bit to output:

Argument	Outputs
`openblock`	`{%`
`closeblock`	`%}`
`openvariable`	`{{`
`closevariable`	`}}`
`openbrace`	`{`
`closebrace`	`}`
`opencomment`	`{#`
`closecomment`	`#}`

Sample usage:

```
{% templatetag openblock %} url 'entry_list' {% templatetag closeblock %}
```

---

url

Returns an absolute path reference (a URL without the domain name) matching a given view and optional parameters. Any special characters in the resulting path will be encoded using **`iri_to_uri()`**.

This is a way to output links without violating the DRY principle by having to hard-code URLs in your templates:

언어 : **ko**

문서 버전 : **2.2**

```
{% url 'some-url-name' v1 v2 %}
```

The first argument is a URL pattern name. It can be a quoted literal or any other context variable. Additional arguments are optional and should be space-separated values that will be used as arguments in the URL. The example above shows passing positional arguments. Alternatively you may use keyword syntax:

```
{% url 'some-url-name' arg1=v1 arg2=v2 %}
```

Do not mix both positional and keyword syntax in a single call. All arguments required by the URLconf should be present.

For example, suppose you have a view, **app_views.client**, whose URLconf takes a client ID (here, **client()** is a method inside the views file **app_views.py**). The URLconf line might look like this:

```
path('client/<int:id>/', app_views.client, name='app-views-client')
```

If this app's URLconf is included into the project's URLconf under a path such as this:

```
path('clients/', include('project_name.app_name.urls'))
```

...then, in a template, you can create a link to this view like this:

```
{% url 'app-views-client' client.id %}
```

The template tag will output the string **/clients/client/123/**.

Note that if the URL you're reversing doesn't exist, you'll get an **NoReverseMatch** exception raised, which will cause your site to display an error page.

If you'd like to retrieve a URL without displaying it, you can use a slightly different call:

```
{% url 'some-url-name' arg arg2 as the_url %}

<a href="{{ the_url }}">I'm linking to {{ the_url }}</a>
```

The scope of the variable created by the **as var** syntax is the **{% block %}** in which the **{% url %}** tag appears.

This **{% url ... as var %}** syntax will *not* cause an error if the view is missing. In practice you'll use this to link to views that are optional:

```
{% url 'some-url-name' as the_url %}
{% if the_url %}
  <a href="{{ the_url }}">Link to optional stuff</a>
{% endif %}
```

If you'd like to retrieve a namespaced URL, specify the fully qualified name:

```
{% url 'myapp:view-name' %}
```

This will follow the normal namespaced URL resolution strategy, including using any hints provided by the context as to the current application.

언어 : **ko**

문서 버전 : **2.2**

Warning

Don't forget to put quotes around the URL pattern **name**, otherwise the value will be interpreted as a context variable!

---

verbatim

Stops the template engine from rendering the contents of this block tag.

A common use is to allow a JavaScript template layer that collides with Django's syntax. For example:

```
{% verbatim %}
    {{if dying}}Still alive.{{/if}}
{% endverbatim %}
```

You can also designate a specific closing tag, allowing the use of **{% endverbatim %}** as part of the unrendered contents:

```
{% verbatim myblock %}
    Avoid template rendering via the {% verbatim %}{% endverbatim %} block.
{% endverbatim myblock %}
```

---

widthratio

For creating bar charts and such, this tag calculates the ratio of a given value to a maximum value, and then applies that ratio to a constant.

For example:

```
<img src="bar.png" alt="Bar"
     height="10" width="{% widthratio this_value max_value max_width %}">
```

If **this_value** is 175, **max_value** is 200, and **max_width** is 100, the image in the above example will be 88 pixels wide (because 175/200 = .875; .875 * 100 = 87.5 which is rounded up to 88).

In some cases you might want to capture the result of **widthratio** in a variable. It can be useful, for instance, in a **blocktrans** like this:

```
{% widthratio this_value max_value max_width as width %}
{% blocktrans %}The width is: {{ width }}{% endblocktrans %}
```

---

with

Caches a complex variable under a simpler name. This is useful when accessing an "expensive" method (e.g., one that hits the database) multiple times.

For example:

```
{% with total=business.employees.count %}
    {{ total }} employee{{ total|pluralize }}
{% endwith %}
```

The added variable (in the example above, **total**) is only available between the **{% with %}** and **{% endwith %}** tags.

You can assign more than one context variable:

```
{% with alpha=1 beta=2 %}
    ...
{% endwith %}
```

Note

The previous more verbose format is still supported: **{% with business.employees.count as total %}**

Built-in filter reference

add

Adds the argument to the value.

For example:

```
{{ value|add:"2" }}
```

If **value** is **4**, then the output will be **6**.

This filter will first try to coerce both values to integers. If this fails, it'll attempt to add the values together anyway. This will work on some data types (strings, list, etc.) and fail on others. If it fails, the result will be an empty string.

