BUILT-IN TAGS

- autoescape
- block
- comment
- csrf_token
- cycle
- debug
- extends
- filter
- firstof
- for
- if
- ifchanged
- ifequal
- ifnotequal
- include
- load
- lorem
- now
- regroup
- resetcycle
- spaceless
- templatetag
- url
- verbatim
- widthratio
- with

ADMIN_LIST

- admin_actions
- admin_list_filter
- change_list_object_tools
- date_hierarchy
- pagination
- paginator_number
- result_list
- search_form

ADMIN_MODIFY

- change_form_object_tools
- prepopulated_fields_js
- submit_row

ADMIN_URLS

- add_preserved_filters

CACHE

- cache

HIGHLIGHTING

- highlight

HOSTS

- host_url

HOSTS_OVERRIDE

- url

I18N

- blocktrans
- get_available_languages
- get_current_language
- get_current_language_bidi
- get_language_info
- get_language_info_list
- language
- trans

INDENT_TEXT

- indentby

JET_DASHBOARD_TAGS

- get_dashboard

BUILT-IN TAGS

autoescape

Force autoescape behavior for this block.

block

Define a block that can be overridden by child templates.

comment

Ignore everything between {% comment %} and {% endcomment %}.

csrf_token

cycle

Cycle among the given strings each time this tag is encountered.

Within a loop, cycles among the given strings each time through the loop:

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

Outside of a loop, give the values a unique name the first time you call it, then use that name each successive time through:

```
<tr class="{% cycle 'row1' 'row2' 'row3' as rowcolors %}">...</tr>
<tr class="{% cycle rowcolors %}">...</tr>
<tr class="{% cycle rowcolors %}">...</tr>
```

You can use any number of values, separated by spaces. Commas can also be used to separate values; if a comma is used, the cycle values are interpreted as literal strings.

The optional flag "silent" can be used to prevent the cycle declaration from returning any value:

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

debug

Output a whole load of debugging information, including the current context and imported modules.

Sample usage:

```
<pre>
    {% debug %}
</pre>
```

extends

Signal that this template extends a parent template.

This tag may be used in two ways: `{% extends "base" %}` (with quotes) uses the literal value "base" as the name of the parent template to extend, or `{% extends variable %}` uses the value of `variable` as either the name of the parent template to extend (if it evaluates to a string) or as the parent template itself (if it evaluates to a Template object).

filter

Filter the contents of the block through variable filters.

Filters can also be piped through each other, and they can have arguments -- just like in variable syntax.

Sample usage:

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

JET_TAGS

- jet_change_form_sibling_links_enabled
- jet_delete_confirmation_context
- jet_get_bookmarks
- jet_get_current_theme
- jet_get_current_version
- jet_get_date_format
- jet_get_datetime_format
- jet_get_menu
- jet_get_side_menu_compact
- jet_get_themes
- jet_get_time_format
- jet_next_object
- jet_popup_response_data
- jet_previous_object
- jet_static_translation_urls

L10N

- localize

LOG

- get_admin_log

STATIC

- get_media_prefix
- get_static_prefix
- static

SYNTAX_COLOR

- pygments_css

TZ

- get_current_timezone
- localtime
- timezone

Note that the `escape` and `safe` filters are not acceptable arguments. Instead, use the `autoescape` tag to manage autoescaping for blocks of template code.

firstof

Output the first variable passed that is not False.

Output nothing if all the passed variables are False.

Sample usage:

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

This is equivalent to:

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

but obviously much cleaner!

You can also use a literal string as a fallback value in case all passed variables are False:

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

If you want to disable auto-escaping of variables you can use:

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

Or if only some variables should be escaped, you can use:

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

for

Loop over each item in an array.

For example, to display a list of athletes given `athlete_list`:

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

You can loop over a list in reverse by using `{% for obj in list reversed %}`.

You can also unpack multiple values from a two-dimensional array:

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

The `for` tag can take an optional `{% empty %}` clause that will be 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>
```

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

VARIABLE	DESCRIPTION

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 "above" the current one

if

Evaluate a variable, and if that variable is "true" (i.e., exists, is not empty, and is not a false boolean value), output the contents of the block:

```
{% if athlete_list %}
    Number of athletes: {{ athlete_list|count }}
{% 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|count }}` variable.

