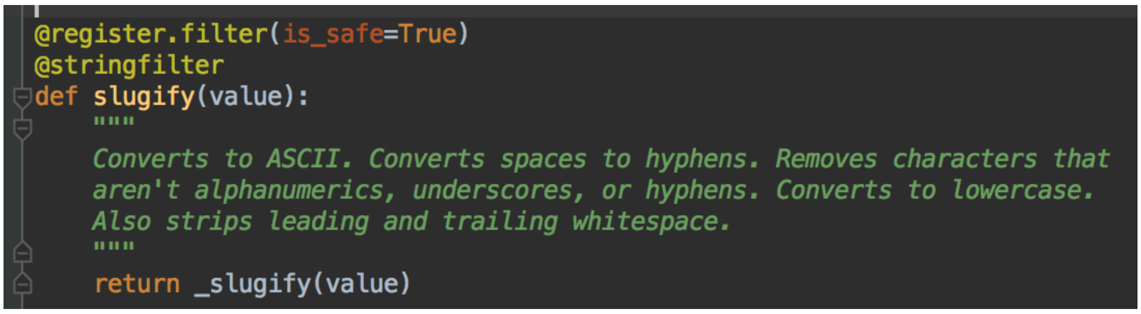
14 장 템플릿 태그와 필터

기본내용 :
https://djangobook.com/basic-template-tags-filters/

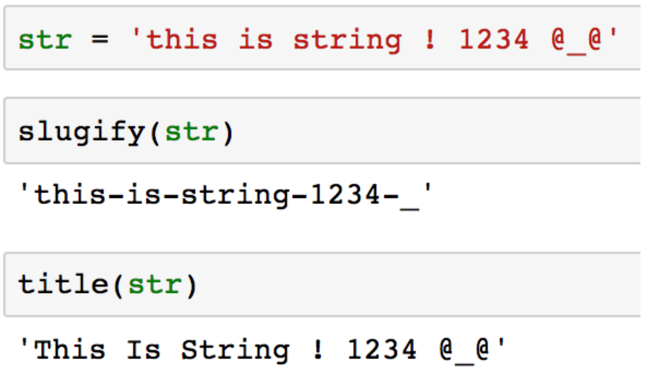
slugify 필터

@register.filter(is_safe=True)
필터 등록을 담당하는 클래스는 django.template.library.py 에 Library 클래스이다.
filter 함수를 통해 함수를 등록할 수 있다.



```
@register.filter(is_safe=True)
@stringfilter
def slugify(value):
    """
    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.
    """
    return _slugify(value)
```

주피터에서는 이런 식으로 테스트도 가능 a



```
str = 'this is string ! 1234 @_@'
```

```
slugify(str)
```

```
'this-is-string-1234-_'
```

```
title(str)
```

```
'This Is String ! 1234 @_@'
```

template filter 는 꼭 2 개의 인자만 받을 수 있는가? :
https://stackoverflow.com/questions/420703/how-do-i-add-multiple-arguments-to-my-custom-template-filter-in-a-django-templat

So, as with query strings, each parameter is separated by '&':

```
{{ attr.name|replace:"cherche=_&remplacement= " }}
```

Then your replace function will now look like this:

```python
from django import template
from django.http import QueryDict

register = template.Library()

@register.filter
def replace(value, args):
    qs = QueryDict(args)
    if qs.has_key('cherche') and qs.has_key('remplacement'):
        return value.replace(qs['cherche'], qs['remplacement'])
    else:
        return value
```

You could speed this up some at the risk of doing some incorrect replacements:

```python
qs = QueryDict(args)
return value.replace(qs.get('cherche',''), qs.get('remplacement',''))
```

결론 : 방법은 있지만 개인적으로는 추천하고 싶지 않다.

