■ 뷰

● 뷰를 구현하는 2가지 방법

- Function views / Include App urls
  app/views.py 내의 파이썬 함수로 표현
  함수의 인자로 request를 받도록 작성

- Class-based views / Include App urls
  app/views.py 내의 파이썬 클래스로 표현
  generic 모듈 내의 View 상속

■ Function view

- app/views.py 내의 파이썬 함수로 표현
- 함수의 인자로 request를 받도록 작성
  - 코드 구성 : HttpResponse 사용

```
from django.http import HttpResponse

def index(request):
    return HttpResponse('출력 문자열')
```

  - 코드 구성 : render 사용

```
from django.shortcuts import render

def index(request):
    return render(request, '템플릿/HTML', 데이터객체)
```

■ Function view

● 뷰의 함수만 단독으로 사용 할 수 없고 URL 정보를 담당하는

Project/urls.py 또는 app/urls.py 를 통해서 사용

- Project/urls.py 이용

```
from django.urls import path
from [app] import views

urlpatterns = [
    path('접속주소/', views.함수명, name='이름'),
]
```

- app/urls.py 이용

```
from django.urls import path
from . import views

urlpatterns = [
    path('접속주소/', views.함수명, name='이름'),
]
```

```
# Project/urls.py

from django.urls import path, include

urlpatterns = [
    path('app/', include('app.urls')),
]
```

■ Class-based view

- app/views.py 내의 파이썬 클래스로 표현
- generic 모듈 내의 View 상속 (ListView, DetailView 등)

  - Project/urls.py 이용

```
class ShowView(generic.ListView):
    template_name = 'firstapp/show.html'
    context_object_name = 'list'

    def get_queryset(self):
        return Curriculum.objects.all()
```

```
# Project/urls.py

from django.urls import path

urlpatterns = [
    path('show2/', views.ShowView.as_view(), name='showview_class'),
]
```

■ Class-based view

● app/views.py 내의 파이썬 클래스로 표현

● generic 모듈 내의 View 상속 (ListView, DetailView 등)

- app/urls.py 이용

```
class ShowView(generic.ListView):
    template_name = 'firstapp/show.html'
    context_object_name = 'list'

    def get_queryset(self):
        return Curriculum.objects.all()
```

```
# Project/urls.py

from django.urls import path, include

urlpatterns = [
    path('app/', views.include('app.urls')),
]
```

```
# app/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('show2/', views.ShowView.as_view(), name='showview_class2'),
]
```

■ Generic View

- 공통적으로 사용하는 로직을 미리 작성해놓고 클래스로 제공
- 개발하려는 로직에 맞는 뷰를 선택만 해주면 빠른 개발 가능

제네릭 뷰 구분	뷰 이름	기능
Base View	View	최상위 제네릭 뷰 (기본 뷰)
	TemplateView	템플릿을 지정하여 렌더링
	RedirectView	URL을 지정하여 이동
Generic Display View	DetailView	단일 객체 출력
	ListView	다중 객체 출력
Generic Edit View	FormView	폼 출력
	CreateView	객체 생성 폼 출력
	UpdateView	객체 수정 폼 출력
	DeleteView	객체 삭제 폼 출력
Generic Date View	ArchiveIndexView	날짜 필드를 기준으로 다중 객체 출력
	YearArchiveView	연도에 해당하는 다중 객체 출력
	MonthArchiveView	연/월에 해당하는 다중 객체 출력
	WeekArchiveView	연/주에 해당하는 다중 객체 출력
	DayArchiveView	연/월/일에 해당하는 다중 객체 출력
	TodayArchiveView	현재 날짜에 해당하는 다중 객체 출력
	DateDetailView	날짜에 해당하는 단일 객체 출력

■ Generic View

● 기본 템플릿 파일명

뷰 이름	기본 템플릿 파일명
View	없음
TemplateView	없음
RedirectView	없음
DetailView	**모델명_detail.html**
ListView	**모델명_list.html**
FormView	없음
CreateView	**모델명_form.html**
UpdateView	**모델명_form.html**
DeleteView	**모델명_confirm_delete.html**

- 기본 템플릿이 없는 뷰는 template_name 속성에 직접 값을 입력하거나 get_template_names() 메소드를 오버라이딩하여 템플릿 지정