For example, if we have:

```
{{ first|add:second }}
```

and **first** is **[1, 2, 3]** and **second** is **[4, 5, 6]**, then the output will be **[1, 2, 3, 4, 5, 6]**.

Warning

Strings that can be coerced to integers will be **summed**, not concatenated, as in the first example above.

---

addslashes

Adds slashes before quotes. Useful for escaping strings in CSV, for example.

For example:

```
{{ value|addslashes }}
```

If **value** is **"I'm using Django"**, the output will be **"I\'m using Django"**.

언어 : **ko**

문서 버전 : **2.2**

capfirst

Capitalizes the first character of the value. If the first character is not a letter, this filter has no effect.

For example:

```
{{ value|capfirst }}
```

If **value** is **"django"**, the output will be **"Django"**.

---

center

Centers the value in a field of a given width.

For example:

```
"{{ value|center:"15" }}"
```

If **value** is **"Django"**, the output will be **" Django "**.

---

cut

Removes all values of arg from the given string.

For example:

```
{{ value|cut:" " }}
```

If **value** is **"String with spaces"**, the output will be **"Stringwithspaces"**.

---

date

Formats a date according to the given format.

Uses a similar format as PHP's **date()** function (https://php.net/date) with some differences.

Note

These format characters are not used in Django outside of templates. They were designed to be compatible with PHP to ease transitioning for designers.

Available format strings:

Format character	Description	Example output
Day		
d	Day of the month, 2 digits with leading zeros.	**'01'** to **'31'**
j	Day of the month without leading zeros.	**'1'** to **'31'**
D	Day of the week, textual, 3 letters.	**'Fri'**
l	Day of the week, textual, long.	**'Friday'**

Format character	Description	Example output
S	English ordinal suffix for day of the month, 2 characters.	**'st'**, **'nd'**, **'rd'** or **'th'**
w	Day of the week, digits without leading zeros.	**'0'** (Sunday) to **'6'** (Saturday)
z	Day of the year.	**0** to **365**
Week		
W	ISO-8601 week number of year, with weeks starting on Monday.	**1**, **53**
Month		
m	Month, 2 digits with leading zeros.	**'01'** to **'12'**
n	Month without leading zeros.	**'1'** to **'12'**
M	Month, textual, 3 letters.	**'Jan'**
b	Month, textual, 3 letters, lowercase.	**'jan'**
E	Month, locale specific alternative representation usually used for long date representation.	**'listopada'** (for Polish locale, as opposed to **'Listopad'**)
F	Month, textual, long.	**'January'**
N	Month abbreviation in Associated Press style. Proprietary extension.	**'Jan.'**, **'Feb.'**, **'March'**, **'May'**
t	Number of days in the given month.	**28** to **31**
Year		
y	Year, 2 digits.	**'99'**
Y	Year, 4 digits.	**'1999'**
L	Boolean for whether it's a leap year.	**True** or **False**
o	ISO-8601 week-numbering year, corresponding to the ISO-8601 week number (W) which uses leap weeks. See Y for the more common year format.	**'1999'**
Time		
g	Hour, 12-hour format without leading zeros.	**'1'** to **'12'**
G	Hour, 24-hour format without leading zeros.	**'0'** to **'23'**
h	Hour, 12-hour format.	**'01'** to **'12'**
H	Hour, 24-hour format.	**'00'** to **'23'**
i	Minutes.	**'00'** to **'59'**
s	Seconds, 2 digits with leading zeros.	**'00'** to **'59'**
u	Microseconds.	**000000** to **999999**
a	**'a.m.'** or **'p.m.'** (Note that this is slightly different than PHP's output, because this includes periods to match Associated Press style.)	**'a.m.'**
A	**'AM'** or **'PM'**.	**'AM'**
f	Time, in 12-hour hours and minutes, with minutes left off if they're zero. Proprietary extension.	**'1'**, **'1:30'**
P	Time, in 12-hour hours, minutes and 'a.m.'/'p.m.', with minutes left off if they're zero and the special-case strings 'midnight' and 'noon' if appropriate. Proprietary extension.	**'1 a.m.'**, **'1:30 p.m.'**, **'midnight'**, **'noon'**, **'12:30 p.m.'**
Timezone		
e	Timezone name. Could be in any format, or might return an empty string, depending on the datetime.	**''**, **'GMT'**, **'-500'**, **'US/Eastern'**, etc.
I	Daylight Savings Time, whether it's in effect or not.	**'1'** or **'0'**
O	Difference to Greenwich time in hours.	**'+0200'**
T	Time zone of this machine.	**'EST'**, **'MDT'**
Z	Time zone offset in seconds. The offset for timezones west of UTC is always negative, and for those east of UTC is always positive.	**-43200** to **43200**
Date/Time		
c	ISO 8601 format. (Note: unlike others formatters, such as "Z", "O" or "r", the "c" formatter will not add timezone offset if value is a naive datetime (see **datetime.tzinfo**).	**2008-01-02T10:30:00.000123+02:00**, or **2008-01-02T10:30:00.000123** if the datetime is naive
r	**RFC 5322** formatted date.	**'Thu, 21 Dec 2000 16:01:07 +0200'**
U	Seconds since the Unix Epoch (January 1 1970 00:00:00 UTC).	

```
{{ value|date:"D d M Y" }}
```

If **value** is a **datetime** object (e.g., the result of **datetime.datetime.now()**), the output will be the string **'Wed 09 Jan 2008'**.

