django-best-templatetags Documentation

Release 0.0.4

Eric Lapouyade

May 06, 2020

Contents

Best tags and filters for Django templates

CHAPTER 1

Installation

Install with pip:

```
pip install best_templatetags
```

Then declare the app in your settings.py

```
INSTALLED_APPS = [
...
    'best_templatetags',
]
```

To use the filters, add in your template:

```
{% load best_filters %}
```

To use the tags, add in your template:

```
{% load best_tags %}
```

CHAPTER 2

Filters

age	give the age in year
basename	give the basename of the path
dirname	give the directory name of the path
divide	Divide by a value
get_key	Give access to a dict value with a key contained in a var
listsort	Sort a list or a list of lists/tuples
listsortreversed	Sort a list or a list of lists/tuples in reversed order
multiply	Multiply by a value
replace	Replace a substring
resub	regex substitute a substring
sanitizetags	Remove all tags that is not in the allowed list
truncat	truncate the string at the specified pattern

2.1 best_filters.age

best_filters.**age**(*bday*, *ref_date=None*)

give the age in year

Argument is optionnal. If not set, the refererence day is today

Example

```
>>> c = {'user_birthdate':datetime(2006,11,9),
...       'mytoday' : datetime(2018,1,21) }
>>> t = '{% load best_filters %}{{ user_birthdate|age:mytoday }} years old'
>>> Template(t).render(Context(c))
'11 years old'
```

2.2 best_filters.basename

best_filters.**basename**(*str*)
give the basename of the path

It uses os.path.basename()

Example

```
>>> c = {'mypath':'/a/b/c/myfile.extension'}
>>> t = '{% load best_filters %}{{ mypath|basename }}'
>>> Template(t).render(Context(c))
'myfile.extension'
```

2.3 best_filters.dirname

best_filters.**dirname**(*str*)
give the directory name of the path

It uses os.path.dirname()

Example

```
>>> c = {'mypath':'/a/b/c/myfile.extension'}
>>> t = '{% load best_filters %}{{ mypath|dirname }}'
>>> Template(t).render(Context(c))
'/a/b/c'
```

2.4 best_filters.divide

best_filters.**divide**(*val*, *arg*)
Divide by a value

Examples

```
>>> c = {'myval':50}
>>> t = '{% load best_filters %}{{ myval|divide:2|floatformat:0 }}'
>>> Template(t).render(Context(c))
'25'
```

```
>>> c = {'mystr':100}
>>> t = '{% load best_filters %}{{ mystr|divide:3|floatformat:2 }}'
>>> Template(t).render(Context(c))
'33.33'
```

2.5 best_filters.get_key

best_filters.**get_key**(*object*, *attr*)

Give access to a dict value with a key contained in a var

Example :

```
>>> c = {'countries': {'FR':'France','US':'United States'},
...       'country':'FR'}
>>> t = '{% load best_filters %}Country:{{ countries|get_key:country }}'
>>> Template(t).render(Context(c))
'Country:France'
```

2.6 best_filters.listsort

best_filters.**listsort**(*lst*, *col=None*)

Sort a list or a list of lists/tuples

If no argument is given, the list is sorted like python does by default. If an argunment is given (int), the filter is expecting a list of lists/tuples and will sort following the column 'col' order

Examples :

```
>>> c = { 'lst': ['a','c','b'] }
>>> t = '''{% load best_filters %}
... sorted : {% for i in lst|listsort %}{{i}}{% endfor %}'''
>>> Template(t).render(Context(c))
'\nsorted : abc'
```

```
>>> c = { 'lst': [('a',3),('c',1),('b',2)] }
>>> t = '''{% load best_filters %}
... sorted : {% for i in lst|listsort:1 %}{{i|safe}}{% endfor %}'''
>>> Template(t).render(Context(c))
"\nsorted : ('c', 1)('b', 2)('a', 3)"
```

2.7 best_filters.listsortreversed

best_filters.**listsortreversed**(*lst*, *col=None*)

Sort a list or a list of lists/tuples in reversed order

Same as *listsort()* except that is reverse the order

Examples :

```
>>> c = { 'lst': ['a','c','b'] }
>>> t = '''{% load best_filters %}
... sorted : {% for i in lst|listsortreversed %}{{i}}{% endfor %}'''
>>> Template(t).render(Context(c))
'\nsorted : cba'
```

```
>>> c = { 'lst': [('a',3),('b',1),('c',2)] }
>>> t = '''{% load best_filters %}
... sorted : {% for i in lst|listsortreversed:1 %}{{i|safe}}{% endfor %}'''
```

```
>>> Template(t).render(Context(c))
"\nsorted : ('a', 3)('c', 2)('b', 1)"
```

