

Djagno 기반의 웹프로그래밍

이 소 영

yisy0703@naver.com

Web Frameworks for Python(<https://wiki.python.org/moin/WebFrameworks>)

장고 공식 사이트(<https://www.djangoproject.com/>)

장고 공식 소스 저장소(<http://github.com/django/django>)

장고 참조 문서 영어(<https://docs.djangoproject.com/en/3.2/>)

Agenda

Django(장고) 기반의 파이썬 웹 프로그래밍

Ch01. Django 시작하기

1. Django 란?
2. 개발 환경 구축
3. Django 구조

Ch02. Django App

1. Django Project
2. Model
3. View

Ch03. Model

1. Model 속성 및 옵션
2. Relationship
3. Migrations
4. Admin App

Ch04. Django SQL

1. Django shell
2. Manager & QuerySet
3. 조회 SQL
4. 생성/수정/삭제 SQL
5. Django-Debug-Toolbar

Ch05. Template

1. Template Loader
2. URL Dispatcher
3. Template 상속
4. Template Engines
5. Template Filter

Ch06. Django View

1. View 기본
2. View 활용

Ch07. Django Form

1. HTML form
2. CSRF
3. HttpRequest/HttpResponse
4. Django Form
5. Django Model Form
6. Form Validation

Ch08. File 관리

1. Static Files
2. Media Files
3. Image Thumbnail

Ch09. 사용자 인증

1. Auth App
2. 회원가입 구현
3. 로그인/아웃 구현
4. Oauth 라이브러리 활용

Ch 05 Template

1. Template Loader
2. URL Dispatcher
3. Template 상속
4. Template Engines
5. Template Filter

Chapter 05. Template

- Template API
(<https://docs.djangoproject.com/en/3.2/ref/templates/api/>)
- `resolve_url()`
(<https://github.com/django/django/blob/master/django/shortcuts.py#LC119>)
- Built-in template tags and filters
(<https://docs.djangoproject.com/en/3.2/ref/templates/builtins>)
- Custom template tags and filters
(<https://docs.djangoproject.com/en/3.2/howto/custom-template-tags/>)
- Date Format
(<https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#date>)
- Time Format
(<https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#time>)



1. Template Loader



TEMPLATES

1. Template Loader

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [  
            os.path.join(BASE_DIR, 'myproject', 'templates'),  
        ],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
                'django.template.context_processors.debug',  
                'django.template.context_processors.request',  
                'django.contrib.auth.context_processors.auth',  
                'django.contrib.messages.context_processors.messages',  
            ],  
        },  
    },  
]
```

Django Template Loader

1. Template Loader

1. `django.template.loaders.app_directories.Loader` 실행
 2. `django.template.loaders.filesystem.Loader` 실행
- <https://docs.djangoproject.com/en/3.2/ref/templates/api>
 - Template 파일 사용 함수
 - `render` : `HttpResponse` 객체 리턴
 - `render_to_string` : 문자열 리턴

```
response = render(request, 'blog/post_list.html', context_params)
```

```
welcome_message = render_to_string('accounts/signup_welcome.txt', context_params)
```

app_directories.Loader

1. Template Loader

- settings.INSTALLED_APPS 에 설정된 앱/templates 에서 템플릿 파일 검색

```
INSTALLED_APPS = [  
    'django.contrib.admin',  
    'django.contrib.auth',  
    'django.contrib.contenttypes',  
    'django.contrib.sessions',  
    'django.contrib.messages',  
    'django.contrib.staticfiles',  
    'django_extensions',  
    'blog',  
]
```



/templates

filesystem.Loader

1. Template Loader

- 프로젝트 레벨의 템플릿은 settings.py 에 위치 지정