■ Generic View – Base View

● View

- 모든 클래스형 뷰의 기본이 되는 최상위 뷰
- 원하는 로직의 뷰가 없는 경우 직접 상속받은 후 뷰 작성

```
from django.views import View

class BaseView(View):
    def get(self, request, *args, **kwargs):
        return HttpResponse('BaseView')
```

● TemplateView

- 단순하게 화면에 보여줄 템플릿을 렌더링해주는 뷰

```
from django.views.generic import TemplateView

class TemplateView(TemplateView):
    template_name = 'firstapp/temp.html'
```

■ Generic View – Base View

● RedirectView

- 주어진 URL로 리다이렉트 시켜주는 뷰
- URL 대신 패턴명 지정 가능 ex) 'mysite:main'

```
# views.py

from django.views.generic import RedirectView

class RedirectView(RedirectView):
    # url = '/first/main/'
    pattern_name = 'main'
```

```
# urls.py

urlpatterns = [
    path('main/', views.main, name='main'),
    path('redirectview/', views.RedirectView.as_view(), name='redirectview'),
```

■ Generic View – Display View

● DetailView

- ListView와 함께 가장 많이 사용되는 뷰
- 모델(테이블)에서 레코드 하나를 조회한 후 object 컨텍스트 변수에 담음
- context_object_name과 template_name을 별도 지정하여 컨텍스트 변수명 및 template 파일명 지정 가능

```
# views.py

class DetailView(DetailView):
    model = Curriculum
```

```
# views.py

class DetailView(DetailView):
    model = Curriculum
    template_name = 'firstapp/detail.html'
    context_object_name = 'curriculum'
```

```
# curriculum_detail.html

{{ object.id }} <br>
{{ object.name }}
```

```
# detail.html

{{ curriculum.id }} <br>
{{ curriculum.name }}
```

```
# urls.py
path('detailview/<int:pk>', views.DetailView.as_view()),
```

■ Generic View – Display View

● ListView

- DetailView와 함께 가장 많이 사용되는 뷰
- 레코드 여러개를 조회한 후 object_list 컨텍스트 변수에 담음
- context_object_name과 template_name을 별도 지정하여 컨텍스트 변수명 및 template 파일명 지정 가능

```
# views.py

class ListView(ListView):
    model = Curriculum
```

```
# views.py

class ListView(ListView):
    model = Curriculum
    template_name = 'firstapp/list.html'
    context_object_name = 'curriculum'
```

```
# curriculum_list.html

{% for object in object_list %}
    {{ object.id }} / {{ object.name }}
    <br>
{% endfor %}
```

```
# list.html

{% for cur in curriculum %}
    {{ cur.id }} / {{ cur.name }}
    <br>
{% endfor %}
```

■ Generic View – Edit View

● FormView

- 폼을 보여주기 위한 뷰
- form_class, template_name, success_url(처리 후 이동 주소) 속성 사용
- get / post 메소드를 이용하여 요청 구분 가능

```
# forms.py
from django import forms
class SearchForm(forms.Form):
    search_word = forms.CharField(label='Search Word')
```

```
# views.py
class SearchFormView(FormView):
    form_class = SearchForm
    template_name = 'firstapp/search.html'

    def form_valid(self, form):
        search_word = '%s' % self.request.POST['search_word']
        list = Curriculum.objects.filter(name__contains=search_word)
        context = {}
        context['form'] = form
        context['search_word'] = search_word
        context['object_list'] = list
        return render(self.request, self.template_name, context)
```

