Django Native Tags Documentation

Release 0.5.3

Justin Quick

November 04, 2013

Contents

Authors Justin Quick <justquick@gmail.com>

Version 0.5

```
pip install django-native-tags==0.5.3
```

Django Native Tags is a way of making the creation of template tags stupidly simple. Tags are "native" because there is a much closer relationship between the tag in the template and a Python function behind the scenes. The app abstracts the work needed to parse out the templatetag syntax into a useable form for a Python function. For example:

Define an arbitrary function in your templatetags:

```
def add(x, y):
   return x + y
add.function = True
```

Use the function in your template:

```
{% add 1000 100 as num %}
{{ num|intcomma }}
```

Which outputs:

```
1,100
```

Other *Features* of Native Tags:

- Keyword argument parsing
- Quoted strings parsed correctly
- Add templatetags to Django's builtins (no `{% load %}` required)
- Auto resolve of template variables
- Universal and per-tag caching
- Straightforward template tag unittesting
- Error tolerant by letting you specify a fallback return value

The real power of the module comes in the *Contrib Add Ons* which has tons of tags for various uses including comparisons, regex operations, math operaions, and much more. By default it is a functional replacement to James Bennett's django-template-utils right out of the box

CHAPTER 1

Contents

1.1 Change Log

1.1.1 v0.4

- Reorganized loader to index tags at the beginning and only load tags at runtime, good call legutierr!
- Per tag unittesting with the `test` funciton attribute and added lots more tests to use the new feature
- Added operator tags to `native_tags.contrib.op`

1.1.2 v0.3

- `smart_if` tag removed in favor of Django v1.2's updated if tag.
- Universal and per-tag caching
- Bugfixes to numerous contrib packages

1.1.3 v0.2

- Filter expressions render correctly
- `math_` and `smart_if` contrib packages added
- `NATIVE_CONTRIB` setting changed to `NATIVE_TAGS`
- No more `NATIVE_LOAD` setting
- Ability to have custom names for comparison tags (no prepended `if_`)

1.1.4 v0.1

- `django-template-tags` functionality
- Auto load tags
- Parsing with `shlex`

- Auto resolve template arguments and keyword arguments
- `function`, `comparison`, `block`, and `filter` native tag types

1.2 Types of Tags

1.2.1 Function tags

native_tags.nodes.**do_function**(*parser*, *token*)

Performs a defined function on the passed arguments. Normally this returns the output of the function into the template. If the second to last argument is `as`, the result of the function is stored in the context and is named whatever the last argument is.

Syntax:

```
{% [function] [var args...] [name=value kwargs...] [as varname] %}
```

Examples:

```
{% search '^(\d{3})$' 800 as match %}

{% map sha1 hello world %}
```

1.2.2 Comparison tags

native_tags.nodes.**do_comparison**(*parser*, *token*)

Compares passed arguments. Attached functions should return boolean `True` or `False`. If the attached function returns `True`, the first node list is rendered. If the attached function returns `False`, the second optional node list is rendered (part after the `{% else %}` statement). If the last argument in the tag is `negate`, then the opposite node list is rendered (like an `ifnot` tag).

Syntax:

```
{% if_[comparison] [var args...] [name=value kwargs...] [negate] %}
    {# first node list in here #}
{% else %}
    {# second optional node list in here #}
{% endif_[comparison] %}
```

Supported comparisons are `match`, `find`, `startswith`, `endswith`, `less`, `less_or_equal`, `greater` and `greater_or_equal` and many more. Checkout the *Contrib Add Ons* for more examples

Examples:

```
{% if_less some_object.id 3 %}
{{ some_object }} has an id less than 3.
{% endif_less %}

{% if_match request.path '^/$' %}
Welcome home
{% endif_match %}
```

1.2.3 Block tags

native_tags.nodes.**do_block**(*parser*, *token*)

Process several nodes inside a single block Block functions take `context`, `nodelist` as first arguments If the second to last argument is `as`, the rendered result is stored in the context and is named whatever the last argument is.