```
TEMPLATES = [{  
    'DIRS': [os.path.join(BASE_DIR, '프로젝트명', 'templates'), ],  
}]
```

- template 파일 검색

- 앱의 templates을 순서대로 검색
- 앱에서 검색 실패시 프로젝트의 templates 검색
- 프로젝트의 templates 검색 실패시 TemplateDoesNotExist 발생



2. URL Dispatcher



Django의 요청 처리 순서

2. URL Dispatcher

1. settings.py의 ROOT_URLCONF 설정 값으로 root URLConf 모듈을 결정함
2. root URLConf 모듈을 로드하고 urlpatterns 변수를 찾음. urlpatterns 설정값은 django.urls.path() 또는 django.urls.re_path() 인스턴스 목록임
3. urlpatterns 순서대로 검색 후 요청 URL과 일치하는 첫번째 패턴에서 멈춤
4. 일치하는 첫번째 패턴의 뷰 함수를 호출함. 호출시 다음 인자를 전달함
 - HttpRequest 인스턴스
 - 이름이 지정되지 않은 인자는 위치 기반으로 전달
 - 키워드 인자는 path() 또는 re_path() 의 kwargs 값에 설정

url pattern

2. URL Dispatcher

- URL에서 view의 인자로 사용되는 값은 꼭써 괄호<>를 사용한다.
- URL에서 view의 인자로 사용되는 값의 타입을 지정할 수 있음.
<int:year>로 지정하면 정수로 처리됨.
- <>를 사용하지 않는 경우는 / 를 제외한 모든 문자열이 일치해야 함.
- 첫번째 /는 추가할 필요 없음. URL 자체적으로 지정되기 때문임.
"/articles" 대신 "articles"로 지정해야함.

URLConf

2. URL Dispatcher

```
from django.urls import path
from . import views
urlpatterns = [
    path('articles/2003/', views.special_case_2003),
    path('articles/<int:year>', views.year_archive),
    path('articles/<int:year>/<int:month>', views.month_archive),
    path('articles/<int:year>/<int:month>/<slug:slug>', views.article_detail),
]
```

```
/articles/2005/03/ → views.month_archive(request, year=2005, month=3)
/articles/2003/ → views.special_case_2003(request)
/articles/2003/03/building-a-django-site/ →
views.article_detail(request, year=2003, month=3, slug="building-a-django-site")
```

URL Reverse

2. URL Dispatcher

- URL pattern을 사용하면 URL 변경 시 소스를 수정해야 함
- URL Reverse는 소스 수정없이 유연성 있게 URL을 변경할 수 있도록 함
- URL Reverse는 URL pattern의 이름과 앱의 이름을 사용하여 URL을 지정함

URL Reverse 함수

2. URL Dispatcher

- reverse 함수
 - 리턴값 : str
 - 검색 실패 : NoReversMatch 발생
- resolve_url 함수
 - 리턴값 : str
 - 검색 실패 : URL 문자열 반환
- redirect 함수
 - 리턴값 : str
 - 검색 실패 : URL 문자열을 URL로 판단
 - 내부적으로 resolve_url 함수 사용
- url template tag
 - 리턴값 : str
 - 내부적으로 reverse 함수 사용

URL Reverse 함수

2. URL Dispatcher

```
from django.urls.base import reverse
reverse('article:list')           → '/article/'
reverse('article:detail', args=[2]) → '/article/2/detail'
reverse('article:detail', kwargs={'id':3}) → '/article/3/detail'
```

```
from django.shortcuts import resolve_url, redirect
resolve_url('article:list')       → '/article/'
resolve_url('article:detail', 2)   → '/article/2/detail'
resolve_url('article:detail', id=3) → '/article/3/detail'
```

잘못된 URL

```
reverse('/hello/')                → NoReverseMatch 발생
resolve_url('/hello/')             → '/hello/'
```

```
redirect('article:detail', 2)
<HttpResponseRedirect status_code=302, "text/html; charset=utf-8", url="/article/2/detail">
```


url pattern 사용

2. URL Dispatcher

```
# urls.py
path('<id>/detail/', views.detail),

# views.py
def detail(request, id):
    article = Article.objects.get(id=id)
    return render(request, 'article/article_detail.html', {'article':article})
```

```
# article/article_list.html

{% for article in article_list %}
    <li><a href="/article/{{article.id}}/detail/">
        {{article.title}}</a></li>
{% endfor %}

# article/article_detail.html

<h1>{{article.title}}</h1>
{{article.body | linebreaks}}
```

url reverse 사용

2. URL Dispatcher

```
# article/urls.py
app_name = "article"
urlpatterns = [
    path('', views.article_list, name="list"),
    path('<id>/detail/', views.detail, name='detail'),
]
```