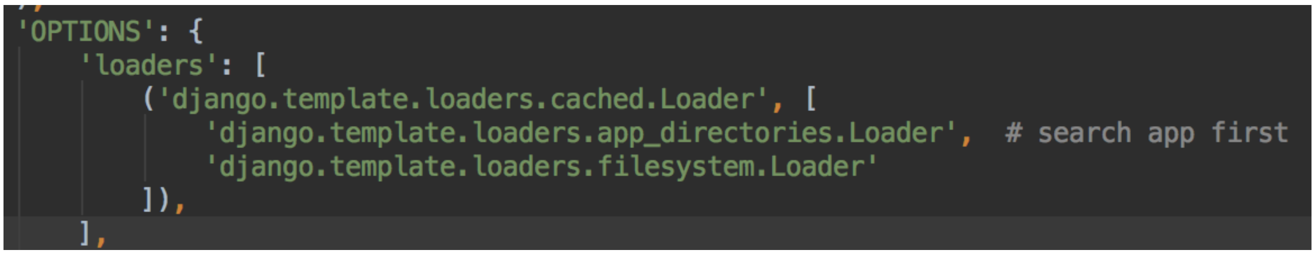
템플릿 캐싱 :
https://docs.djangoproject.com/en/1.11/ref/templates/api/#django.template.loaders.cached.Loader

```python
TEMPLATES = [{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [os.path.join(BASE_DIR, 'templates')],
    'OPTIONS': {
        'loaders': [
            ('django.template.loaders.cached.Loader', [
                'django.template.loaders.filesystem.Loader',
                'django.template.loaders.app_directories.Loader',
                'path.to.custom.Loader',
            ]),
        ],
    },
}]
```

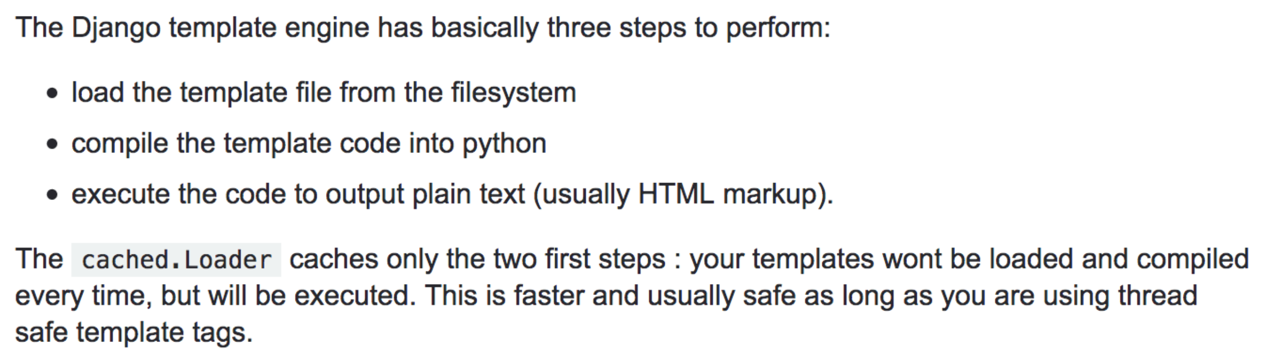
template : 본뜨는 공구, 형판, 보받이, 도리받이, 조선대의 쐐기

템플릿 캐싱의 두가지 방법 :
https://stackoverflow.com/questions/25629831/django-two-ways-of-caching-template-what-is-the-difference

cached.Loader 사용하기



```
'OPTIONS': {
    'loaders': [
        ('django.template.loaders.cached.Loader', [
            'django.template.loaders.app_directories.Loader',  # search app first
            'django.template.loaders.filesystem.Loader'
        ]),
    ],
```



The Django template engine has basically three steps to perform:

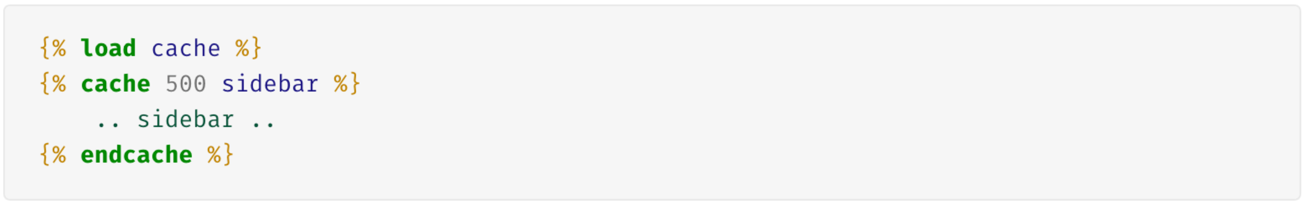
- load the template file from the filesystem
- compile the template code into python
- execute the code to output plain text (usually HTML markup).

The `cached.Loader` caches only the two first steps : your templates wont be loaded and compiled every time, but will be executed. This is faster and usually safe as long as you are using thread safe template tags.

