04장 Django의 핵심 기능

4.1 Admin 사이트 꾸미기

- 장고의 Admin 사이트는 데이터베이스에 들어있는 데이터를 쉽게 관리할 수 있도록 데이터의 생성, 조회, 변경, 삭제 등의 기능을 제공한다.

- Admin 사이트 접속
  - http://127.0.0.1:8000/admin/

4.1.1 데이터 입력 및 수정

- Admin 화면에서 테이블에 데이터를 CRUD할 수 있다.

4.1.2 필드 순서 변경하기

- 테이블의 데이터를 변경하는 것이 아니라, 테이블을 보여주는 UI 양식을 변경하려면 polls/admin.py 파일을 변경하면 된다.

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class QuestionAdmin(admin.ModelAdmin):
07      fields = ['pub_date', 'question_text']  # 필드 순서 변경
08
09  admin.site.register(Question, QuestionAdmin)
10  admin.site.register(Choice)
```

4.1.3 각 필드를 분리해서 보여주기

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class QuestionAdmin(admin.ModelAdmin):
07      fieldsets = [
08          ('Question Statement', {'fields': ['question_text']}),
09          ('Date Information', {'fields': ['pub_date']}),
10      ]
11
12  admin.site.register(Question, QuestionAdmin)
13  admin.site.register(Choice)
```

4.1.4 필드 접기

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class QuestionAdmin(admin.ModelAdmin):
07      fieldsets = [
08          ('Question Statement', {'fields': ['question_text']}),
09          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
10      ]
11
12  admin.site.register(Question, QuestionAdmin)
13  admin.site.register(Choice)
```

4.1.5 외래키 관련 화면

4.1.6 Question 및 Choice를 한 화면에서 변경하기

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class ChoiceInline(admin.StackedInline):
07      model = Choice
08      extra = 2
09
10  class QuestionAdmin(admin.ModelAdmin):
11      fieldsets = [
12          (None, {'fields': ['question_text']}),
13          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14      ]
15      inlines = [ChoiceInline]  # Choice 모델 클래스 같이 보기
16
17  admin.site.register(Question, QuestionAdmin)
18  admin.site.register(Choice)
```

4.1.7 테이블 형식으로 보여주기

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class ChoiceInline(admin.TabularInline):
07      model = Choice
08      extra = 2
09
10  class QuestionAdmin(admin.ModelAdmin):
11      fieldsets = [
12          (None, {'fields': ['question_text']}),
13          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14      ]
15      inlines = [ChoiceInline]  # Choice 모델 클래스 같이 보기
16
17  admin.site.register(Question, QuestionAdmin)
18  admin.site.register(Choice)
```

4.1.8 레코드 리스트 컬럼 지정하기

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class ChoiceInline(admin.TabularInline):
07      model = Choice
08      extra = 2
09
10  class QuestionAdmin(admin.ModelAdmin):
11      fieldsets = [
12          (None, {'fields': ['question_text']}),
13          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14      ]
15      inlines = [ChoiceInline]  # Choice 모델 클래스 같이 보기
16      list_display = ('question_text', 'pub_date')    # 레코드 리스트 컬럼 지정
17
18  admin.site.register(Question, QuestionAdmin)
19  admin.site.register(Choice)
```

4.1.9 list_filter 필터

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
```

```
05
06  class ChoiceInline(admin.TabularInline):
07      model = Choice
08      extra = 2
09
10  class QuestionAdmin(admin.ModelAdmin):
11      fieldsets = [
12          (None, {'fields': ['question_text']}),
13          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14      ]
15      inlines = [ChoiceInline]  # Choice 모델 클래스 같이 보기
16      list_display = ('question_text', 'pub_date')    # 레코드 리스트 컬럼 지정
17      list_filter = ['pub_date']      # 필터 사이드 바 추가
18
19  admin.site.register(Question, QuestionAdmin)
20  admin.site.register(Choice)
```

4.1.10 search_fields

[ch4Lab/polls/admin.py]

```
01  from django.contrib import admin
02
03  # Register your models here.
04  from polls.models import Question, Choice
05
06  class ChoiceInline(admin.TabularInline):
07      model = Choice
08      extra = 2
09
10  class QuestionAdmin(admin.ModelAdmin):
11      fieldsets = [
12          (None, {'fields': ['question_text']}),
13          ('Date Information', {'fields': ['pub_date'], 'classes': ['collapse']}),
14      ]
15      inlines = [ChoiceInline]  # Choice 모델 클래스 같이 보기
16      list_display = ('question_text', 'pub_date')    # 레코드 리스트 컬럼 지정
17      list_filter = ['pub_date']          # 필터 사이드 바 추가
18      search_fields = ['question_text']   # 검색 박스 추가
19
20  admin.site.register(Question, QuestionAdmin)
21  admin.site.register(Choice)
```

4.1.11 polls/admin.py 변경 내역 정리

4.1.12 Admin 사이트 템플릿 수정

```
D:\dev\workspace\django1\ch4Lab1>mkdir templates

D:\dev\workspace\django1\ch4Lab1>mkdir templates\admin

D:\dev\workspace\django1\ch4Lab1>python -c "import django; print(django.__path__)"
['D:\\prod\\Anaconda3\\lib\\site-packages\\django']

D:\dev\workspace\django1\ch4Lab1>copy D:\prod\Anaconda3\Lib\site-
packages\django\contrib\admin\templates\admin\base_site.html templates\admin\
```

```
        1개 파일이 복사되었습니다.