Syntax:

```
{% [block] [var args...] [name=value kwargs...] [as varname] %}
    ... nodelist ...
{% end[block] %}
```

Examples:

```
{% render_block as rendered_output %}
    {{ request.path }}/blog/{{ blog.slug }}
{% endrender_block %}

{% highlight_block python %}
    import this
{% endhighlight_block %}
```

1.2.4 Filter tags

Native Filter tags are pretty much the same as regular Django filter tags. There is no special sauce here because the arguments that you can pass filter tags is very limited and is currently being expanded in this issue. Filters take the piped value as the first argument and as of now *only one* extra argument like so:

```
{{ value|filter:arg }}
```

1.3 Features of Native Tags

1.3.1 Quoted Strings

Using the `shlex` module, double/single quoted strings are interpreted as a single string element now. Take this example templatetag: `{% myfunc 'hello world' %}`. If you were to write your own templatetag from scratch to handle this when calling `token.contents.split()` it would return `["'hello", "world'"]` and ruin the continuity of the string. With Native Tags quoted strings act just like they should and the arguments parsed from the above example would be `['hello world']` as expected.

1.3.2 Keyword arguments

Arguments and keyword arguments are parsed out of the template to more closely match the Python API. That way `{% myfunc arg1 arg2 kwarg1=value1 kwarg2=value2 %}` represents the function with signature `def myfunc(arg1, arg2, kwarg1=value1, kwarg2=value2)`. This also means that `*args` and `**kwargs` can also be used in native tag functions.

1.3.3 Auto resolve

By default variables passed into Native Tags are by default resolved in the tag's context. If a variable cannot be found, then the string is just returned. This functionality can be turned off by setting the tag function's `resolve` attribute to `False`.

1.3.4 Auto load

Tired of calling `{% load %}` on every single template you want to use your favorite tags on? Native Tags lets you define templatetags as part of the builtins which are loaded automagically. Just place your favorite templatetags into the `DJANGO_BUILTIN_TAGS` setting and kiss `{% load %}` tags good bye

1.4 Settings

1.4.1 DJANGO_BUILTIN_TAGS

Tuple of Django templatetags modules to load by default. This places them in the Django template Library builtins. Place your most commonly used templatetags here and you will never have to `{% load %}` them.

Example:

```
DJANGO_BUILTIN_TAGS = (
    'native_tags.templatetags.native',
    'django.contrib.markup.templatetags.markup',
    #...
)
```

1.4.2 NATIVE_LIBRARY

Updates the library of tags to load by default. Define any valid types of functions here. Lets you define them right in `settings.py`

Example:

```
NATIVE_LIBRARY = {
     'function':{ 'add': lambda x, y: x + y },
     'comparison':{ 'is_in': lambda x, y: x in y },
     'filter':{ 'rstrip': lambda s: s.rstrip() },
     'block':{ 'comment': lambda: '' }
 }
```

1.4.3 NATIVE_TAGS

Tuple of contrib tag modules to load. Use the ones from *Contrib Add Ons* or add your own

Out of the box, this is just the set of tags needed to be a functional replacement to django-template-utils

All available options included in this app:

```
    'native_tags.contrib.comparison',
    'native_tags.contrib.context',
    'native_tags.contrib.generic_content',
    'native_tags.contrib.generic_markup',
    'native_tags.contrib.hash',
    'native_tags.contrib.serializers',
    'native_tags.contrib.baseencode',
    'native_tags.contrib.regex',
    'native_tags.contrib.math',
    'native_tags.contrib.mapreduce',
    'native_tags.contrib.cal',
    'native_tags.contrib.rand',

    # Native tags with dependencies
    'native_tags.contrib.gchart', # GChartWrapper
    'native_tags.contrib.pygmentize', # Pygments
    'native_tags.contrib.feeds', # Feedparser
```

1.4.4 NATIVE_DEFAULT_CACHE_TIMEOUT

Cache timeout (if any) in seconds. If this is set, all tags will cache for this amount of time, otherwise you may define caching at the per-tag level. Defaults to *None*

Example 1hr cache:

```
NATIVE_DEFAULT_CACHE_TIMEOUT = 60 * 60
```

1.5 Decorators

The decorators that come with this app are there for convienence but are not necessary to use. Their purpose is to just set some important attributes on the wrapped function and maintain docstrings.

The normal syntax (without decorators):

```
def myfunc(context):
    return context['myvar'].split()
myfunc.function = 1
myfunc.takes_context = 1
myfunc.name = 'my_function_tag'
myfunc.cache = 3600
```

Is equivalent to:

```
from native_tags.decorators import function

@function(takes_context=1, cache=3600, name='my_function_tag')
def myfunc(context):
    return context['myvar'].split()
```

And can be used in the template using the *name*:

```
{% my_function_tag as varsplit %}
```

1.5.1 Important Attributes

`function`, `filter`, `comparison`, `block` - boolean. Determines which kind of tag the function represents. A function can have multiple uses (eg. function and filter)

`name` - string. The actual name of the tag to use in the template. Defaults to the function name.