The format passed can be one of the predefined ones **DATE_FORMAT**, **DATETIME_FORMAT**, **SHORT_DATE_FORMAT** or **SHORT_DATETIME_FORMAT**, or a custom format that uses the format specifiers shown in the table above. Note that predefined formats may vary depending on the current locale.

Assuming that **USE_L10N** is **True** and **LANGUAGE_CODE** is, for example, **"es"**, then for:

```
{{ value|date:"SHORT_DATE_FORMAT" }}
```

the output would be the string **"09/01/2008"** (the **"SHORT_DATE_FORMAT"** format specifier for the **es** locale as shipped with Django is **"d/m/Y"**).

When used without a format string, the **DATE_FORMAT** format specifier is used. Assuming the same settings as the previous example:

```
{{ value|date }}
```

outputs **9 de Enero de 2008** (the **DATE_FORMAT** format specifier for the **es** locale is **r'j \d\e F \d\e Y'**.

You can combine **date** with the **time** filter to render a full representation of a **datetime** value. E.g.:

```
{{ value|date:"D d M Y" }} {{ value|time:"H:i" }}
```

---

default

If value evaluates to **False**, uses the given default. Otherwise, uses the value.

For example:

```
{{ value|default:"nothing" }}
```

If **value** is **""** (the empty string), the output will be **nothing**.

---

default_if_none

If (and only if) value is **None**, uses the given default. Otherwise, uses the value.

Note that if an empty string is given, the default value will *not* be used. Use the **default** filter if you want to fallback for empty strings.

For example:

```
{{ value|default_if_none:"nothing" }}
```

If **value** is **None**, the output will be **nothing**.

dictsort

Takes a list of dictionaries and returns that list sorted by the key given in the argument.

For example:

```
{{ value|dictsort:"name" }}
```

If **value** is:

```
[
    {'name': 'zed', 'age': 19},
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
]
```

then the output would be:

```
[
    {'name': 'amy', 'age': 22},
    {'name': 'joe', 'age': 31},
    {'name': 'zed', 'age': 19},
]
```

You can also do more complicated things like:

```
{% for book in books|dictsort:"author.age" %}
    * {{ book.title }} ({{ book.author.name }})
{% endfor %}
```

If **books** is:

```
[
    {'title': '1984', 'author': {'name': 'George', 'age': 45}},
    {'title': 'Timequake', 'author': {'name': 'Kurt', 'age': 75}},
    {'title': 'Alice', 'author': {'name': 'Lewis', 'age': 33}},
]
```

then the output would be:

```
* Alice (Lewis)
* 1984 (George)
* Timequake (Kurt)
```

dictsort can also order a list of lists (or any other object implementing **__getitem__()**) by elements at specified index. For example:

언어 : **ko**

문서 버전 : **2.2**

```
{{ value|dictsort:0 }}
```

If `value` is:

```
[
    ('a', '42'),
    ('c', 'string'),
    ('b', 'foo'),
]
```

then the output would be:

```
[
    ('a', '42'),
    ('b', 'foo'),
    ('c', 'string'),
]
```

You must pass the index as an integer rather than a string. The following produce empty output:

```
{{ values|dictsort:"0" }}
```

`dictsortreversed`

Takes a list of dictionaries and returns that list sorted in reverse order by the key given in the argument. This works exactly the same as the above filter, but the returned value will be in reverse order.

`divisibleby`

Returns **`True`** if the value is divisible by the argument.

For example:

```
{{ value|divisibleby:"3" }}
```

If **`value`** is **`21`**, the output would be **`True`**.

`escape`

Escapes a string's HTML. Specifically, it makes these replacements:

- **`<`** is converted to **`<`**
- **`>`** is converted to **`>`**
- **`'`** (single quote) is converted to **`&#39;`**
- **`"`** (double quote) is converted to **`&quot;`**
- **`&`** is converted to **`&`**

Applying **`escape`** to a variable that would normally have auto-escaping applied to the result will only result in one round of escaping being done. So it is safe to use this function even in auto-escaping environments. If you want multiple escaping passes to be applied, use the **`force_escape`** filter.

For example, you can apply **`escape`** to fields when **`autoescape`** is off:

```
{% autoescape off %}
    {{ title|escape }}
{% endautoescape %}
```

escapejs

Escapes characters for use in JavaScript strings. This does *not* make the string safe for use in HTML or JavaScript template literals, but does protect you from syntax errors when using templates to generate JavaScript/JSON.