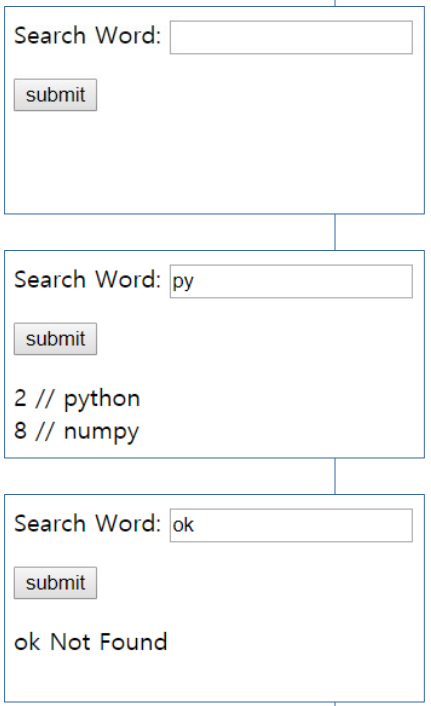
■ Generic View – Edit View

● FormView

- 폼을 보여주기 위한 뷰
- form_class, template_name, success_url(처리 후 이동 주소) 속성 사용
- get / post 메소드를 이용하여 요청 구분 가능

```
# search.html
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="submit">
</form>

{% if object_list %}
    {% for object in object_list %}
        {{ object.id }} // {{ object.name }}
        <br>
    {% endfor %}
{% elif search_word %}
    {{ search_word }} Not Found
{% endif %}
```



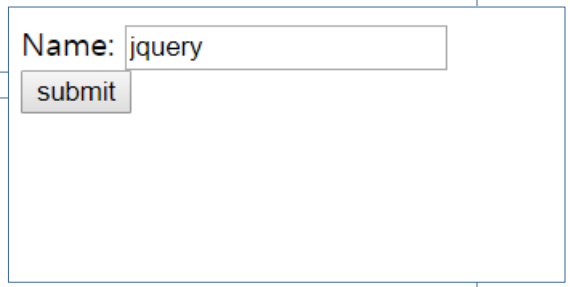
■ Generic View – Edit View

● CreateView

- 새로운 레코드를 저장하기 위한 뷰
- FormView 기능 포함

```
# views.py
class CreateView(CreateView):
    model = Curriculum
    fields = ['name']
    success_url = reverse_lazy('show')
```

```
# curriculum_form.html
<form method="post">
    {% csrf_token %}
    {{ form.name.label_tag }} {{ form.name }}
    <br>
    <input type="submit" value="submit">
</form>
```



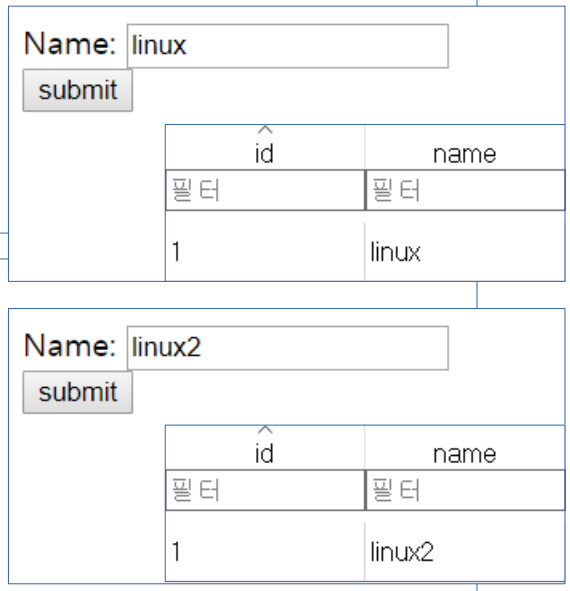
■ Generic View – Edit View

● UpdateView

- 존재하는 레코드를 수정하기 위한 뷰
- CreateView와 유사하고, FormView 기능 포함

```
# views.py
class UpdateView(UpdateView):
    model = Curriculum
    fields = ['name']
    success_url = reverse_lazy('show')
```

```
# curriculum_form.html
<form method="post">
    {% csrf_token %}
    {{ form.name.label_tag }} {{ form.name }}
    <br>
    <input type="submit" value="submit">
</form>
```




```
# urls.py
path('updateview/<int:pk>', views.UpdateView.as_view()),
```