`resolve` - boolean. If `True`, all argments are resolved in the context of the tag. Set to `False` to parse your own raw text arguments. Default is `True`

`takes_context` - boolean. If `True` the context itself is prepended to the function arguments. Default is `False`

`takes_request` - boolean. If `True` the request object is prepended to the function arguments. Make sure `django.core.context_processors.request` is in your `TEMPLATE_CONTEXT_PROCESSORS` setting. Default is `False`

`inclusion` - boolean. If `True` then the function is treated as an inclusion tag. Inclusion tags work a bit differently in native tags, the function must return a tuple of (`template_name`, `context`). This lets you dynamically define the name of the template to use in rendering.

`apply_filters` - boolean. If `True`, the filter expressions are resolved for each argument and keyword argument if present. Default is `True`

`cache` - integer. The cache timeout time in seconds to use if any. Default is no caching.

`test` - dictionary. Configuration data for a unittest. Keys are: `args`,‘‘kwargs‘‘, and `result`. When testing native_tags it will assert that the tag called with args/kwargs will return the expected result. Default expected result is `True`.

`fallback` - arbitrary object. If for any reason the native tag you are running encounters an error, return this fallback value instead. Providing a default (like an empty list) is a good way to fail elegantly and not have your site crash.

1.5.2 Decorator Types

native_tags.decorators.**function**(*inner*, ***options*)
Function tag function decorator

Syntax:

```
@function([**options]):
def my_function([*args], [**kwargs]):
    return args, kwargs
```

native_tags.decorators.**comparison**(*inner*, ***options*)
Comparison tag function decorator

Syntax:

```
@comparison([**options]):
def my_comparison([*vars], [**tag_options]):
    return True
```

native_tags.decorators.**block**(*inner*, ***options*)
Block tag function decorator

Syntax:

```
@block([**options])
def my_tag_function(context, nodelist, [*vars], [**tag_options]):
    return nodelist.render(context)
```

native_tags.decorators.**filter**(*inner*, ***options*)

Filter tag function decorator

Syntax:

```
@filter([**options]):
def my_filter(value, arg):
    return value
```

1.6 Testing Your Tags

With native tags, the templatetag is simply just a function you define which means that testing the underlying functionality of your templatetag does not require going to your browser. Just pass in the appropriate arguments and test the output. Here is an example of unittesting a native tag which demonstrates how trivial it can be:

```
# tests.py
from django.test import TestCase
from native_tags import function

# Define or import your tags here

@function
def adder(x, y):
    return x + y

class AdderTest(TestCase):
    def test_adder(self):
        self.assertEqual(adder(1, 1), 2)
```

That is the full example above, but to add simple tests to a native function, you can use the `test` attribute passing in the arguments and expected result of the function call. Here is a shorter example that does the same thing as above:

```
def adder(x, y):
    return x + y
adder = function(adder, test={'args': (1, 1), 'result': 2})
```

Running `./manage.py test native_tags` will test *only* your own tags in your project. This is useful for testing the sanity of your project and its applications and great to put in a CI job.

1.7 Test Application

The `example_project` folder in the source tree root is also useful for testing native tags itself. Run `./manage.py test app` in that directory to run a test application on native tags.

1.8 Contrib Add Ons

Tags ported from django-template-utils and other sources

1.8.1 baseencode - base64 encoding/decoding

native_tags.contrib.baseencode.**b16decode**(*s*, **args*, ***kwargs*)
Decode a Base16 encoded string.

s is the string to decode. Optional casefold is a flag specifying whether a lowercase alphabet is acceptable as input. For security purposes, the default is False.

The decoded string is returned. A TypeError is raised if s were incorrectly padded or if there are non-alphabet characters present in the string.

Syntax:

```
{% b16decode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

native_tags.contrib.baseencode.**b16encode**(*s*, **args*, ***kwargs*)
Encode a string using Base16.

s is the string to encode. The encoded string is returned.

Syntax:

```
{% b16encode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

native_tags.contrib.baseencode.**b32decode**(*s*, **args*, ***kwargs*)
Decode a Base32 encoded string.

s is the string to decode. Optional casefold is a flag specifying whether a lowercase alphabet is acceptable as input. For security purposes, the default is False.