```
# templates/article/article_list.html
<a href="{% url 'article:detail' article.id %}">
    {{article.id}}
    {{article.title}}
</a>
```

redirect

2. URL Dispatcher

```
# myproject/urls.py
from django.shortcuts import redirect
def root(request):
    return redirect('article:list')

urlpatterns = [
    path("", root, name='root'),
]
```

웹브라우저에서 다음 URL로 접속
<http://127.0.0.1:8000>
views.article_list 함수 실행됨

get_absolute_url 함수

2. URL Dispatcher

```
# article/models.py
from django.urls import reverse

class Article(models.Model):

    def get_absolute_url(self):
        return reverse('article:detail', args=[self.id])
```

```
# django.shortcuts의 resolve_url 함수
def resolve_url(to, *args, **kwargs):
    return to.get_absolute_url()
```

get_absolute_url 함수

2. URL Dispatcher

```
from article.models import Article
article = Article.objects.first()
reverse('article:detail', args=[article.id]) #'/article/1/detail'
resolve_url('article:detail', article.id)    #'/article/1/detail'
article.get_absolute_url()                    #'/article/1/detail'
resolve_url(article)                          #'/article/1/detail'
redirect(article)
<HttpResponseRedirect status_code=302, "text/html; charset=utf-8", url="/article/1/detail">
```



3. Template 상속



Template 상속

3. Template 상속

- 코드의 재사용성, 유지 보수성 좋음
- 부모 템플릿
 - 전체 레이아웃과 자식 템플릿이 재정의 할 block 지정
- 자식 템플릿
 - 상속받는 템플릿의 block 영역만 재정의 가능
 - block 영역이 아닌 경우는 무시
- 템플릿 상속 문법
 - `{% extends "부모템플릿경로" %}`
 - 반드시 첫줄에 위치해야 함

Template 상속을 적용하지 않은 예제

3. Template 상속

```
# article_list.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" >
    <title>Django Example</title>
</head>
<body>
    <h1>Article</h1>
    <form action="" method="get">
        <input type="text" name="q" value="{{q}}">
        <input type="submit" value="검색">
    </form>
    {% for article in article_list %}
        <li>
            <a href="{%url 'article:detail' article.id %}">
                {{article.id}}. {{article.title}}
            </a>
        </li>
    {% endfor %}
    <hr>
    Django Lecture 2022
</body>
</html>
```


Template 상속을 적용하지 않은 예제

3. Template 상속

```
# article_detail.html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Django Example</title>
</head>
<body>
    <h1>Article</h1>
    <h2>{{article.id}}. {{article.title}}</h2>
    {{article.body | linebreaks }}
    <p>
        <a href="#수정">[수정]</a>
        <a href="#삭제">[삭제]</a>
        <a href="{% url 'article:list' %}">[목록]</a>
    </p>
    <hr>
    Django Lecture 2022
</body>
</html>
```

Template 상속을 적용한 예제

3. Template 상속

```
# article/templates/article/layout.html
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>{% block title %}Django Example{% endblock %}</title>
</head>
<body>
    <h1>Article</h1>
    {% block content %}
    {% endblock %}
    <hr>
    Django Lecture 2022
</body>
</html>
```

Template 상속을 적용한 예제(자식 template)

3. Template 상속