파일 IO 를 줄여줄 뿐, template rendering 에는 영향을 미치지 않는다. html 파일이 변경 된 경우, 서버 재부팅 전에는 반영되지 않으므로 주의해야 한다.

Template fragment caching 사용하기

공식문서 : https://docs.djangoproject.com/en/1.11/topics/cache/#template-fragment-caching



```
{% load cache %}
{% cache 500 sidebar %}
    .. sidebar ..
{% endcache %}
```

template 에서 사용되는 variable 에 이렇게 적용할 수 있지만.. 값이 바뀌면 어찌할까?

참고문서 : http://agiliq.com/blog/2015/08/template-fragment-caching-gotchas/

Variables in cached template fragment

Assuming this is in template.

```
{% cache 300 nums %}
{% for i in nums %}
    <p>i</p>
{% endfor %}
{% endcache %}
```

And assuming we send {'nums': range(100)} from context, then 0 to 99 will be sent in the response.

Now suppose we change context to {'nums': range(1000)}, still for next 5 minutes i.e until the cache expires, 0 to 99 will be sent in the response. 0 to 999 will not be sent in the response.

To fix this, we should use the variable too with the {% cache %} tag. So correct code would be

```
{% cache 300 nums_cache nums %}
{% for i in nums %}
    <p>i</p>
{% endfor %}
{% endcache %}
```

After this whenever context **nums** changes, cache would be reevaluated.

anti pattern

Example 14.2: Implicit Loading of Template Tag Libraries

```
# settings/base.py
TEMPLATES = [
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'OPTIONS': {
        # Don't do this!
        # It's an evil anti-pattern!
        'builtins': ['flavors.templatetags.flavors_tags'],
    },
]
```

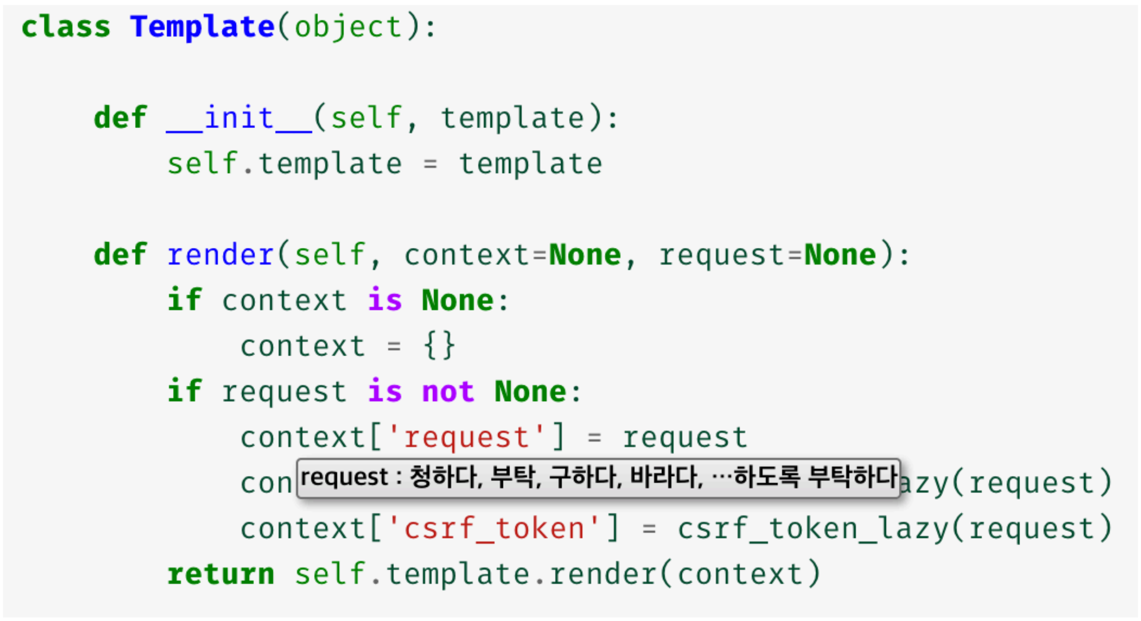
1.11 에서 예제가 바뀌었지만 동작은 동일함. 사용하지 말아야 하는 이유도 동일.