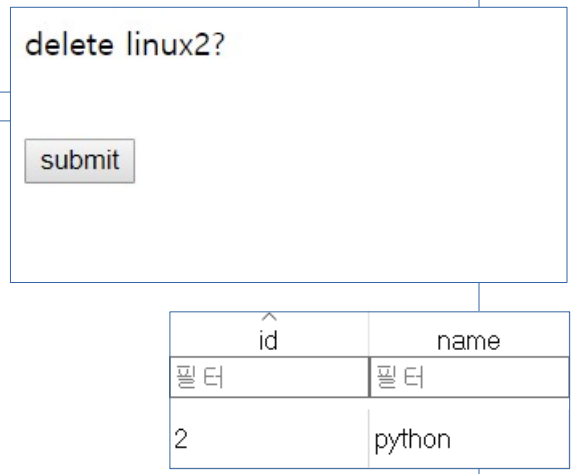
■ Generic View – Edit View

● DeleteView

- 존재하는 레코드를 삭제하기 위한 뷰
- UpdateView와 유사하지만 폼의 모습이 다름

```
# views.py
class DeleteView(DeleteView):
    model = Curriculum
    success_url = reverse_lazy('show')
```

```
# curriculum_confirm_delete.html
<form method="post">
    {% csrf_token %}
    <p>delete {{ object }}?</p>
    <br>
    <input type="submit" value="submit">
</form>
```




```
# urls.py
path('deleteview/<int:pk>', views.DeleteView.as_view()),
```

■ Generic View – 속성

● model / queryset

- 기본 뷰 (View, TemplateView, RedirectView)를 제외한 제네릭 뷰에서 사용
- 뷰가 출력 또는 입력받을 데이터로 사용

```
class TestView(ListView):
    model = Curriculum                  # model 사용
    queryset = Curriculum.objects.all()    # queryset 사용
```

● template_name / get_template_names()

- 모든 제네릭 뷰에서 사용
- 템플릿 파일명을 문자열로 지정하여 뷰가 보여줄 모습으로 사용

```
class TestView(TemplateView):
    template_name = '/first/templateview'     # template_name 사용
    def get_template_names(self):             # get_template_names() 사용
        return 'firstapp/temp.html'
```

■ Generic View - 속성

● context_object_name

- 기본 뷰 (View, TemplateView, RedirectView)를 제외한 제네릭 뷰에서 사용
- 템플릿 파일에서 사용할 컨텍스트 변수명 지정

```
# views.py

class DetailView(DetailView):
    model = Curriculum
    template_name = 'firstapp/detail.html'
    context_object_name = 'curriculum'
```

```
# detail.html

{{ curriculum.id }} <br>
{{ curriculum.name }}
```

■ Generic View – 속성

● pagenate_by

- 목록을 출력하는 ListView와 같은 뷰에서 사용
- 페이지당 출력될 항목 개수 지정

```
class ListView(ListView):
    model = Curriculum
    template_name = 'firstapp/list.html'
    context_object_name = 'curriculum'
    paginate_by = 5
```

```
2 / python
3 / html/css/js
4 / django
5 / block chain
7 / sqlite
```

```
class ListView(ListView):
    model = Curriculum
    template_name = 'firstapp/list.html'
    context_object_name = 'curriculum'
    paginate_by = 3
```

```
2 / python
3 / html/css/js
4 / django
```

■ Generic View – 속성

● form_class

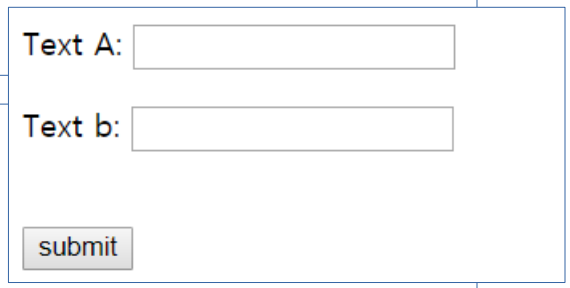
– FormView / CreateView / UpdateView에서 사용

– 폼을 만드는데 사용할 클래스 지정

```
# forms.py
class TestForm(forms.Form):
    text_a = forms.CharField(label='Text A')
    text_b = forms.CharField()
```

```
# views.py
class TestView(FormView):
    form_class = TestForm
    template_name = 'firstapp/test.html'
```

```
# test.html
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <br>
    <input type="submit" value="submit">
</form>
```