```
# article_list.html
```

```
{% extends "article/layout.html" %}
```

```
{% block title %}
```

```
    Article list
```

```
{% endblock %}
```

```
{% block content %}
```

```
    <form action="" method="get">
```

```
        <input type="text" name="q" value="{{q}}">
```

```
        <input type="submit" value="검색">
```

```
    </form>
```

```
    {% for article in article_list %}
```

```
        <li>
```

```
<!--          <a href="/article/{{article.id}}/detail/">-->
```

```
          <a href="{%url 'article:detail' article.id %}">
```

```
            {{article.id}}. {{article.title}}
```

```
          </a>
```

```
        </li>
```

```
    {% endfor %}
```

```
{% endblock %}
```

Template 상속을 적용한 예제(자식 template)

3. Template 상속

```
# article_detail.html
```

```
{% extends "article/layout.html" %}
```

```
{% block title %}  
    {{article.title}} 상세보기  
{% endblock %}
```

```
{% block content %}  
    <h2>{{article.id}}. {{article.title}}</h2>  
    {{article.body | linebreaks }}  
    <p>  
        <a href="#수정">[수정]</a>  
        <a href="#삭제">[삭제]</a>  
        <a href="{% url 'article:list' %}">[목록]</a>  
    </p>  
{% endblock %}
```

2단계 상속

3. Template 상속

- 1단계 : 프로젝트 레벨의 레이아웃 템플릿
- 2단계 : 각 앱별 레이아웃 템플릿

```
1. myproject/templates/layout.html 작성
2. settings.py
    TEMPLATES = [
        'DIRS': [
            os.path.join(BASE_DIR, 'myproject', 'templates'),
        ],
    ]
3. article_list.html
    {% extends "layout.html" %}
```



4. Template Engines



Template Engines

4. Template Engines

- Django Template Engine : Django 기본
- Jinja2 : 써드파트 엔진, django에서 최소한 지원
 - Django Template Engine과 비슷
- Django-jinja
- Mako, HamIPy

Django Template Engines 문법

4. Template Engines

{% 태그 인자 %}

```
{% extends "base.html" %}
```

```
{# 파이썬 로직 불가 #}
```

```
{% for row in rows %}
```

```
    <tr>
```

```
        {% for name in row %}
```

```
            <td>{{ name }}</td>
```

```
        {% endfor %}
```

```
    </tr>
```

```
{% endfor %}
```


Variables

4. Template Engines

- `{{ first_name }}`
- `{{ mydict.key }}` : dict의 key에 attr처럼 접근
- `{{ myobj.attr }}`
- `{{ myobj.func }}` : 함수호출. 인자있는 함수호출 불가
- `{{ mylist.0 }}` : 인덱스 접근도 attr처럼 접근

```
people = {'Amy':23, 'Josh':25}
people['Amy'] / {{ people.Amy }}

class Person(object):
    def say_hello(self):
        print('hello')
person.say_hello() / {{ person.say_hello }}

names=['Amy', 'Josh']
names[0] / {{ names.0 }}
```

Variables

4. Template Engines

```
# mytest/views.py
from django.shortcuts import render

class Person():
    def __init__(self, name):
        self.name = name

    def say_hello(self):
        return 'hello'

def test(request):
    people = ['Amy', 'Josh', 'Tobey', 'John']
    person = Person('Amy')
    return render(request, 'mytest/test.html', {'people':people, 'person':person})
```

```
# mytest/template/mytest/test.html
<html>
<head></head>
<body>
    {{person.say_hello}}
    {{person.name}} <br/>

    {{people}} <br/>
    {{ people.1 }}
</body>
</html>
```

Variables

4. Template Engines

```
# mytest/views.py
from django.shortcuts import render
from django.utils import timezone

class Person():
    def __init__(self, name):
        self.name = name
    def say_hello(self):
        return 'hello ~ ' + self.name

def test(request):
    people = ['Amy', 'Josh', 'Tobey', 'Josh']
    person = Person('Amy')
    person_list = []
    now = timezone.now()
    past_dt = timezone.datetime(1972,1,12,10,30)
    criteria_dt = timezone.datetime(2022,3,10,9,30)
    future_dt = timezone.datetime(2057,12,1,0,0)
```

Variables

4. Template Engines

```
value = '''Miracles happen to only those who believe in them.
```

```
    Better the last smile than the first laughter. '''
```

```
value1 = 'Joel is a slug'
```

```
value2 = '<p>Joel is a slug</p>'
```

```
value3 = "https://www.example.org/foo?a=b&c=d"
```

```
value4 = "Check out www.djangoproject.com"
```

```
value5 = "Send questions to foo@example.com"
```

```
value6 = '<b>Joel</b> \n is a slug'
```

```
return render(request, 'mytest/test.html',
               {'people':people, 'person':person,
                'person_list':person_list,
                'datetime_obj':now, 'past_dt':past_dt,
                'criteria_dt':criteria_dt, 'future_dt':future_dt,
                'value':value, 'value1':value1,
                'value2':value2, 'value3':value3,
                'value4':value4, 'value5':value5, 'value6':value6})
```

Variables

4. Template Engines

templates/mytest/names.html