2.8 best_filters.multiply

best_filters.**multiply**(*val*, *arg*)

Multiply by a value

Examples

```
>>> c = {'myval':50}
>>> t = '{% load best_filters %}{{ myval|multiply:1024 }}'
>>> Template(t).render(Context(c))
'51200'
```

```
>>> c = {'mystr':'*'}
>>> t = '{% load best_filters %}{{ mystr|multiply:8 }}'
>>> Template(t).render(Context(c))
'********'
```

2.9 best_filters.replace

best_filters.**replace**(*str*, *arg*)

Replace a substring

The replacement syntax is : <chosen separator><string to replace><separator><replacement string>

Examples

```
>>> c = {'mystr':'hello world'}
>>> t = '{% load best_filters %}{{ mystr|replace:"/world/eric" }}'
>>> Template(t).render(Context(c))
'hello eric'
```

```
>>> c = {'mypath':'/home/theuser/projects'}
>>> t = '{% load best_filters %}{{ mypath|replace:",/home,/Users" }}'
>>> Template(t).render(Context(c))
'/Users/theuser/projects'
```

2.10 best_filters.resub

best_filters.**resub**(*str*, *arg*)

regex substitute a substring

The substitution syntax is : <chosen separator><regex pattern to search><separator><replacement string>

Examples

```
>>> c = {'mystr':'hello world'}
>>> t = '{% load best_filters %}{{ mystr|resub:"/ .*/ eric" }}'
>>> Template(t).render(Context(c))
'hello eric'
```

```
>>> c = {'mypath':'/home/theuser/projects'}
>>> t = r'''{% load best_filters %}
... {{ mypath|resub:",/home/([^/]*)/projects,login=\1" }}'''
>>> Template(t).render(Context(c))
'\nlogin=theuser'
```

2.11 best_filters.sanitizetags

best_filters.**sanitizetags**(*value*, *allowed_tags=None*)

Remove all tags that is not in the allowed list

Argument should be in form 'tag1:attr1:attr2 tag2:attr1 tag3', where tags are allowed HTML tags, and attrs are the allowed attributes for that tag.

In the example above, it means accepted tags are : <tag1 attr1="..." attr2="..."> and <tag2 attr1="..."> and <tag3> All other HTML tags an attributes will be removed.

for example <tag2 attr1="..." attr3="..."> <tag4 ...> will be replaced by just <tag2 attr1="...">

The filter also unconditionnaly removes attributes having values starting with 'javascript:' to avoid malicious code.

If No argument is given, the filter will look for SANITIZETAGS_ALLOWED in settings or will use this default value: 'a:href:name b u p i h1 h2 h3 hr img:src table tr td th code'

Notes

- The output is marked as a safe string.
- If the HTML given has not a correct syntax, an error html message is displayed instead of the original value.
- Only tags are sanitized, not the text in between

Examples

```
>>> c = {'comment':'''<a href="x" name="y" id="z"></a> <b></b> <u></u>
... <p></p> <i></i> <h1></h1> <h2></h2> <h3></h3> <hr>
... <img src="x" id="y"> <table></table> <tr></tr> <td></td> <th></th>
... <code></code> <unkown_tag></unknown_tag> <div></div>'''}
>>> t = '{% load best_filters %}{{ comment|sanitizetags}}'
>>> print(Template(t).render(Context(c)))
<a href="x" name="y"></a> <b></b> <u></u>
<p></p> <i></i> <h1></h1> <h2></h2> <h3></h3> <hr/>
<img src="x"/> <table></table> <tr></tr> <td></td> <th></th>
<code></code>
```

```
>>> c = {'comment':'My comment <b>with</b> <a href="spam">ads</a>'}
>>> t = '{% load best_filters %}{{ comment|sanitizetags:"B u i"}}'
>>> Template(t).render(Context(c))
'My comment <b>with</b> ads'
```

```
>>> c = {'comment':
... '<i>Go</i> <a badattrib="xx" href="google.com">here</a>'}
>>> t = '{% load best_filters %}{{ comment|sanitizetags:"a:href"}}'
>>> Template(t).render(Context(c))
'Go <a href="google.com">here</a>'
```

```
>>> c = {'comment':'<b><i><u>nested tags</u></i></u>'}
>>> t = '{% load best_filters %}{{ comment|sanitizetags:"b u"}}'
>>> Template(t).render(Context(c))
'<b><u>nested tags</u></b>'
```

```
>>> c = {'comment':'''<a href="javascript:hack_me();" name="iambad">
... <a href="http://google.com" name="iamgood">'''}
>>> t = '{% load best_filters %}{{ comment|sanitizetags:"a:href:name"}}'
>>> Template(t).render(Context(c))
'<a name="iambad">\n<a href="http://google.com" name="iamgood"></a></a>'
```