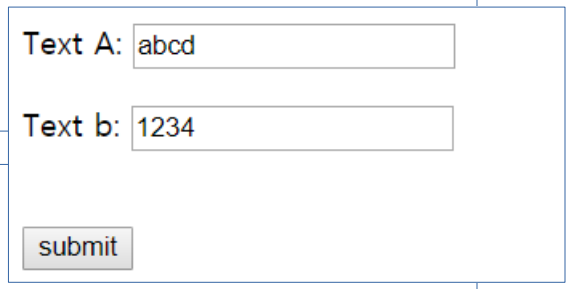
■ Generic View – 속성

● initial

- FormView / CreateView / UpdateView에서 사용
- 폼을 사용할 초기 데이터를 딕셔너리 형식으로 지정

```
# views.py
class TestView(FormView):
    form_class = TestForm
    template_name = 'firstapp/test.html'
    initial = {'text_a': 'abcd', 'text_b': '1234'}
```

```
# test.html
<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <br>
    <input type="submit" value="submit">
</form>
```



■ Generic View - 속성

- fields
  - CreateView / UpdateView에서 사용
  - 폼으로 사용할 필드 지정 (= ModelForm의 Meta.fields 속성)
- success_url
  - FormView / CreateView / UpdateView / DeleteView에서 사용
  - 폼에 대한 처리가 이루어진 이후 리다이렉트될 URL 지정

■ Generic View - 메소드

● get_queryset()

- QuerySet 객체 또는 객체 리스트 반환
- queryset 속성이 지정되지 않은 경우 모델의 매니저 클래스 all() 메소드를 호출하여 QuerySet 객체 반환

```
class ListView(ListView):
    model = Curriculum
    template_name = 'firstapp/list.html'
    context_object_name = 'curriculum'
```

```
2 / python
3 / html/css/js
4 / django
5 / block chain
7 / sqlite
8 / numpy
9 / jquery
```

```
class ListView(ListView):
    model = Curriculum
    template_name = 'firstapp/list.html'
    context_object_name = 'curriculum'

    def get_queryset(self):
        return self.model.objects.filter(name__contains='n')
```

```
2 / python
4 / django
5 / block chain
8 / numpy
```

■ Generic View - 메소드

● get_context_data()

- TemplateView를 포함하여 모든 제네릭 뷰에서 사용
- 템플릿 파일에서 사용할 컨텍스트 데이터 반환

```
# temp.html
<h1>TemplateView</h1>
{{ data }}
```

```
class TemplateView(TemplateView):
    template_name = 'firstapp/temp.html'
```

```
class TemplateView(TemplateView):
    template_name = 'firstapp/temp.html'

    def get_context_data(self, **kwargs):
        context = {}
        context['data'] = 'Context Data'
        return context
```

TemplateView

TemplateView

Context Data

■ Templates

- 실제 View의 역할
- View로부터 전달된 데이터 사용 가능
- app/templates/app/html파일 형태로 사용
- 장고 템플릿에서만 사용하는 특별한 태그 및 문법 제공

  - 변수, 태그, 필터, 코멘트, HTML 확장 등...

- https://docs.djangoproject.com/ko/2.1/ref/templates/builtins

■ Templates

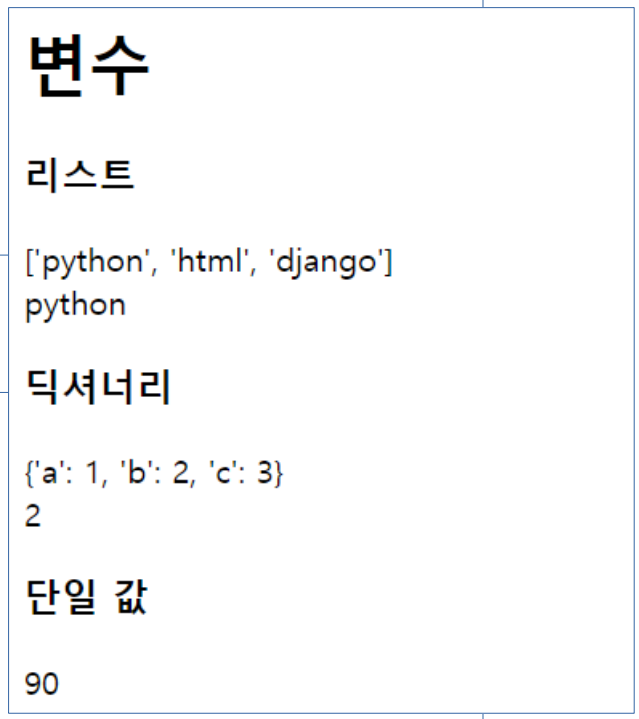
● 변수

- app/views.py

```
def template(request):
    words = ['python', 'html', 'django']
    dic = {'a': 1, 'b': 2, 'c': 3}
    score = 90
    return render(
        request,
        'firstapp/template.html',
        {'words': words, 'dic': dic, 'score': score}
    )
```