```
{{name}} 님 {{greeting}} ~
```

templates/mytest/test.html

```
<html> <body>
```

```
  {% include "mytest/names.html" with name='홍' greeting='안녕하세요' %}
```

```
  {{person.say_hello}}<br>
```

```
  {{person.name}}<hr>
```

```
  {{people}}<br>
```

```
  {{people.1}}<hr>
```

```
  {% for p in people %}
```

```
    {{forloop.counter}} : {{p}}
```

```
  {% endfor %}
```

```
<hr>
```

```
  {% for p in person_list %}
```

```
    {{p.name}}
```

```
  {% empty %}
```

```
    no name
```

```
  {% endfor %}
```

<hr>

{% lorem %}

{% lorem 3 p %}

{% lorem 2 w random %}

<hr> It is {% now "jS F Y H:i" %}

<hr>

{% verbatim %}

 {{if dying}} still alive.{{/if}}

{% endverbatim %}

<hr>

{% with alpha=1 beta=2 %}

 {{alpha}}

{% endwith %}

<hr>

지금

{{ datetime_obj|date:"D d M Y" }}

{{datetime_obj|date:"Y년m월d일 H시|i분s초"}}

{{ datetime_obj|date:"DATE_FORMAT" }}

{{ datetime_obj|date:"DATETIME_FORMAT" }}

{{ datetime_obj|date:"SHORT_DATE_FORMAT" }}

{{ datetime_obj|date:"SHORT_DATETIME_FORMAT" }}

{{ datetime_obj|time:"TIME_FORMAT" }}

<hr>

```
{{ past_dt|timesince }} <br/>
```

```
{{ past_dt|timesince:criteria_dt }} <br/>
```

```
{{ future_dt|timeuntil }} <br/>
```

```
{{ future_dt|timeuntil:past_dt}} <br/>
```

```
<hr>
```

```
{{person_list|default:"nothing False"}} 값이 없는 경우 default값<br>
```

```
{{person_list|default_if_none:"nothing None"}} 값이 None인 경우 default값
```

```
<hr>
```

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

```
<hr>{{people|length}}
```

```
<hr> {{value|linebreaks}}
```

```
<hr> {{people|random}}
```

```
<hr><!-- html태그 적용. 기본은 html태그도 문자처럼 !-->
```

```
{% autoescape off %}
```

```
    {{value6}} <br>
```

```
{% endautoescape %} {{value6}}
```

```
<hr> {{value6|safe}}
```

```
<hr> {{people|slice:"2"}}
```

```
<hr> {{value6|striptags}}
```

```
<hr> {{value1|truncatechars:9}}
```

```
<hr> {{value2|truncatechars_html:9}}
```

```
<hr> {{value1|truncatewords:2}}
```

```
<hr> {{value2|truncatewords_html:2}}
```

```
<hr> {{value3|urlencode}}
```

```
<hr>
{{value4 | urlize}}
<hr>
{{value5 | urlize}}
</body>
</html>
```


Django Template Tag

4. Template Engines

- Django Templates 용 함수
- 문법 : {% %}
- 빌트인 태그 지원 및 커스텀 태그 추가 가능
- 참조 문서
 - <https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#built-in-tag-reference>

Built-in tag reference

autoescape


Controls the current auto-escaping behavior. This tag takes either **on** or **off** as an argument and that determines whether auto-escaping is in effect inside the block. The block is closed with an **endautoescape** ending tag.

When auto-escaping is in effect, all variable content has HTML escaping applied to it before placing the result into the output (but after any filters have been applied). This is equivalent to manually applying the **escape** filter to each variable.

The only exceptions are variables that are already marked as "safe" from escaping, either by the code that populated the variable, or because it has had the **safe** or **escape** filters applied.

Sample usage:

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



raiseo media donated to the
Django Software Foundation to
support Django development.
Donate today!

Contents

- [Built-in template tags and filters](#)
 - [Built-in tag reference](#)
 - [autoescape](#)
 - [block](#)
 - [comment](#)
 - [csrf_token](#)
 - [cycle](#)
 - [debug](#)
 - [extends](#)
 - [extends3](#)

block 태그

4. Template Engines

- 템플릿 상속에서 사용
- 자식 템플릿이 오버라이딩할 block 영역 정의
- 자식 템플릿은 부모가 정의한 block 만 재정의 가능