For example:

```
{{ value|escapejs }}
```

If **value** is **"testing\r\njavascript \'string" escaping"**, the output will be **"testing\\u000D\\u000Ajavascript \\u0027string\\u0022 \\u003Cb\\u003Eescaping\\u003C/b\\u003E"**.

filesizeformat

Formats the value like a 'human-readable' file size (i.e. **'13 KB'**, **'4.1 MB'**, **'102 bytes'**, etc.).

For example:

```
{{ value|filesizeformat }}
```

If **value** is 123456789, the output would be **117.7 MB**.

File sizes and SI units

Strictly speaking, **filesizeformat** does not conform to the International System of Units which recommends using KiB, MiB, GiB, etc. when byte sizes are calculated in powers of 1024 (which is the case here). Instead, Django uses traditional unit names (KB, MB, GB, etc.) corresponding to names that are more commonly used.

first

Returns the first item in a list.

For example:

```
{{ value|first }}
```

If **value** is the list **['a', 'b', 'c']**, the output will be **'a'**.

floatformat

언어 : **ko**

문서 버전 : **2.2**

When used without an argument, rounds a floating-point number to one decimal place – but only if there's a decimal part to be displayed. For example:

value	Template	Output

value	Template	Output
34.23234	{{ value|floatformat }}	34.2
34.00000	{{ value|floatformat }}	34
34.26000	{{ value|floatformat }}	34.3

숫자 형 정수 인수와 함께 사용되는 경우, **floatformat**소수점 이하를 소수점 이하로 반올림합니다. 예 :

value	주형	산출
34.23234	{{ value|floatformat:3 }}	34.232
34.00000	{{ value|floatformat:3 }}	34.000
34.26000	{{ value|floatformat:3 }}	34.260

float를 가장 가까운 정수로 반올림하는 인수로 0을 전달하는 것이 특히 유용합니다.

value	주형	산출
34.23234	{{ value|floatformat:"0" }}	34
34.00000	{{ value|floatformat:"0" }}	34
39.56000	{{ value|floatformat:"0" }}	40

전달 된 인수 **floatformat**가 음수이면 숫자를 해당 소수 자리로 반올림합니다. 단, 표시 할 소수 부분 만있는 경우에만 해당합니다. 예 :

value	주형	산출
34.23234	{{ value|floatformat:"-3" }}	34.232
34.00000	{{ value|floatformat:"-3" }}	34
34.26000	{{ value|floatformat:"-3" }}	34.260

사용 **floatformat**인수없이하는 것은 사용하는 것과 같습니다 **floatformat** 의 인수 **-1**.

---

force_escape

HTML 이스케이프를 문자열에 적용합니다 (자세한 내용은 **escape**필터 참조). 이 필터는 *즉시*적용 되고 이스케이프 된 새 문자열을 반환합니다. 이는 여러 개의 이스케이프가 필요하거나 이스케이프 된 결과에 다른 필터를 적용하려는 드문 경우에 유용합니다. 일반적으로 **escape**필터 를 사용하려고합니다 .

예를 들어 필터로 **<p>**만든 HTML 요소 를 잡으려면 다음과 같이 하십시오 **linebreaks**.

```
{% autoescape off %}
    {{ body|linebreaks|force_escape }}
{% endautoescape %}
```

---

get_digit

주어진 숫자를 받으면 요청 된 숫자를 반환합니다. 여기서 1은 가장 오른쪽 숫자이고, 2는 오른쪽에서 두 번째 숫자입니다. 잘못된 입력에 대한 원래 값을 반환합니다 (입력 또는 인수가 정수가 아니면 인수가 1보다 작음). 그렇지 않으면 출력은 항상 정수입니다.

예 :

```
{{ value|get_digit:"2" }}
```

경우 **value**이며 **123456789**, 출력이 될 것입니다 **8**.

---

iriencode

IRI (Internationalized Resource Identifier)를 URL에 포함하기에 적합한 문자열로 변환합니다. 이것은 URL에 비 ASCII 문자가 포함 된 문자열을 사용하려는 경우에 필요합니다.

언어 : **ko**

문서 버전 : **2.2**

아마도 **iriencode**필터를 통과 한 문자열에이 필터를 사용하는 것이 안전합니다.

예 :

```
{{ value|iriencode }}
```

경우 **value**이며 **"?test=1&me=2"**, 출력이 될 것입니다 **"?test=1&me=2"**.

---

join

파이썬과 같이 문자열로 목록을 조인합니다. **str.join(list)**

예 :

```
{{ value|join:" // " }}
```

if **value**가 목록 이면 결과는 문자열 입니다.**['a', 'b', 'c']"a // b // c"**

---

json_script

장고 2.1의 새로운 기능 :

안전하게 파이썬 객체를 JSON으로 출력하고, **<script>**태그로 싸서 JavaScript와 함께 사용할 준비가되었습니다.

인수 :<script> 태그 의 HTML "id" .