15 장 장고 템플릿과 jinja2

jinja2 tutorial : https://realpython.com/blog/python/primer-on-jinja-templating/

jinja2 csrf_token

code : https://docs.djangoproject.com/en/1.8/_modules/django/template/backends/jinja2/#Jinja2

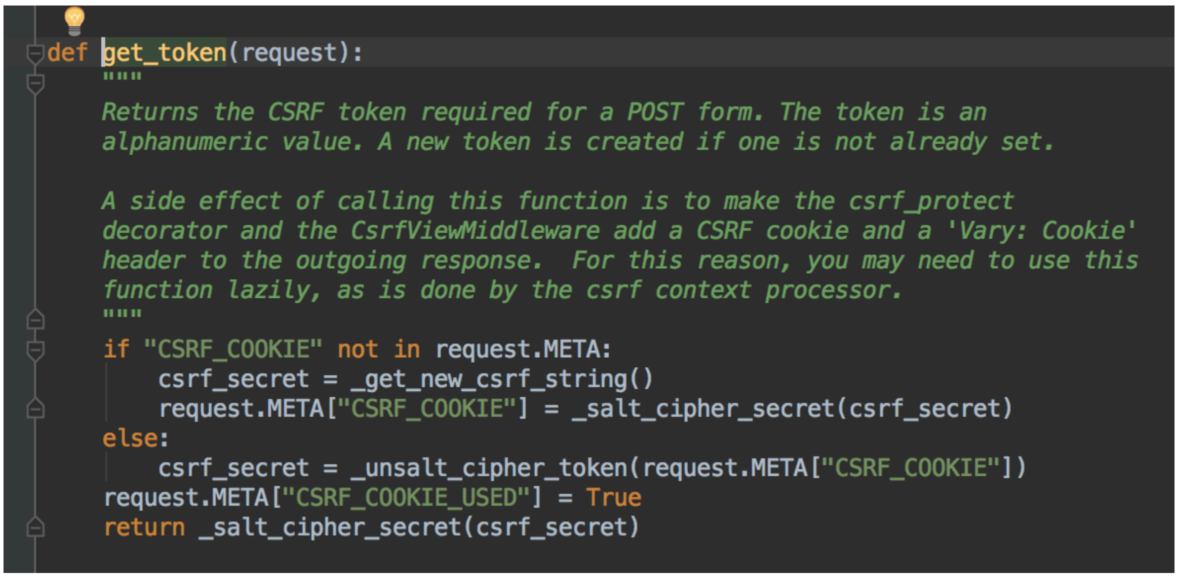


```
class Template(object):

    def __init__(self, template):
        self.template = template

    def render(self, context=None, request=None):
        if context is None:
            context = {}
        if request is not None:
            context['request'] = request
            con                                        azy(request)
            context['csrf_token'] = csrf_token_lazy(request)
        return self.template.render(context)
```