```
{% block block-name %}  
    block 내에 내용을 쓰실 수 있습니다.  
{% endblock %}
```

comment 태그

4. Template Engines

- 템플릿 주석 지정
- comment 영역은 서버 단에서 처리하지 않음
- 여러줄 주석 처리

```
{% comment "Optional note" %}
```

```
주석 1
```

```
주석 2
```

```
{% endcomment %}
```

```
{# 한줄 주석 #}
```

csrf_token 태그

4. Template Engines

- Cross Site Request Forgeries를 막기 위해 CSRF Middleware 제공

```
45 MIDDLEWARE = [  
46     'django.middleware.security.SecurityMiddleware',  
47     'django.contrib.sessions.middleware.SessionMiddleware',  
48     'django.middleware.common.CommonMiddleware',  
49     'django.middleware.csrf.CsrfViewMiddleware',  
50     'django.contrib.auth.middleware.AuthenticationMiddleware',  
51     'django.contrib.messages.middleware.MessageMiddleware',  
52     'django.middleware.clickjacking.XFrameOptionsMiddleware',  
53 ]
```

- POST 요청시 CSRF 토큰 체크를 함
- csrf_token 태그로 CSRF 토큰 발급

csrf_token 태그

4. Template Engines

```
<form method="POST" action="">
    {% csrf_token %}
    <input type="text" name="author" />
    <textarea name="message"></textarea>
    <input type="submit" />
</form>
```

extends 태그

4. Template Engines

- 자식 템플릿에서 부모 템플릿 상속
- 소스의 첫번째줄에 위치해야 함
- 자식 템플릿에서는 상속받는 부모 템플릿의 block만 정의할 수 있음

```
{% extends "base.html" %}
```

for 태그

4. Template Engines

- 지정 객체를 순회
- 파이썬의 for문과 동일

```
<ul>
  {% for article in article_list %}
      <li>{{ article.title }}</li>
  {% endfor %}
</ul>
```

반복문안에서 변수 사용

4. Template Engines

- forloop.counter : 1부터 시작하여 1씩 증가 index
- forloop.counter0 : 0부터 시작하여 1씩 증가 index
- forloop.revcounter : 끝인덱스부터 시작, 1씩 감소 index
- forloop.revcounter0 : 끝에서부터 1씩 감소 index
- forloop.first , forloop.last : 첫 실행 , 마지막 실행 여부
- forloop.parentloop : 중첩 loop에서 부모 loop를 지정

```
#views.py
def test(request):
    people = ['Amy', 'Josh', 'Tobey', 'John']
    return render(request, 'mytest/test.html', {'people':people})

#test.html
{% for p in people%}
{{forloop.counter}} : {{ p }} <br/>
{% endfor %}
```


for ... empty 태그

4. Template Engines

- for문내에서 지정 object를 찾을 수 없거나, 비었을 때 empty block이 수행

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

```
#views.py
def test(request):
    person_list = []
    return render(request, 'mytest/test.html', {'person_list':person_list})
```

```
#test.html
{% for p in person_list %}
  {{ p.name }}
{% empty %}
  no name
{% endfor %}
```

for ... empty 태그

4. Template Engines

```
athlete_list = []

## 파이썬 코드
if athlete_list:
    for athlete in athlete_list:
        print(athlete.name)
else:
    print("empty")

## 템플릿 코드
{% for athlete in athlete_list %}
    {{ athlete.name }}
{% empty %}
    empty
{% endfor %}

## 템플릿 동일 코드
{% if athlete_list %}
    {% for athlete in athlete_list %}
        {{ athlete.name }}
    {% else %}
        empty
    {% endif %}
{% endif %}
```

if 태그

4. Template Engines

● 파이썬의 if문과 동일