예 :

```
{{ value|json_script:"hello-data" }}
```

경우 **value**사전이며 , 출력은 다음과 같습니다**{'hello': 'world'}**

```
<script id="hello-data" type="application/json">{"hello": "world"}</script>
```

결과 데이터는 다음과 같이 JavaScript로 액세스 할 수 있습니다.

```
var value = JSON.parse(document.getElementById('hello-data').textContent);
```

XSS 공격은 "<", ">"및 "&".자를 이스케이프 처리하여 완화됩니다. 예를 들어 **value**이며 , 출력은이다 :**{'hello': 'world</script>&'}**

```
<script id="hello-data" type="application/json">{"hello": "world\\u003C/script\\u003E\\u0026amp;"}</script>
```

이는 인 페이지 스크립트 실행을 금지하는 엄격한 콘텐츠 보안 정책과 호환됩니다. 또한 패시브 데이터와 실행 코드를 명확하게 구분합니다.

언어 : **ko**

문서 버전 : **2.2**

last

목록의 마지막 항목을 반환합니다.

예 :

```
{{ value|last }}
```

if **value**가 목록 이면 결과는 문자열 입니다.**['a', 'b', 'c', 'd']"d"**

---

length

값의 길이를 리턴합니다. 이것은 문자열과 목록 모두에서 작동합니다.

예 :

```
{{ value|length }}
```

If **value**가 or 이면 출력이됩니다 .**['a', 'b', 'c', 'd']"abcd"4**

필터는 **0**정의되지 않은 변수를 반환 합니다.

---

length_is

True값의 길이가 인수 일지 어떨지를 돌려 **False**줍니다.

예 :

```
{{ value|length_is:"4" }}
```

If **value**가 or 이면 출력이됩니다 .**['a', 'b', 'c', 'd']"abcd"True**

---

linebreaks

일반 텍스트의 줄 바꿈을 적절한 HTML로 바꿉니다. 한 줄 바꿈이 HTML 줄 바꿈 (**
**)이되고 새 줄 다음에 빈 줄이 있으면 단락 나누기 (** **)가됩니다.

예 :

```
{{ value|linebreaks }}
```

경우 **value**이며 , 출력이 될 것입니다 .**Joel\nis a slug Joel
is a slug **

---

linebreaksbr

일반 텍스트의 모든 줄 바꿈을 HTML 줄 바꿈 (**
**)으로 변환합니다.

예 :

언어 : **ko**

문서 버전 : **2.2**

```
{{ value|linebreaksbr }}
```

경우 **value**이며 , 출력이 될 것입니다 .**Joel\nis a slugJoel
is a slug**

---

linenumbers

줄 번호가있는 텍스트를 표시합니다.

예 :

```
{{ value|linenumbers }}
```

있는 경우 **value**:

```
one
two
three
```

출력은 다음과 같습니다.

```
1. one
2. two
3. three
```

---

ljust

주어진 너비의 필드에서 값을 왼쪽 정렬합니다.

인수 : 필드 크기

예 :

```
"{{ value|ljust:"10" }}"
```

경우 **value**이며 **Django**, 출력이 될 것입니다 .**"Django   "**

---

lower

문자열을 모두 소문자로 변환합니다.

예 :

```
{{ value|lower }}
```

경우 **value**이며 , 출력이 될 것입니다 .**Totally LOVING this Album!totally loving this album!**

언어 : **ko**

---

make_list

문서 버전 : **2.2**

리스트로 변환 된 값을 리턴합니다. 문자열의 경우 문자 목록입니다. 정수의 경우 인수는 목록을 만들기 전에 문자열로 변환됩니다.

```
{{ value|make_list }}
```

if **value**가 문자열 **"Joel"**이면 출력이 목록 입니다. 경우 이며 , 출력은 목록이 될 것입니다 .**['J', 'o', 'e', 'l']value123['1', '2', '3']**

---

phone2numeric

전화 번호 (문자 포함 가능)를 해당 숫자로 변환합니다.

입력이 유효한 전화 번호 일 필요는 없습니다. 이것은 행복하게 모든 문자열을 변환합니다.

예 :

```
{{ value|phone2numeric }}
```

경우 **value**이며 **800-COLLECT**, 출력이 될 것입니다 **800-2655328**.

---

pluralize

값이없는 경우 복수 접미사를 돌려 **1**, **'1'**또는 기본적으로 길이 1의 목적은,이 접미사입니다 **'s'**.

예:

```
You have {{ num_messages }} message{{ num_messages|pluralize }}.
```

경우 **num_messages**이며 **1**, 출력이 될 것입니다 경우 입니다 출력이 될 것입니다**You have 1 message.num_messages2You have 2 messages.**

이외의 접미사가 필요한 단어 인 **'s'**경우 대체 접미어를 매개 변수로 필터에 제공 할 수 있습니다.

예:

```
You have {{ num_walruses }} walrus{{ num_walruses|pluralize:"es" }}.
```

For words that don't pluralize by simple suffix, you can specify both a singular and plural suffix, separated by a comma.