하지만 그 뒷단은 django 와 동일함

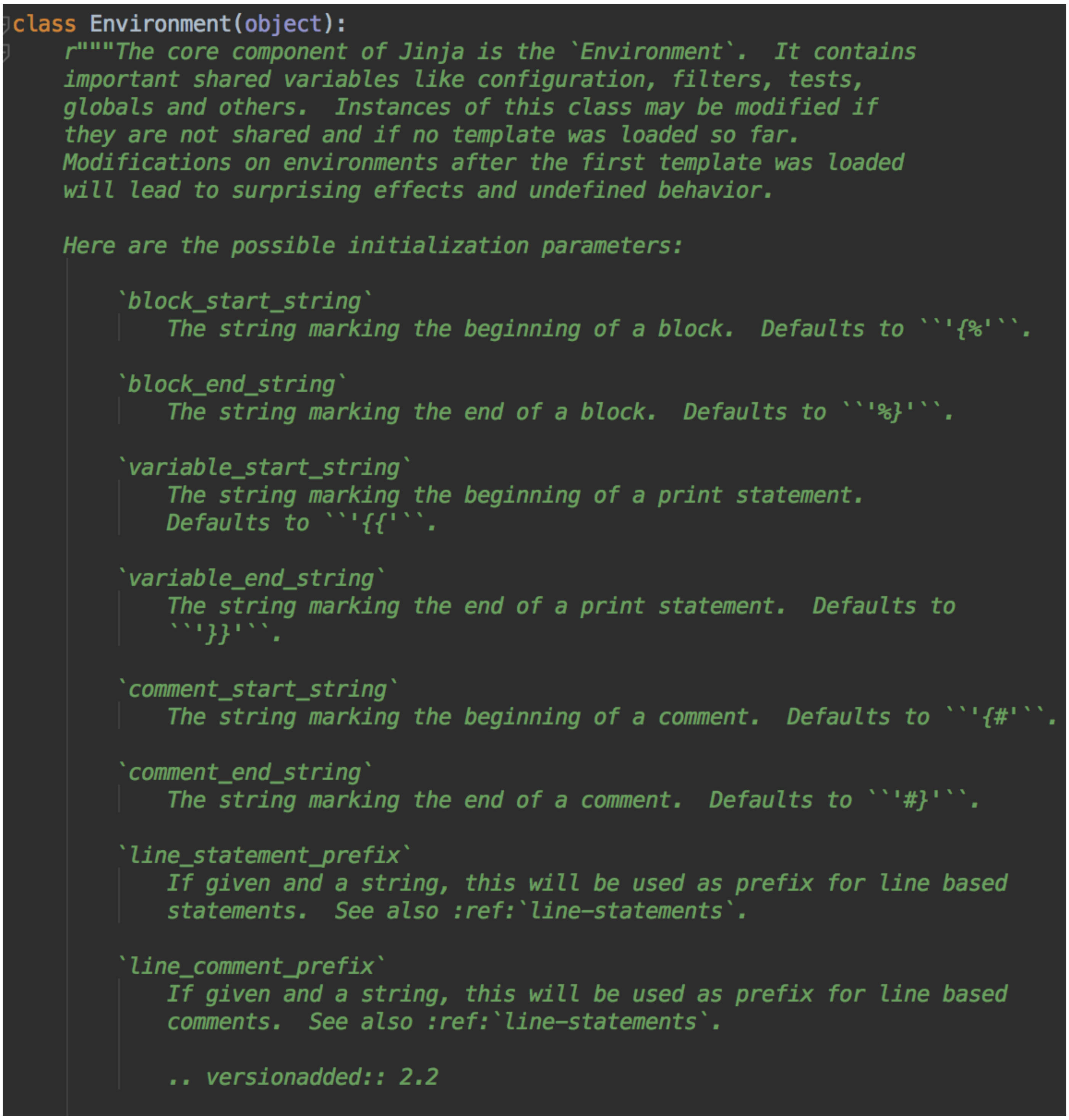


```
def get_token(request):
    """
    Returns the CSRF token required for a POST form. The token is an
    alphanumeric value. A new token is created if one is not already set.

    A side effect of calling this function is to make the csrf_protect
    decorator and the CsrfViewMiddleware add a CSRF cookie and a 'Vary: Cookie'
    header to the outgoing response.  For this reason, you may need to use this
    function lazily, as is done by the csrf context processor.
    """
    if "CSRF_COOKIE" not in request.META:
        csrf_secret = _get_new_csrf_string()
        request.META["CSRF_COOKIE"] = _salt_cipher_secret(csrf_secret)
    else:
        csrf_secret = _unsalt_cipher_token(request.META["CSRF_COOKIE"])
    request.META["CSRF_COOKIE_USED"] = True
    return _salt_cipher_secret(csrf_secret)
```

jinja2 environment : http://jinja.pocoo.org/docs/dev/api/#jinja2.Environment

jinja2.environment.py



```python
class Environment(object):
    r"""The core component of Jinja is the `Environment`.  It contains
    important shared variables like configuration, filters, tests,
    globals and others.  Instances of this class may be modified if
    they are not shared and if no template was loaded so far.
    Modifications on environments after the first template was loaded
    will lead to surprising effects and undefined behavior.

    Here are the possible initialization parameters:

        `block_start_string`
            The string marking the beginning of a block.  Defaults to ``'{%'``.

        `block_end_string`
            The string marking the end of a block.  Defaults to ``'%}'``.

        `variable_start_string`
            The string marking the beginning of a print statement.
            Defaults to ``'{{'``.

        `variable_end_string`
            The string marking the end of a print statement.  Defaults to
            ``'}}'``.

        `comment_start_string`
            The string marking the beginning of a comment.  Defaults to ``'{#'``.

        `comment_end_string`
            The string marking the end of a comment.  Defaults to ``'#}'``.

        `line_statement_prefix`
            If given and a string, this will be used as prefix for line based
            statements.  See also :ref:`line-statements`.

        `line_comment_prefix`
            If given and a string, this will be used as prefix for line based
            comments.  See also :ref:`line-statements`.

            .. versionadded:: 2.2
```