2.12 best_filters.truncat

best_filters.**truncat**(*str*, *pattern*)

truncate the string at the specified pattern

Useful with filters timesince and timeuntil pattern is a regex expression string Do not forget to escape the dot (.) if it the char you want to search

Examples

```
>>> c = {'str':'abc...xyz'}
>>> t = '{% load best_filters %}{{ str|truncat:"\." }}'
>>> Template(t).render(Context(c))
'abc'
```

```
>>> c = {'t1':datetime(1789,7,14),'t2':datetime(2018,1,21)}
>>> t = '''{% load best_filters %}
... timesince with 2 terms : {{ t1|timesince:t2 }}
... timesince with 1 term : {{ t1|timesince:t2|truncat:"," }}'''
>>> print(Template(t).render(Context(c)))

timesince with 2 terms : 228 years, 6 months
timesince with 1 term : 228 years
```

CHAPTER 3

Tags

extend_url	Update url parameters
hash	Return a hexadecimal md5 digest of a string
render_template	Render a string as it was a Django template
update_url	Update url parameters

3.1 best_tags.extend_url

best_tags.**extend_url**(*url*, ***kwargs*)

Update url parameters

- Not existing parameters are added
- Existing parameters are extended
- parameters with "__del__" value are deleted

Note: It takes care to not have duplicate values for a same parameter. The values taken from the tag parameters are converted to string.

Examples

```
>>> c = {'myurl':'http://a.com/b/c.html?d=1&e=2'}
>>> t = '{% load best_tags %}{% extend_url myurl d=1 e=3 f=4 %}'
>>> Template(t).render(Context(c))
'http://a.com/b/c.html?d=1&e=2&e=3&f=4'
```

```
>>> c = {'myurl':'?d=1&e=2'}
>>> t = '{% load best_tags %}{% extend_url myurl e=3 f=4 %}'
```

```
>>> Template(t).render(Context(c))
'?d=1&e=2&e=3&f=4'
```

```
>>> c = {'myurl':'http://a.com/b/c.html?d=1&e=2'}
>>> t = '{% load best_tags %}{% extend_url myurl d="__del__" e=3 %}'
>>> Template(t).render(Context(c))
'http://a.com/b/c.html?e=2&e=3'
```

3.2 best_tags.hash

best_tags.**hash**(*algorithm*, *str*)

Return a hexadecimal md5 digest of a string

First argument is a string giving the hash algorithm, for example: "md5", "sha1" ... Second argument is the string or variable to hash

Note: string are encoded to utf-8 prior calculating the hash

Example

```
>>> c = {'title':'My worderful document title'}
>>> t = '{% load best_tags %}{% hash "md5" title %}'
>>> Template(t).render(Context(c))
'3ddbd7936634a6a47f978376674dea31'
```

3.3 best_tags.render_template

best_tags.**render_template**(*value*)

Render a string as it was a Django template

It will use the same context as the outer template.

Example

```
>>> c = {'mytemplate':'my value = {{myvar}}',
...      'myvar':'myvalue'}
>>> t = '''{% load best_tags %}My template : {{ mytemplate }}
... with myvar = {{myvar}}
... My template rendered : {% render_template mytemplate %}'''
>>> print(Template(t).render(Context(c)))
My template : my value = {{myvar}}
with myvar = myvalue
My template rendered : my value = myvalue
```

3.4 best_tags.update_url

best_tags.**update_url**(*url*, ***kwargs*)

Update url parameters

- Not existing parameters are added
- Existing parameters are replaced
- parameters with "__del__" value are deleted

Examples

```
>>> c = {'myurl':'http://a.com/b/c.html?d=1&e=2'}
>>> t = '{% load best_tags %}{% update_url myurl e=3 f=4 %}'
>>> Template(t).render(Context(c))
'http://a.com/b/c.html?d=1&e=3&f=4'
```

```
>>> c = {'myurl':'?d=1&e=2'}
>>> t = '{% load best_tags %}{% update_url myurl e=3 f=4 %}'
>>> Template(t).render(Context(c))
'?d=1&e=3&f=4'
```

```
>>> c = {'myurl':'http://a.com/b/c.html?d=1&e=2'}
>>> t = '{% load best_tags %}{% update_url myurl d="__del__" f=4 %}'
>>> Template(t).render(Context(c))
'http://a.com/b/c.html?e=2&f=4'
```

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Index