Example:

```
You have {{ num_cherries }} cherr{{ num_cherries|pluralize:"y,ies" }}.
```

Note

Use **blocktrans** to pluralize translated strings.

언어 : **ko**

---

pprint

문서 버전 : **2.2**

A wrapper around **pprint.pprint()** – for debugging, really.

random

Returns a random item from the given list.

For example:

```
{{ value|random }}
```

If **value** is the list **['a', 'b', 'c', 'd']**, the output could be **"b"**.

---

rjust

Right-aligns the value in a field of a given width.

Argument: field size

For example:

```
"{{ value|rjust:"10" }}"
```

If **value** is **Django**, the output will be **" Django"**.

---

safe

Marks a string as not requiring further HTML escaping prior to output. When autoescaping is off, this filter has no effect.

Note

If you are chaining filters, a filter applied after **safe** can make the contents unsafe again. For example, the following code prints the variable as is, unescaped:

```
{{ var|safe|escape }}
```

---

safeseq

Applies the **safe** filter to each element of a sequence. Useful in conjunction with other filters that operate on sequences, such as **join**. For example:

```
{{ some_list|safeseq|join:", " }}
```

You couldn't use the **safe** filter directly in this case, as it would first convert the variable into a string, rather than working with the individual elements of the sequence.

---

slice

Returns a slice of the list.

Uses the same syntax as Python's list slicing. See https://www.diveinto.org/python3/native-datatypes.html#slicinglists for an introduction.

```
{{ some_list|slice:":2" }}
```

If **some_list** is **['a', 'b', 'c']**, the output will be **['a', 'b']**.

---

slugify

Converts to ASCII. Converts spaces to hyphens. Removes characters that aren't alphanumerics, underscores, or hyphens. Converts to lowercase. Also strips leading and trailing whitespace.

For example:

```
{{ value|slugify }}
```

If **value** is **"Joel is a slug"**, the output will be **"joel-is-a-slug"**.

---

stringformat

Formats the variable according to the argument, a string formatting specifier. This specifier uses the printf-style String Formatting syntax, with the exception that the leading "%" is dropped.

For example:

```
{{ value|stringformat:"E" }}
```

If **value** is **10**, the output will be **1.000000E+01**.

---

striptags

Makes all possible efforts to strip all [X]HTML tags.

For example:

```
{{ value|striptags }}
```

If **value** is **"Joel <button>is</button> a slug"**, the output will be **"Joel is a slug"**.

No safety guarantee

Note that **striptags** doesn't give any guarantee about its output being HTML safe, particularly with non valid HTML input. So **NEVER** apply the **safe** filter to a **striptags** output. If you are looking for something more robust, you can use the **bleach** Python library, notably its clean method.

언어 : **ko**

문서 버전 : **2.2**

---

time

Formats a time according to the given format.

Given format can be the predefined one **TIME_FORMAT**, or a custom format, same as the **date** filter. Note that the predefined format is locale-dependent.

For example:

```
{{ value|time:"H:i" }}
```

If **value** is equivalent to **datetime.datetime.now()**, the output will be the string **"01:23"**.

Another example:

Assuming that **USE_L10N** is **True** and **LANGUAGE_CODE** is, for example, **"de"**, then for:

```
{{ value|time:"TIME_FORMAT" }}
```

the output will be the string **"01:23"** (The **"TIME_FORMAT"** format specifier for the **de** locale as shipped with Django is **"H:i"**).

The **time** filter will only accept parameters in the format string that relate to the time of day, not the date (for obvious reasons). If you need to format a **date** value, use the **date** filter instead (or along **time** if you need to render a full **datetime** value).

There is one exception the above rule: When passed a **datetime** value with attached timezone information (a time-zone-aware **datetime** instance) the **time** filter will accept the timezone-related format specifiers **'e'**, **'O'** , **'T'** and **'Z'**.

When used without a format string, the **TIME_FORMAT** format specifier is used:

```
{{ value|time }}
```

is the same as:

```
{{ value|time:"TIME_FORMAT" }}
```

---

timesince

Formats a date as the time since that date (e.g., “4 days, 6 hours”).

Takes an optional argument that is a variable containing the date to use as the comparison point (without the argument, the comparison point is *now*). For example, if **blog_date** is a date instance representing midnight on 1 June 2006, and **comment_date** is a date instance for 08:00 on 1 June 2006, then the following would return “8 hours”:

```
{{ blog_date|timesince:comment_date }}
```

Comparing offset-naive and offset-aware datetimes will return an empty string.

Minutes is the smallest unit used, and “0 minutes” will be returned for any date that is in the future relative to the comparison point.

---

timeuntil

Similar to **timesince**, except that it measures the time from now until the given date or datetime. For example, if today is 1 June 2006 and **conference_date** is a date instance holding 29 June 2006, then **{{ conference_date|timeuntil }}** will return “4 weeks”.