RFC 3548 allows for optional mapping of the digit 0 (zero) to the letter O (oh), and for optional mapping of the digit 1 (one) to either the letter I (eye) or letter L (el). The optional argument map01 when not None, specifies which letter the digit 1 should be mapped to (when map01 is not None, the digit 0 is always mapped to the letter O). For security purposes the default is None, so that 0 and 1 are not allowed in the input.

The decoded string is returned. A TypeError is raised if s were incorrectly padded or if there are non-alphabet characters present in the string.

Syntax:

```
{% b32decode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

native_tags.contrib.baseencode.**b32encode**(*s*, **args*, ***kwargs*)
Encode a string using Base32.

s is the string to encode. The encoded string is returned.

Syntax:

```
{% b32encode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

native_tags.contrib.baseencode.**b64decode**(*s*, **args*, ***kwargs*)
Decode a Base64 encoded string.

s is the string to decode. Optional altchars must be a string of at least length 2 (additional characters are ignored) which specifies the alternative alphabet used instead of the '+' and '/' characters.

The decoded string is returned. A TypeError is raised if s were incorrectly padded or if there are non-alphabet characters present in the string.

Syntax:

```
{% b64decode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

native_tags.contrib.baseencode.**b64encode**(*s*, **args*, ***kwargs*)
Encode a string using Base64.

s is the string to encode. Optional altchars must be a string of at least length 2 (additional characters are ignored) which specifies an alternative alphabet for the '+' and '/' characters. This allows an application to e.g. generate url or filesystem safe Base64 strings.

The encoded string is returned.

Syntax:

```
{% b64encode [string] [options] %}
```

This is a *function tag*. This is a *filter tag*.

1.8.2 `cal` - create formatted `HTMLCalendar`

native_tags.contrib.cal.**calendar**(*format*, **args*, ***kwargs*)

Creates a formatted `HTMLCalendar`. Argument `format` can be one of `month`, `year`, or `yearpage` Keyword arguments are collected and passed into `HTMLCalendar.formatmonth`, `HTMLCalendar.formatyear`, and `HTMLCalendar.formatyearpage`

Syntax:

```
{% calendar month [year] [month] %}
{% calendar year [year] %}
{% calendar yearpage [year] %}
```

Example:

```
{% calendr month 2009 10 %}
```

This is a *function tag*.

1.8.3 `context` - context manipulation tags

native_tags.contrib.context.**do_del**(*context*, **args*)
Deletes template variables from the context This is a *function tag*.

native_tags.contrib.context.**do_set**(*context*, ***kwargs*)
Updates the context with the keyword arguments This is a *function tag*.

native_tags.contrib.context.**document**(*o*)
Returns the docstring for a given object This is a *function tag*. This is a *filter tag*.

native_tags.contrib.context.**native_debug**()
None This is a *function tag*.

native_tags.contrib.context.**render_block**(*context*, *nodelist*)
Simply renders the nodelist with the current context This is a *block tag*.

native_tags.contrib.context.**template_block**(*context*, *nodelist*)
Return the rendered block's content with the current context This is a *block tag*.

native_tags.contrib.context.**template_string**(*context*, *template*)
Return the rendered template content with the current context This is a *function tag*.

1.8.4 comparison - compare template variables

Comparison tags

Tags supported by django-template-utils

native_tags.contrib.comparison.**greater**(*x*, *y*)
True if x is greater than y This is a *comparison tag*.

native_tags.contrib.comparison.**greater_or_equal**(*x*, *y*)
True if x is greater than or equal to y This is a *comparison tag*.

native_tags.contrib.comparison.**less**(*x*, *y*)
True if x is less than y This is a *comparison tag*.

native_tags.contrib.comparison.**less_or_equal**(*x*, *y*)
True if x is less than or equal to y This is a *comparison tag*.

New comparison tags

native_tags.contrib.comparison.**startswith**(*x*, *y*)
String comparison. True if x startswith y This is a *comparison tag*.

native_tags.contrib.comparison.**endswith**(*x*, *y*)
String comparison. True if x endswith y This is a *comparison tag*.

native_tags.contrib.comparison.**contains**(*x*, *y*)
String comparison. True if x contains y anywhere This is a *comparison tag*.

native_tags.contrib.comparison.**setting**(*x*)
True if setting x is defined in your settings This is a *comparison tag*.

native_tags.contrib.comparison.**divisible_by**(*x*, *y*)
Numeric comparison. True if x is divisible by y This is a *comparison tag*.

native_tags.contrib.comparison.**superset**(*x*, *y*)
Set comparison. True if x is a superset of y This is a *comparison tag*.

native_tags.contrib.comparison.**subset**(*x*, *y*)
Set comparison. True if x is a subset of y This is a *comparison tag*.

1.8.5 hash - MD5 and SHA tags

native_tags.contrib.hash.**md5**(*value*)
Returns MD5 hexadecimal hash of the value. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.hash.**sha1**(*value*)
Returns SHA1 hexadecimal hash of the value. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.hash.**sha224**(*value*)
Returns SHA224 hexadecimal hash of the value. Requires the `hashlib` module This is a *function tag*. This is a *filter tag*.

native_tags.contrib.hash.**sha256**(*value*)
Returns SHA256 hexadecimal hash of the value. Requires the `hashlib` module This is a *function tag*. This is a *filter tag*.

native_tags.contrib.hash.**sha384**(*value*)
Returns SHA384 hexadecimal hash of the value. Requires the `hashlib` module This is a *function tag*. This is a *filter tag*.

native_tags.contrib.hash.**sha512**(*value*)
Returns SHA512 hexadecimal hash of the value. Requires the `hashlib` module This is a *function tag*. This is a *filter tag*.

1.8.6 `feeds` - RSS feed parsing using `feedparser`

1.8.7 `gchart` - Google Charts via `GChartWrapper`

1.8.8 `generic_content` - access model content

Template tags which can do retrieval of content from any model.

native_tags.contrib.generic_content.**get_latest_object**(*model*, *field=None*)

Retrieves the latest object from a given model, in that model's default ordering, and stores it in a context variable. The optional field argument specifies which field to get_latest_by, otherwise the model's default is used

Syntax:

```
{% get_latest_object [app_name].[model_name] [field] as [varname] %}
```

Example:

```
{% get_latest_object comments.freecomment submitted_date as latest_comment %}
```

This is a *function tag*.

native_tags.contrib.generic_content.**get_latest_objects**(*model*, *num*, *field='?'*)

Retrieves the latest `num` objects from a given model, in that model's default ordering, and stores them in a context variable. The optional field argument specifies which field to get_latest_by, otherwise the model's default is used

Syntax:

```
{% get_latest_objects [app_name].[model_name] [num] [field] as [varname] %}
```

Example:

```
{% get_latest_objects comments.freecomment 5 submitted_date as latest_comments %}
```

This is a *function tag*.

native_tags.contrib.generic_content.**get_random_object**(*model*)

Retrieves a random object from a given model, and stores it in a context variable.

Syntax:

```
{% get_random_object [app_name].[model_name] as [varname] %}
```

Example:

```
{% get_random_object comments.freecomment as random_comment %}
```

This is a *function tag*.

native_tags.contrib.generic_content.**get_random_objects**(*model*, *num*)

Retrieves num random objects from a given model, and stores them in a context variable.

Syntax:

```
{% get_random_objects [app_name].[model_name] [num] as [varname] %}
```

Example:

```
{% get_random_objects comments.freecomment 5 as random_comments %}
```

This is a *function tag*.

native_tags.contrib.generic_content.**retrieve_object**(*model*, **args*, ***kwargs*)

Retrieves a specific object from a given model by primary-key lookup, and stores it in a context variable.

Syntax:

```
{% retrieve_object [app_name].[model_name] [lookup kwargs] as [varname] %}
```

Example:

```
{% retrieve_object flatpages.flatpage pk=12 as my_flat_page %}
```

This is a *function tag*.

1.8.9 generic_markup - filters for converting plain text to HTML

Filters for converting plain text to HTML and enhancing the typographic appeal of text on the Web.

native_tags.contrib.generic_markup.**apply_markup**(*value*, *arg=None*)

Applies text-to-HTML conversion.

Takes an optional argument to specify the name of a filter to use.

This is a *filter tag*.

native_tags.contrib.generic_markup.**smartypants**(*value*)

Applies SmartyPants to a piece of text, applying typographic niceties.

Requires the Python SmartyPants library to be installed; see http://web.chad.org/projects/smartypants.py/

This is a *filter tag*.

1.8.10 `mapreduce` - native `map` and `reduce` tags

native_tags.contrib.mapreduce.**do_map**(*func_name*, **sequence*)

Return a list of the results of applying the function to the items of the argument sequence(s).

Functions may be registered with `native_tags` or can be `builtins` or from the `operator` module

If more than one sequence is given, the function is called with an argument list consisting of the corresponding item of each sequence, substituting None for missing values when not all sequences have the same length. If the function is None, return a list of the items of the sequence (or a list of tuples if more than one sequence).

Syntax:

```
{% map [function] [sequence] %}
{% map [function] [item1 item2 ...] %}
```

For example:

```
{% map sha1 hello world %}
```

calculates:

```
[sha1(hello), sha1(world)]
```

This is a *function tag*.

native_tags.contrib.mapreduce.**do_reduce**(*func_name*, **sequence*)

Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value.

Functions may be registered with `native_tags` or can be `builtins` or from the `operator` module

Syntax:

```
{% reduce [function] [sequence] %}
{% reduce [function] [item1 item2 ...] %}
```

For example:

```
{% reduce add 1 2 3 4 5 %}
```

calculates:

```
((((1+2)+3)+4)+5) = 15
```

This is a *function tag*.

1.8.11 `math` - mathematical operations

native_tags.contrib.math_.**acos**(*arg1*, *arg2=None*)
Return the arc cosine (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**acosh**(*arg1*, *arg2=None*)
Return the hyperbolic arc cosine (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**asin**(*arg1*, *arg2=None*)
Return the arc sine (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**asinh**(*arg1*, *arg2=None*)
Return the hyperbolic arc sine (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**atan**(*arg1*, *arg2=None*)
Return the arc tangent (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**atan2**(*arg1*, *arg2=None*)
Return the arc tangent (measured in radians) of y/x. Unlike atan(y/x), the signs of both x and y are considered. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**atanh**(*arg1*, *arg2=None*)
Return the hyperbolic arc tangent (measured in radians) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**ceil**(*arg1*, *arg2=None*)
Return the ceiling of x as a float. This is the smallest integral value >= x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**copysign**(*arg1*, *arg2=None*)
Return x with the sign of y. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**cos**(*arg1*, *arg2=None*)
Return the cosine of x (measured in radians). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**cosh**(*arg1*, *arg2=None*)
Return the hyperbolic cosine of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**degrees**(*arg1*, *arg2=None*)
Convert angle x from radians to degrees. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**exp**(*arg1*, *arg2=None*)
Return e raised to the power of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**fabs**(*arg1*, *arg2=None*)
Return the absolute value of the float x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**factorial**(*arg1*, *arg2=None*)
Find x!. Raise a ValueError if x is negative or non-integral. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**floor**(*arg1*, *arg2=None*)
Return the floor of x as a float. This is the largest integral value <= x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**fmod**(*arg1*, *arg2=None*)
Return fmod(x, y), according to platform C. x % y may differ. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**frexp**(*arg1*, *arg2=None*)
Return the mantissa and exponent of x, as pair (m, e). m is a float and e is an int, such that x = m * 2.**e. If x is 0, m and e are both 0. Else 0.5 <= abs(m) < 1.0. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**fsum**(*arg1*, *arg2=None*)
Return an accurate floating point sum of values in the iterable. Assumes IEEE-754 floating point arithmetic. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**hypot**(*arg1*, *arg2=None*)
Return the Euclidean distance, sqrt(x*x + y*y). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**isinf**(*arg1*, *arg2=None*)
Check if float x is infinite (positive or negative). This is a *comparison tag*.

native_tags.contrib.math_.**isnan**(*arg1*, *arg2=None*)
Check if float x is not a number (NaN). This is a *comparison tag*.

native_tags.contrib.math_.**ldexp**(*arg1*, *arg2=None*)
Return x * (2**i). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**log**(*arg1*, *arg2=None*)
Return the logarithm of x to the given base. If the base not specified, returns the natural logarithm (base e) of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**log10**(*arg1*, *arg2=None*)
Return the base 10 logarithm of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**log1p**(*arg1*, *arg2=None*)
Return the natural logarithm of 1+x (base e). The result is computed in a way which is accurate for x near zero. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**modf**(*arg1*, *arg2=None*)
Return the fractional and integer parts of x. Both results carry the sign of x and are floats. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**pow**(*arg1*, *arg2=None*)
Return x**y (x to the power of y). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**radians**(*arg1*, *arg2=None*)
Convert angle x from degrees to radians. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**sin**(*arg1*, *arg2=None*)
Return the sine of x (measured in radians). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**sinh**(*arg1*, *arg2=None*)