```
{% 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 %}
```

include 태그

4. Template Engines

- 다른 템플릿을 로딩/렌더링을 수행하며, 현재 context가 그대로 전달
- include 시에 keyword 인자를 지정하여, 추가 context 지정 가능

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

lorem 태그

4. Template Engines

- 랜덤 채우기 텍스트를 생성

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

- count : 생성할 단락/단어의 수 (디폴트 : 1)
- method : 단어 - w, HTML 단락- p, Plain Text 단락- b 지정 (디폴트 : b)
- random : random 지정/미지정

```
{% lorem %} : 보통의 채우기 텍스트 출력  
{% lorem 3 p %} : HTML 단락 3개 출력  
{% lorem 2 w random %} : 랜덤 단어 2개 출력
```

- 현재 날짜/시간 출력
 - It is {% now "jS F Y H:I" %}
- 출력 포맷 참조
 - <https://docs.djangoproject.com/en/3.2/ref/templates/builtins/#std:templatefilter-date>

- URL Reverse를 수행한 URL문자열을 출력
- 인자처리는 `django.shortcuts.resolve_url` 함수와 유사하게 처리하나, `get_absolute_url` 처리는 하지 않음

```
{% url "some-url-name-1" %}  
{% url "some-url-name-2" arg arg2 %}  
{% url "some-url-name-2" arg arg2 as the_url %}
```



5. Template Filter



Template Filter

5. Template Filter

- **템플릿 변수값 변환을 위한 함수, | (파이프 기호) 사용**
 - `{{ var|filter1 }}` : var 은 filter1의 첫번째 인자로 사용
 - `{{ var|filter2:var2 }}` : var은 filter2의 첫번째 인자, var2은 두번째 인자
- **다중 필터 함수 연결 가능**
 - `{{ var|filter3:var2|filter4 }}` : var은 filter3의 첫번째 인자, var2는 두번째 인자이며 filter3에서 처리된 결과는 filter4의 첫번째 인자
- **빌트인 Filter 지원, 장고앱별로 커스텀 Filter 추가 가능**

date/time filter

5. Template Filter

- 지정 포맷으로 출력
- date/time 참조 문서
 - <https://docs.djangoproject.com/en/5.1/ref/templates/builtins/#date>
 - <https://docs.djangoproject.com/en/5.1/ref/templates/builtins/#time>
- Django 기본값
 - [django.conf.global_settings.py](#)

```
332 # Default formatting for date objects. See all available format strings here:
333 # https://docs.djangoproject.com/en/dev/ref/templates/builtins/#date
334 DATE_FORMAT = 'N j, Y'
335
336 # Default formatting for datetime objects. See all available format strings here:
337 # https://docs.djangoproject.com/en/dev/ref/templates/builtins/#date
338 DATETIME_FORMAT = 'N j, Y, P'
339
340 # Default formatting for time objects. See all available format strings here:
341 # https://docs.djangoproject.com/en/dev/ref/templates/builtins/#date
342 TIME_FORMAT = 'P'
```

date/time filter

5. Template Filter

```
{{ datetime_obj|date:"D d M Y" }}  
=> 'Wed 09 Jan 2008'  
{{ datetime_obj|date:"DATE_FORMAT" }}  
=> 디폴트 'N j, Y' (e.g. Feb. 4, 2003)  
{{ datetime_obj|date:"DATETIME_FORMAT" }}  
=> 디폴트 'N j, Y, P' (e.g. Feb. 4, 2003, 4 p.m.)  
{{ datetime_obj|date:"SHORT_DATE_FORMAT" }}  
=> 디폴트 'm/d/Y' (e.g. 12/31/2003)  
{{ datetime_obj|date:"SHORT_DATETIME_FORMAT" }}  
=> 디폴트 'm/d/Y P' (e.g. 12/31/2003 4 p.m.)  
{{ datetime_obj|time:"TIME_FORMAT" }}  
=> 디폴트 'P' (e.g. 4 p.m.)
```