Takes an optional argument that is a variable containing the date to use as the comparison point (instead of *now*). If **from_date** contains 22 June 2006, then the following will return “1 week”:

```
{{ conference_date|timeuntil:from_date }}
```

Comparing offset-naive and offset-aware datetimes will return an empty string.

Minutes is the smallest unit used, and “0 minutes” will be returned for any date that is in the past relative to the comparison point.

`title`

Converts a string into titlecase by making words start with an uppercase character and the remaining characters lowercase. This tag makes no effort to keep “trivial words” in lowercase.

For example:

```
{{ value|title }}
```

If **`value`** is **`"my FIRST post"`**, the output will be **`"My First Post"`**.

`truncatechars`

Truncates a string if it is longer than the specified number of characters. Truncated strings will end with a translatable ellipsis character (“…”).

Argument: Number of characters to truncate to

For example:

```
{{ value|truncatechars:7 }}
```

If **`value`** is **`"Joel is a slug"`**, the output will be **`"Joel i…"`**.

`truncatechars_html`

Similar to **`truncatechars`**, except that it is aware of HTML tags. Any tags that are opened in the string and not closed before the truncation point are closed immediately after the truncation.

For example:

```
{{ value|truncatechars_html:7 }}
```

If **`value`** is **`"<p>Joel is a slug</p>"`**, the output will be **`"<p>Joel i…</p>"`**.

Newlines in the HTML content will be preserved.

`truncatewords`

Truncates a string after a certain number of words.

Argument: Number of words to truncate after

For example:

```
{{ value|truncatewords:2 }}
```

If **value** is **"Joel is a slug"**, the output will be **"Joel is …"**.

Newlines within the string will be removed.

---

truncatewords_html

Similar to **truncatewords**, except that it is aware of HTML tags. Any tags that are opened in the string and not closed before the truncation point, are closed immediately after the truncation.

This is less efficient than **truncatewords**, so should only be used when it is being passed HTML text.

For example:

```
{{ value|truncatewords_html:2 }}
```

If **value** is **" Joel is a slug "**, the output will be **" Joel is … "**.

Newlines in the HTML content will be preserved.

---

unordered_list

Recursively takes a self-nested list and returns an HTML unordered list – WITHOUT opening and closing <ul> tags.

The list is assumed to be in the proper format. For example, if **var** contains **['States', ['Kansas', ['Lawrence', 'Topeka'], 'Illinois']]**, then **{{ var|unordered_list }}** would return:

```
<li>States
<ul>
        <li>Kansas
        <ul>
                <li>Lawrence</li>
                <li>Topeka</li>
        </ul>
        </li>
        <li>Illinois</li>
</ul>
</li>
```

---

upper

Converts a string into all uppercase.

For example:

```
{{ value|upper }}
```

If **value** is **"Joel is a slug"**, the output will be **"JOEL IS A SLUG"**.

---

urlencode

Escapes a value for use in a URL.

For example:

언어 : **ko**

문서 버전 : **2.2**

```
{{ value|urlencode }}
```

If **value** is **"https://www.example.org/foo?a=b&c=d"**, the output will be **"https%3A//www.example.org/foo%3Fa%3Db%26c%3Dd"**.

An optional argument containing the characters which should not be escaped can be provided.

If not provided, the '/' character is assumed safe. An empty string can be provided when *all* characters should be escaped. For example:

```
{{ value|urlencode:"" }}
```

If **value** is **"https://www.example.org/"**, the output will be **"https%3A%2F%2Fwww.example.org%2F"**.

---

urlize

Converts URLs and email addresses in text into clickable links.

This template tag works on links prefixed with **http://**, **https://**, or **www.**. For example, **https://goo.gl/aia1t** will get converted but **goo.gl/aia1t** won't.

It also supports domain-only links ending in one of the original top level domains (**.com**, **.edu**, **.gov**, **.int**, **.mil**, **.net**, and **.org**). For example, **djangoproject.com** gets converted.

Links can have trailing punctuation (periods, commas, close-parens) and leading punctuation (opening parens), and **urlize** will still do the right thing.

Links generated by **urlize** have a **rel="nofollow"** attribute added to them.

For example:

```
{{ value|urlize }}
```

If **value** is **"Check out www.djangoproject.com"**, the output will be **"Check out www.djangoproject.com"**.

In addition to web links, **urlize** also converts email addresses into **mailto:** links. If **value** is **"Send questions to foo@example.com"**, the output will be **"Send questions to foo@example.com"**.

The **urlize** filter also takes an optional parameter **autoescape**. If **autoescape** is **True**, the link text and URLs will be escaped using Django's built-in **escape** filter. The default value for **autoescape** is **True**.

Note

If **urlize** is applied to text that already contains HTML markup, things won't work as expected. Apply this filter only to plain text.

---

urlizetrunc

Converts URLs and email addresses into clickable links just like urlize, but truncates URLs longer than the given character limit.