Return the hyperbolic sine of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**sqrt**(*arg1*, *arg2=None*)
Return the square root of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**tan**(*arg1*, *arg2=None*)
Return the tangent of x (measured in radians). This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**tanh**(*arg1*, *arg2=None*)
Return the hyperbolic tangent of x. This is a *function tag*. This is a *filter tag*.

native_tags.contrib.math_.**trunc**(*arg1*, *arg2=None*)
Truncates x to the nearest Integral toward 0. Uses the __trunc__ magic method. This is a *function tag*. This is a *filter tag*.

1.8.12 op - Operators

native_tags.contrib.op.**abs**(*a*)
abs(a) – Same as abs(a). This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**add**(*a*, *b*)
add(a, b) – Same as a + b. This is a *function tag*.

native_tags.contrib.op.**and_**(*a*, *b*)
and_(a, b) – Same as a & b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**concat**(*a*, *b*)
concat(a, b) – Same as a + b, for a and b sequences. This is a *function tag*.

native_tags.contrib.op.**contains**(*a*, *b*)
contains(a, b) – Same as b in a (note reversed operands). This is a *function tag*.

native_tags.contrib.op.**countOf**(*a*, *b*)
countOf(a, b) – Return the number of times b occurs in a. This is a *function tag*.

native_tags.contrib.op.**delitem**(*a*, *b*)
delitem(a, b) – Same as del a[b]. This is a *function tag*.

native_tags.contrib.op.**delslice**(*a*, *b*, *c*)
delslice(a, b, c) – Same as del a[b:c]. This is a *function tag*.

native_tags.contrib.op.**div**(*a*, *b*)
div(a, b) – Same as a / b when __future__.division is not in effect. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**eq**(*a*, *b*)
eq(a, b) – Same as a==b. This is a *comparison tag*.

native_tags.contrib.op.**floordiv**(*a*, *b*)
floordiv(a, b) – Same as a // b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**ge**(*a*, *b*)
ge(a, b) – Same as a>=b. This is a *comparison tag*.

native_tags.contrib.op.**getitem**(*a*, *b*)
getitem(a, b) – Same as a[b]. This is a *function tag*.

native_tags.contrib.op.**getslice**(*a*, *b*, *c*)
getslice(a, b, c) – Same as a[b:c]. This is a *function tag*.

native_tags.contrib.op.**gt**(*a*, *b*)
gt(a, b) – Same as a>b. This is a *comparison tag*.

native_tags.contrib.op.**index**(*a) – Same as a.__index__()*
This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**indexOf**(*a*, *b*)
indexOf(a, b) – Return the first index of b in a. This is a *function tag*.

native_tags.contrib.op.**inv**(*a*)
inv(a) – Same as ~a. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**is_**(*a*)
is_(a, b) – Same as a is b. This is a *comparison tag*.

native_tags.contrib.op.**is_not**(*a*)
is_not(a, b) – Same as a is not b. This is a *comparison tag*.

native_tags.contrib.op.**le**(*a*, *b*)
le(a, b) – Same as a<=b. This is a *comparison tag*.

native_tags.contrib.op.**lshift**(*a*, *b*)
lshift(a, b) – Same as a << b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**lt**(*a*, *b*)
lt(a, b) – Same as a<b. This is a *comparison tag*.

native_tags.contrib.op.**mod**(*a*, *b*)
mod(a, b) – Same as a % b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**mul**(*a*, *b*)
mul(a, b) – Same as a * b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**ne**(*a*, *b*)
ne(a, b) – Same as a!=b. This is a *comparison tag*.

native_tags.contrib.op.**neg**(*a*)
neg(a) – Same as -a. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**not_**(*a*)
not_(a) – Same as not a. This is a *comparison tag*.

native_tags.contrib.op.**or_**(*a*, *b*)
or_(a, b) – Same as a | b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**pos**(*a*)
pos(a) – Same as +a. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**pow**(*a*, *b*)
pow(a, b) – Same as a ** b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**repeat**(*a*, *b*)
repeat(a, b) – Return a * b, where a is a sequence, and b is an integer. This is a *function tag*.

native_tags.contrib.op.**rshift**(*a*, *b*)
rshift(a, b) – Same as a >> b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**setitem**(*a*, *b*, *c*)
setitem(a, b, c) – Same as a[b] = c. This is a *function tag*.

native_tags.contrib.op.**setslice**(*a*, *b*, *c*, *v*)
setslice(a, b, c, d) – Same as a[b:c] = d. This is a *function tag*.

native_tags.contrib.op.**sub**(*a*, *b*)
sub(a, b) – Same as a - b. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**truediv**(*a*, *b*)
truediv(a, b) – Same as a / b when __future__.division is in effect. This is a *function tag*. This is a *comparison tag*.

native_tags.contrib.op.**xor**(*a*, *b*)
xor(a, b) – Same as a ^ b. This is a *function tag*. This is a *comparison tag*.

1.8.13 `pygmentize` - using `Pygments` to highlight source code

native_tags.contrib.pygmentize.**highlight**(*code*, *lexer*, ***kwargs*)

Returns highlighted code `div` tag from `HtmlFormatter` Lexer is guessed by `lexer` name arguments are passed into the formatter

Syntax:

```
{% highlight [source code] [lexer name] [formatter options] %}
```

Example:

```
{% highlight 'print "Hello World"' python linenos=true %}
```

This is a *function tag*.

native_tags.contrib.pygmentize.**highlight_block**(*context*, *nodelist*, *lexer*, ***kwargs*)

Code is nodelist `rendered` in `context` Returns highlighted code `div` tag from `HtmlFormatter` Lexer is guessed by `lexer` name arguments are passed into the formatter

Syntax:

```
{% highlight_block [lexer name] [formatter options] %}
    ... source code ..
{% endhighlight_block %}
```

Example:

```
{% highlight_block python linenos=true %}
    print '{{ request.path }}'
{% endhighlight_block %}
```

This is a *block tag*.

native_tags.contrib.pygmentize.**highlight_style**(*cssclass='highlight'*, ***kwargs*)

Returns the CSS from the `HtmlFormatter`. `cssclass` is the name of the `div` css class to use

Syntax:

```
{% highlight_style [cssclass] [formatter options] %}
```

Example:

```
{% highlight_style code linenos=true %}
```

This is a *function tag*.

1.8.14 `rand` - random number operations

native_tags.contrib.rand.**randchoice**(**seq*)
Choose a random element from a non-empty sequence.

Syntax:

```
{% randchoice [sequence] %}
{% randchoice [choice1 choice2 ...] %}
```

native_tags.contrib.rand.**randint**(*a*, *b*)
Return random integer in range [a, b], including both end points.

Syntax:

```
{% randint [a] [b] %}
```

native_tags.contrib.rand.**random**() → x in the interval [0, 1).
Syntax:

```
{% random %}
```

native_tags.contrib.rand.**randrange**(**args*, ***kwargs*)
Choose a random item from range(start, stop[, step]).

> This fixes the problem with randint() which includes the endpoint; in Python this is usually not what you want. Do not supply the 'int', 'default', and 'maxwidth' arguments.

Syntax:

```
{% randrange [stop] [options] %}
{% randrange [start] [stop] [step] [options] %}
```

1.8.15 `regex` - regular expression pattern operations

native_tags.contrib.regex.**matches**(*pattern*, *text*)
String comparison. True if string `text` matches regex `pattern` This is a *comparison tag*.

native_tags.contrib.regex.**search**(*pattern*, *text*)
Regex pattern search. Returns match if `pattern` is found in `text` This is a *function tag*.

native_tags.contrib.regex.**substitute**(*search*, *replace*, *text*)
Regex substitution function. Replaces regex `search` with `replace` in `text` This is a *function tag*.

1.8.16 `serializers` - wrapper for `django.core.serializers`

native_tags.contrib.serializers.**serialize**(*format*, *queryset*, ***options*)
Serialize a queryset (or any iterator that returns database objects) using a certain serializer.

native_tags.contrib.serializers.**serialize_json**(*queryset*)

native_tags.contrib.serializers.**serialize_xml**(*queryset*)

native_tags.contrib.serializers.**serialize_python**(*queryset*)

native_tags.contrib.serializers.**serialize_yaml**(*queryset*)

CHAPTER 2

Indices and tables

- *genindex*
- *modindex*
- *search*

Python Module Index

n