- app/template/app/template.html

```
<h1>변수</h1>
<h3>리스트</h3>
{{words}}<br>
{{words.0}}<br>
<h3>딕셔너리</h3>
{{dic}}<br>
{{dic.b}}
<h3>단일 값</h3>
{{score}}
```



■ Templates

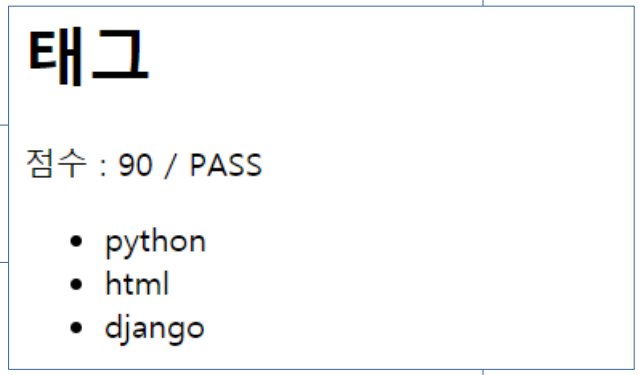
● 태그

- app/views.py

```
def template(request):
    words = ['python', 'html', 'django']
    dic = {'a': 1, 'b': 2, 'c': 3}
    score = 90
    return render(
        request,
        'firstapp/template.html',
        {'words': words, 'dic': dic, 'score': score}
    )
```

- app/template/app/template.html

```
<h1>태그</h1>
{% if score > 60 %} 점수 : {{score}} / PASS
{% else %} 점수 : {{score}} / FAIL
{% endif %}
<ul>
{% for word in words %}
    <li>{{ word }}</li>
{% endfor %}
</ul>
```



■ Templates

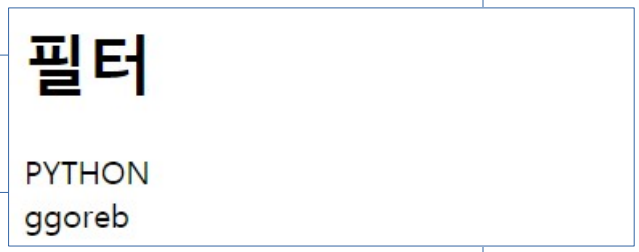
● 필터

- app/views.py

```
def template(request):
    words = ['python', 'html', 'django']
    dic = {'a': 1, 'b': 2, 'c': 3}
    score = 90
    return render(
        request,
        'firstapp/template.html',
        {'words': words, 'dic': dic, 'score': score}
    )
```

- app/template/app/template.html

```
<h1>필터</h1>
{{words.0|upper}}<br>
{{'GGOREB'|lower}}<br>
```



■ Templates

● 코멘트

- app/template/app/template.html

```
<h1>코멘트</h1>
{# 한줄 주석 #}
{% comment %}
    주석 처리된 코드
    html에서 보이지 않는 코드
{% endcomment %}
```

코멘트

■ Templates

● HTML 확장

- app/template/app/extends1.html

```
{% extends 'firstapp/base.html' %}
{% block content %}
<form>
    아이디 : <input type="text" ><br>
    비밀번호 : <input type="text" >
</form>
{% endblock %}
```

- app/template/app/extends2.html

```
{% extends 'firstapp/base.html' %}
{% block content %}
<table border="1">
    <tr><td>번호</td><td>제목</td></tr>
    <tr><td>2</td><td>B</td></tr>
    <tr><td>1</td><td>A</td></tr>
</table>
{% endblock %}
```

- base.html

```
<h1>Base HTML</h1>
<div>
    <h1>Header</h1>
</div>
<div>
    <h1>Content</h1>
    {% block content %}
    {% endblock %}
</div>
<div>
    <h1>Footer</h1>
</div>
```

■ Templates

● HTML 확장

– extends1

– extends2