Argument: Number of characters that link text should be truncated to, including the ellipsis that's added if truncation is necessary.

For example:

```
{{ value|urlizetrunc:15 }}
```


If **value** is **"Check out www.djangoproject.com"**, the output would be **'Check out www.djangoproj…'**.

As with urlize, this filter should only be applied to plain text.

`wordcount`

Returns the number of words.

For example:

```
{{ value|wordcount }}
```

If **`value`** is **`"Joel is a slug"`**, the output will be **`4`**.

`wordwrap`

Wraps words at specified line length.

Argument: number of characters at which to wrap the text

For example:

```
{{ value|wordwrap:5 }}
```

If **`value`** is **`Joel is a slug`**, the output would be:

```
Joel
is a
slug
```

`yesno`

Maps values for **`True`**, **`False`**, and (optionally) **`None`**, to the strings “yes”, “no”, “maybe”, or a custom mapping passed as a comma-separated list, and returns one of those strings according to the value:

For example:

```
{{ value|yesno:"yeah,no,maybe" }}
```

Value	Argument	Outputs
`True`		**`yes`**
`True`	**`"yeah,no,maybe"`**	**`yeah`**
`False`	**`"yeah,no,maybe"`**	**`no`**
`None`	**`"yeah,no,maybe"`**	**`maybe`**
`None`	**`"yeah,no"`**	**`no`** (converts **`None`** to **`False`** if no mapping for **`None`** is given)

Internationalization tags and filters

Django provides template tags and filters to control each aspect of internationalization in templates. They allow for granular control of translations, formatting, and time zone conversions.

i18n

This library allows specifying translatable text in templates. To enable it, set **USE_I18N** to **True**, then load it with **{% load i18n %}**.

See Internationalization: in template code.

---

l10n

This library provides control over the localization of values in templates. You only need to load the library using **{% load l10n %}**, but you'll often set **USE_L10N** to **True** so that localization is active by default.

See Controlling localization in templates.

---

tz

This library provides control over time zone conversions in templates. Like **l10n**, you only need to load the library using **{% load tz %}**, but you'll usually also set **USE_TZ** to **True** so that conversion to local time happens by default.

See Time zone aware output in templates.

Other tags and filters libraries

Django comes with a couple of other template-tag libraries that you have to enable explicitly in your **INSTALLED_APPS** setting and enable in your template with the **{% load %}** tag.

django.contrib.humanize

A set of Django template filters useful for adding a "human touch" to data. See django.contrib.humanize.

---

static

static

To link to static files that are saved in **STATIC_ROOT** Django ships with a **static** template tag. If the **django.contrib.staticfiles** app is installed, the tag will serve files using **url()** method of the storage specified by **STATICFILES_STORAGE**. For example:

```
{% load static %}
<img src="{% static "images/hi.jpg" %}" alt="Hi!">
```

It is also able to consume standard context variables, e.g. assuming a **user_stylesheet** variable is passed to the template:

```
{% load static %}
<link rel="stylesheet" href="{% static user_stylesheet %}" type="text/css" media="screen">
```

If you'd like to retrieve a static URL without displaying it, you can use a slightly different call:

```
{% load static %}
{% static "images/hi.jpg" as myphoto %}
<img src="{{ myphoto }}">
```

Using Jinja2 templates?

See **Jinja2** for information on using the **static** tag with Jinja2.

---

get_static_prefix

You should prefer the **static** template tag, but if you need more control over exactly where and how **STATIC_URL** is injected into the template, you can use the **get_static_prefix** template tag:

```
{% load static %}
<img src="{% get_static_prefix %}images/hi.jpg" alt="Hi!">
```

여러 번 값을 필요로하는 경우 추가 처리를 피할 수있는 두 번째 형식이 있습니다.

```
{% load static %}
{% get_static_prefix as STATIC_PREFIX %}

<img src="{{ STATIC_PREFIX }}images/hi.jpg" alt="Hi!">
<img src="{{ STATIC_PREFIX }}images/hi2.jpg" alt="Hello!">
```

---

get_media_prefix

받는 유사 **get_static_prefix**, **get_media_prefix**미디어 접두사 템플릿 변수를 채 웁니다 **MEDIA_URL**, 예를 :

```
{% load static %}
<body data-media-url="{% get_media_prefix %}">
```

값을 데이터 속성에 저장하여 JavaScript 컨텍스트에서 값을 사용하려는 경우 해당 값을 적절하게 이스케이프 처리합니다.

Django 템플릿 언어

Django 템플릿 언어 : Python 프로그래머 용

더 알아보기

Django 정보

장고 시작하기

팀 구성

장고 소프트웨어 재단

윤리 강령

언어 : **ko**

문서 버전 : **2.2**

다양성 선언문

참여하다

그룹 가입

장고에 기부하기

버그 제출

보안 문제 신고

우리를 따르라.

GitHub

지저귀다

뉴스 RSS

장고 사용자 메일 링리스트