The `if` tag may take one or several `` {% elif %}`` clauses, as well as an `{% else %}` clause that will be displayed if all previous conditions fail. These clauses are optional.

`if` tags may use `or`, `and` or `not` to test a number of variables or to negate a given variable:

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

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

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

Comparison operators are also available, and the use of filters is also allowed, for example:

```
{% if articles|length >= 5 %}...{% endif %}
```

Arguments and operators _must_ have a space between them, so `{% if 1>2 %}` is not a valid if tag.

All supported operators are: `or`, `and`, `in`, `not in` `==`, `!=`, `>`, `>=`, `<` and `<=`.

Operator precedence follows Python.

ifchanged

Check if a value has changed from the last iteration of a loop.

The `{% ifchanged %}` block tag is used within a loop. It has two possible uses.

1. Check its own rendered contents against its previous state and only displays the content if it has changed. For example, this displays a list of days, only displaying the month if it changes:

```
<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. If given one or more variables, check whether any variable has changed. For example, the following shows the date every time it changes, while showing the hour if either the hour or the date has changed:

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

ifequal

Output the contents of the block if the two arguments equal each other.

Examples:

```
{% ifequal user.id comment.user_id %}
    ...
{% endifequal %}

{% ifnotequal user.id comment.user_id %}
    ...
{% else %}
    ...
{% endifnotequal %}
```

ifnotequal

Output the contents of the block if the two arguments are not equal. See ifequal.

include

Load a template and render it with the current context. You can pass additional context using keyword arguments.

Example:

```
{% include "foo/some_include" %}
{% include "foo/some_include" with bar="BAZZ!" baz="BING!" %}
```

Use the `only` argument to exclude the current context when rendering the included template:

```
{% include "foo/some_include" only %}
{% include "foo/some_include" with bar="1" only %}
```

load

Load a custom template tag library into the parser.

For example, to load the template tags in `django/templatetags/news/photos.py`:

```
{% load news.photos %}
```

Can also be used to load an individual tag/filter from a library:

```
{% load byline from news %}
```

lorem

Create random Latin text useful for providing test data in templates.

Usage format:

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

`count` is a number (or variable) containing the number of paragraphs or words to generate (default is 1).

`method` is either `w` for words, `p` for HTML paragraphs, `b` for plain-text paragraph blocks (default is `b`).

`random` is the word `random`, which if given, does not use the common paragraph (starting "Lorem ipsum dolor sit amet, consectetuer...").

Examples:

- `{% lorem %}` outputs the common "lorem ipsum" paragraph
- `{% lorem 3 p %}` outputs the common "lorem ipsum" paragraph and two random paragraphs each wrapped in HTML `<p>` tags
- `{% lorem 2 w random %}` outputs two random latin words

now

Display the date, formatted according to the given string.

Use the same format as PHP's `date()` function; see https://php.net/date for all the possible values.

Sample usage:

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

regroup

Regroup a list of alike objects by a common attribute.

This complex tag is best illustrated by use of an example: say that `musicians` is a list of `Musician` objects that have `name` and `instrument` attributes, and you'd like to display a list that looks like:

> - **Guitar:**
>   - Django Reinhardt
>   - Emily Remler
> - **Piano:**
>   - Lovie Austin
>   - Bud Powell
> - **Trumpet:**
>   - Duke Ellington

The following snippet of template code would accomplish this dubious task:

```
{% regroup musicians by instrument as grouped %}
<ul>
{% for group in grouped %}
    <li>{{ group.grouper }}
    <ul>
        {% for musician in group.list %}
        <li>{{ musician.name }}</li>
        {% endfor %}
    </ul>
{% endfor %}
</ul>
```

As you can see, `{% regroup %}` populates a variable with a list of objects with `grouper` and `list` attributes. `grouper` contains the item that was grouped by; `list` contains the list of objects that share that `grouper`. In this case, `grouper` would be `Guitar`, `Piano` and `Trumpet`, and `list` is the list of musicians who play this instrument.

Note that `{% regroup %}` does not work when the list to be grouped is not sorted by the key you are grouping by! This means that if your list of musicians was not sorted by instrument, you'd need to make sure it is sorted before using it, i.e.:

```
{% regroup musicians|dictsort:"instrument" by instrument as grouped %}
```

resetcycle

Reset a cycle tag.

If an argument is given, reset the last rendered cycle tag whose name matches the argument, else reset the last rendered cycle tag (named or unnamed).

spaceless

Remove whitespace between HTML tags, including tab and newline characters.

Example usage:

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

This example returns this HTML:

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

Only space between *tags* is normalized -- not space between tags and text. In this example, the space around `Hello` isn't stripped:

```
{% spaceless %}
    <strong>
```

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

templatetag

Output one of the bits 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	#}

url

Return an absolute URL matching the given view with its parameters.

This is a way to define links that aren't tied to a particular URL configuration:

```
{% url "url_name" arg1 arg2 %}

or

{% url "url_name" name1=value1 name2=value2 %}
```

The first argument is a URL pattern name. Other arguments are space-separated values that will be filled in place of positional and keyword arguments in the URL. Don't mix positional and keyword arguments. All arguments for the URL must be present.

For example, if you have a view `app_name.views.client_details` taking the client's id and the corresponding line in a URLconf looks like this:

```
path('client/<int:id>/', views.client_details, name='client-detail-view')
```

and this app's URLconf is included into the project's URLconf under some path:

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

then in a template you can create a link for a certain client like this:

```
{% url "client-detail-view" client.id %}
```

The URL will look like `/clients/client/123/`.

The first argument may also be the name of a template variable that will be evaluated to obtain the view name or the URL name, e.g.:

```
{% with url_name="client-detail-view" %}
{% url url_name client.id %}
{% endwith %}
```

verbatim

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

Usage:

```
{% verbatim %}
    {% don't process this %}
{% endverbatim %}
```

You can also designate a specific closing tag block (allowing the unrendered use of `{% endverbatim %}`):

```
{% verbatim myblock %}
    ...
{% endverbatim myblock %}
```

widthratio

For creating bar charts and such. Calculate the ratio of a given value to a maximum value, and then apply 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

Add one or more values to the context (inside of this block) for caching and easy access.

For example:

```
{% with total=person.some_sql_method %}
    {{ total }} object{{ total|pluralize }}
{% endwith %}
```

Multiple values can be added to the context:

```
{% with foo=1 bar=2 %}
    ...
{% endwith %}
```

The legacy format of `{% with person.some_sql_method as total %}` is still accepted.

ADMIN_LIST

To use these tags, put `{% load admin_list %}` in your template before using the tag.

admin_actions

admin_list_filter

change_list_object_tools

Display the row of change list object tools.

date_hierarchy

pagination

paginator_number

Generate an individual page index link in a paginated list.

result_list

search_form

ADMIN_MODIFY

To use these tags, put `{% load admin_modify %}` in your template before using the tag.

change_form_object_tools

Display the row of change form object tools.

prepopulated_fields_js

submit_row

ADMIN_URLS

To use these tags, put `{% load admin_urls %}` in your template before using the tag.

add_preserved_filters

CACHE

To use these tags, put `{% load cache %}` in your template before using the tag.

cache

This will cache the contents of a template fragment for a given amount of time.

Usage:

```
{% load cache %}
{% cache [expire_time] [fragment_name] %}
    .. some expensive processing ..
{% endcache %}
```

This tag also supports varying by a list of arguments:

```
{% load cache %}
{% cache [expire_time] [fragment_name] [var1] [var2] .. %}
    .. some expensive processing ..
{% endcache %}
```

Optionally the cache to use may be specified thus:

```
{% cache ....  using="cachename" %}
```

Each unique set of arguments will result in a unique cache entry.

HIGHLIGHTING

To use these tags, put `{% load highlighting %}` in your template before using the tag.

highlight

Tag to put a highlighted source code <pre> block in your code. This takes two arguments, the language and a little explaination message that will be generated before the code. The second argument is optional.

Your code will be fed through pygments so you can use any language it supports.

Usage:

```
{% load highlighting %}
{% highlight 'python' 'Excerpt: blah.py' %}
def need_food(self):
    print("Love is colder than death")
{% endhighlight %}
```

HOSTS

To use these tags, put `{% load hosts %}` in your template before using the tag.

host_url

Simple tag to reverse the URL inclusing a host.

{% host_url 'view-name' host 'host-name' %} {% host_url 'view-name' host 'host-name' 'spam' %} {% host_url 'view-name' host 'host-name' scheme 'https' %} {% host_url 'view-name' host 'host-name' as url_on_host_variable %} {% host_url 'view-name' varg1=vvalue1 host 'host-name' 'spam' 'hvalue1' %} {% host_url 'view-name' vvalue2 host 'host-name' 'spam' harg2=hvalue2 %}

HOSTS_OVERRIDE

To use these tags, put `{% load hosts_override %}` in your template before using the tag.

url

A tag to override the built-in url template tag. Accepts host parameters optionally.

{% url 'view-name' host 'host-name' %} {% url 'view-name' host 'host-name' 'spam' %} {% url 'view-name' host 'host-name' scheme 'https' %} {% url 'view-name' host 'host-name' as url_on_host_variable %} {% url 'view-name' varg1=vvalue1 host 'host-name' 'spam' 'hvalue1' %} {% url 'view-name' vvalue2 host 'host-name' 'spam' harg2=hvalue2 %}

I18N

To use these tags, put `{% load i18n %}` in your template before using the tag.

blocktrans

Translate a block of text with parameters.

Usage:

```
{% blocktrans with bar=foo|filter boo=baz|filter %}
This is {{ bar }} and {{ boo }}.
{% endblocktrans %}
```

Additionally, this supports pluralization:

```
{% blocktrans count count=var|length %}
There is {{ count }} object.
{% plural %}
There are {{ count }} objects.
{% endblocktrans %}
```

This is much like ngettext, only in template syntax.

The "var as value" legacy format is still supported:

```
{% blocktrans with foo|filter as bar and baz|filter as boo %}
{% blocktrans count var|length as count %}
```

The translated string can be stored in a variable using asvar:

```
{% blocktrans with bar=foo|filter boo=baz|filter asvar var %}
This is {{ bar }} and {{ boo }}.
{% endblocktrans %}
{{ var }}
```

Contextual translations are supported:

```
{% blocktrans with bar=foo|filter context "greeting" %}
    This is {{ bar }}.
{% endblocktrans %}
```

This is equivalent to calling pgettext/npgettext instead of (u)gettext/(u)ngettext.

get_available_languages

Store a list of available languages in the context.

Usage:

```
{% get_available_languages as languages %}
{% for language in languages %}
...
{% endfor %}
```

This puts settings.LANGUAGES into the named variable.

get_current_language

Store the current language in the context.

Usage:

```
{% get_current_language as language %}
```

This fetches the currently active language and puts its value into the `language` context variable.

get_current_language_bidi

Store the current language layout in the context.

Usage:

```
{% get_current_language_bidi as bidi %}
```

This fetches the currently active language's layout and puts its value into the `bidi` context variable. True indicates right-to-left layout, otherwise left-to-right.

get_language_info

Store the language information dictionary for the given language code in a context variable.

Usage:

```
{% get_language_info for LANGUAGE_CODE as l %}
{{ l.code }}
{{ l.name }}
{{ l.name_translated }}
{{ l.name_local }}
{{ l.bidi|yesno:"bi-directional,uni-directional" }}
```

get_language_info_list

Store a list of language information dictionaries for the given language codes in a context variable. The language codes can be specified either as a list of strings or a settings.LANGUAGES style list (or any sequence of sequences whose first items are language codes).

Usage:

```
{% get_language_info_list for LANGUAGES as langs %}
{% for l in langs %}
  {{ l.code }}
  {{ l.name }}
  {{ l.name_translated }}
  {{ l.name_local }}
  {{ l.bidi|yesno:"bi-directional,uni-directional" }}
{% endfor %}
```

language

Enable the given language just for this block.

Usage:

```
{% language "de" %}
    This is {{ bar }} and {{ boo }}.
{% endlanguage %}
```

trans

Mark a string for translation and translate the string for the current language.

Usage:

```
{% trans "this is a test" %}
```

This marks the string for translation so it will be pulled out by makemessages into the .po files and runs the string through the translation engine.

There is a second form:

```
{% trans "this is a test" noop %}
```

This marks the string for translation, but returns the string unchanged. Use it when you need to store values into forms that should be translated later on.

You can use variables instead of constant strings to translate stuff you marked somewhere else:

```
{% trans variable %}
```

This tries to translate the contents of the variable `variable`. Make sure that the string in there is something that is in the .po file.

It is possible to store the translated string into a variable:

```
{% trans "this is a test" as var %}
{{ var }}
```

Contextual translations are also supported:

```
{% trans "this is a test" context "greeting" %}
```

This is equivalent to calling pgettext instead of (u)gettext.

INDENT_TEXT

To use these tags, put `{% load indent_text %}` in your template before using the tag.

indentby

Add indentation to text between the tags by the given indentation level.

{% indentby <indent_level> [if <statement>] %} ... {% endindentby %}

JET_DASHBOARD_TAGS

To use these tags, put `{% load jet_dashboard_tags %}` in your template before using the tag.

get_dashboard

JET_TAGS

To use these tags, put `{% load jet_tags %}` in your template before using the tag.

jet_change_form_sibling_links_enabled

jet_delete_confirmation_context

jet_get_bookmarks

jet_get_current_theme

jet_get_current_version

jet_get_date_format

jet_get_datetime_format

jet_get_menu

jet_get_side_menu_compact

jet_get_themes

jet_get_time_format

jet_next_object

jet_popup_response_data

jet_previous_object

jet_static_translation_urls

L10N

To use these tags, put `{% load l10n %}` in your template before using the tag.

localize

Force or prevents localization of values, regardless of the value of settings.USE_L10N.

Sample usage:

```
{% localize off %}
    var pi = {{ 3.1415 }};
{% endlocalize %}
```

LOG

To use these tags, put `{% load log %}` in your template before using the tag.

get_admin_log

Populate a template variable with the admin log for the given criteria.

Usage:

```
{% get_admin_log [limit] as [varname] for_user [context_var_containing_user_   ] %}
```

Examples:

```
{% get_admin_log 10 as admin_log for_user 23 %}
{% get_admin_log 10 as admin_log for_user user %}
{% get_admin_log 10 as admin_log %}
```

Note that `context_var_containing_user_obj` can be a hard-coded integer (user ID) or the name of a template context variable containing the user object whose ID you want.

STATIC

To use these tags, put `{% load static %}` in your template before using the tag.

get_media_prefix

Populate a template variable with the media prefix, settings.MEDIA_URL.

Usage:

```
{% get_media_prefix [as varname] %}
```

Examples:

```
{% get_media_prefix %}
{% get_media_prefix as media_prefix %}
```

get_static_prefix

Populate a template variable with the static prefix, settings.STATIC_URL.

Usage:

```
{% get_static_prefix [as varname] %}
```

Examples:

```
{% get_static_prefix %}
{% get_static_prefix as static_prefix %}
```

static

Join the given path with the STATIC_URL setting.

Usage:

```
{% static path [as varname] %}
```

Examples:

```
{% static "myapp/css/base.css" %}
{% static variable_with_path %}
{% static "myapp/css/base.css" as admin_base_css %}
{% static variable_with_path as varname %}
```

SYNTAX_COLOR

To use these tags, put `{% load syntax_color %}` in your template before using the tag.

pygments_css

TZ

To use these tags, put `{% load tz %}` in your template before using the tag.

get_current_timezone

Store the name of the current time zone in the context.

Usage:

```
{% get_current_timezone as TIME_ZONE %}
```

This will fetch the currently active time zone and put its name into the `TIME_ZONE` context variable.

localtime

Force or prevent conversion of datetime objects to local time, regardless of the value of settings.USE_TZ.

Sample usage:

```
{% localtime off %}{{ value_in_utc }}{% endlocaltime %}
```

timezone

Enable a given time zone just for this block.

The `timezone` argument must be an instance of a `tzinfo` subclass, a time zone name, or `None`. If it is `None`, the default time zone is used within the block.

Sample usage:

```
{% timezone "Europe/Paris" %}
    It is {{ now }} in Paris.
{% endtimezone %}
```