```
from django.utils import timezone
```

```
timezone.now()
```

```
datetime.datetime(2018, 6, 26, 8, 20, 26, 567729, tzinfo=<UTC>)
```

```
timezone.now().strftime("%Y-%m-%d %H:%M:%S")
```

```
'2018-06-26 08:21:04'
```

timesince/timeuntil filter

5. Template Filter

```
{{ past_dt|timesince }} => 현재시각 기준 (now - past_dt)  
{{ past_dt|timesince:criteria_dt }} => 기준시각 기준 (criteria_dt - past_dt)  
{{ future_dt|timeuntil }} => 현재시각 기준 (future_dt - now)  
{{ future_dt|timeuntil:past_dt }} => 기준시각 기준 (future_dt - past_dt)
```

- timesince

- 과거부터 현재까지 시간

- timeuntil

- 미래 시점까지 남아있는 시간

date/time filter

5. Template Filter

```
# views.py
def test(request):
    past_dt = timezone.datetime(1971,8,22,0,0)
    criteria_dt = timezone.datetime(2001,3,19,0,0)
    future_dt = timezone.datetime(2037,1,1,0,0)

    return render(request, 'mytest/test.html', 'datetime_obj':now,
                  'past_dt':past_dt,
                  'criteria_dt':criteria_dt,
                  'future_dt':future_dt))
```

```
#test.html
{{ datetime_obj|date:"D d M Y" }} <br/>
{{ datetime_obj|date:"DATE_FORMAT" }} <br/>
{{ datetime_obj|date:"DATETIME_FORMAT" }} <br/>
{{ datetime_obj|date:"SHORT_DATE_FORMAT" }} <br/>
{{ datetime_obj|date:"SHORT_DATETIME_FORMAT" }} <br/>
{{ datetime_obj|time:"TIME_FORMAT" }} <br/>

{{ past_dt|timesince }} <br/>
{{ past_dt|timesince:criteria_dt }} <br/>
{{ future_dt|timeuntil }} <br/>
{{ future_dt|timeuntil:past_dt}} <br/>
```

timezone

5. Template Filter

● timezone 지정 객체

```
from django.utils import timezone
timezone.now()                                #timezone 정보
datetime.datetime(2018, 6, 26, 8, 20, 26, 567729, tzinfo=<UTC>)

import datetime
datetime.datetime.now()                       #timezone 정보없음
datetime.datetime(2018, 6, 26, 17, 39, 50, 233286)

datetime.datetime(2018,6,26,0,0) # UTC
datetime.datetime(2018,6,26,9,0) # Asia/Seoul
datetime.datetime(2018,6,26,0,0) == datetime.datetime(2018,6,26,9,0)
False
```

● django.utils의 timezone 사용 권장

Template Filter

5. Template Filter

● join

- 순회가능한 객체를 지정 문자열로 연결
- 파이썬의 `str.join(list)`과 동일
 - `{{ people|join:" // " }}` => `['a', 'b', 'c']` 일 경우, `"a // b // c"`를 출력

● length

- value의 길이를 출력
- 파이썬의 `len(value)`과 동일
- Undefined 변수일 경우 0을 출력
 - `{{ people|length }}` => `['a', 'b', 'c', 'd']` 일 경우 4 를 출력

Template Filter

5. Template Filter

- **truncatechars**

- 문자열을 지정 글자 갯수까지만 보여주고 추가

- **truncatechars_html**

- HTML 태그를 보호하면서 문자열을 지정 글자 갯수까지만 보여주고 추가

```
value1 = "Joel is a slug"  
value2 = "<p>Joel is a slug</p>"
```

```
{{ value1|truncatechars:9 }} => "Joel i..."  
{{ value2|truncatechars_html:9 }} => "<p>Joel i...</p>"
```


Template Filter

5. Template Filter

- **truncatewords**

- 문자열을 지정 단어 갯수까지 보여주고 ... 추가

- **truncatewords_html**

- HTML 태그를 보호하면서 문자열을 지정 단어 갯수까지 보여주고 ... 추가

```
value1 = "Joel is a slug"
value2 = "<p>Joel is a slug</p>"

{{ value1|truncatewords:2 }} => "Joel is ..."
{{ value2|truncatewords_html:2 }} => "<p>Joel is ...</p>"
```

- **urlencode**

- 지정값을 urlencode 처리

```
value = "https://www.example.org/foo?a=b&c=d"
{{ value|urlencode }} => "https%3A//www.example.org/foo%3Fa%3Db%26c%3Dd"
```