D:\dev\workspace\django1\ch4Lab1\templates\admin>type base_site.html ¦ more /P > imsi.html
D:\dev\workspace\django1\ch4Lab1\templates\admin>move imsi.html base_site.html
D:\dev\workspace\django1\ch4Lab1\templates\admin\base_site.html을(를) 덮어쓰시겠습니까? (Yes/No/All): Yes
es/No/All): Yes
        1개 파일을 이동했습니다.
```

4.2 장고 파이썬 쉘로 데이터 조작하기

```
D:\dev\workspace\django1\ch4Lab1>python manage.py shell
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

# Create  - 데이터 생성/입력
In [1]: from polls.models import Question, Choice

In [2]: from django.utils import timezone

In [3]: q = Question(question_text="What's new?", pub_date=timezone.now())

In [4]: q.save()


# Read  - 데이터 조회
In [5]: Question.objects.all()
Out[5]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>, <Question: What's new?>]>

In [7]: Question.objects.filter(question_text__startswith='What')
Out[7]: <QuerySet [<Question: What's new?>]>

In [8]: one_entry = Question.objects.get(pk=1)

In [9]: print(one_entry)
오늘의 맛집?

In [10]: Question.objects.all()[:5]
Out[10]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>, <Question: What's new?>]>

In [11]: Question.objects.all()[1:2]
Out[11]: <QuerySet [<Question: 당신의 취미는?>]>

In [12]: Question.objects.all()[0:3]
Out[12]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>, <Question: What's new?>]>

In [14]: Question.objects.all()[:4:2]
Out[14]: [<Question: 오늘의 맛집?>, <Question: What's new?>]

In [18]: Question.objects.all()[:4:2]
Out[18]: [<Question: 오늘의 맛집?>, <Question: What's new?>]


# Update  - 데이터 수정
In [19]: print(q)
What's new?

In [21]: q.question_text = 'What is your favorite hobby?'

In [22]: q.save()
```

```
# Delete - 데이터 삭제
In [23]: Question.objects.filter(question_text__startswith='What').delete(
    ...: )
Out[23]: (1, {'polls.Question': 1}

In [24]: exit()
```

[실습] polls 애플리케이션의 데이터 실습

■ 장고 파이썬 쉘 - 실습 1

```
D:\dev\workspace\django1\ch4Lab1>python manage.py shell
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

# 우리가 정의한 모델을 사용하기 위해 임포트 한다.
In [1]: from polls.models import Question, Choice

# 현재 각 테이블에 들어있는 레코드를 확인한다.
In [2]: Question.objects.all()
Out[2]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>]>

In [3]: Choice.objects.all()
Out[3]: <QuerySet [<Choice: 국밥>, <Choice: 카레2>, <Choice: 쌀국수>, <Choice: 축구>, <Choice: 음악듣기>]>

# 추가로 레코드 하나를 생성한다.
# 날짜를 입력하기 위해 timezone 모듈을 임포트한다.
# settings.py 파일에 USE_TZ=True로 설정된 경우이다.
# datetime.datetime.now()보다는 timezone.now() 사용을 추천한다.
In [4]: from django.utils import timezone

In [5]: q = Question(question_text="What's up?", pub_date=timezone.now())

# 데이터베이스에 저장을 위한 save() 함수를 호출한다.
In [6]: q.save()

# 속성에 접근할 때는 파이썬 문법 그래도 '.'을 사용한다.
In [7]: q.id
Out[7]: 4

In [8]: q.question_text
Out[8]: "What's up?"

In [9]: q.pub_date
Out[9]: datetime.datetime(2021, 12, 9, 2, 59, 57, 581930, tzinfo=<UTC>)

# 기존의 속성값을 변경하고 데이터베이스에 저장한다.
In [10]: q.question_text = "What's new?"

In [11]: q.save()

# 테이블의 모든 레코드를 조회한다.
In [12]: Question.objects.all()
Out[12]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>, <Question: What's new?>]>

# 파이썬 쉘을 빠져 나오기 위해서는 exit() 또는 Ctrl-Z를 입력한다.
In [13]: exit()
```

■ 장고 파이썬 쉘 - 실습 2

```
D:\dev\workspace\django1\ch4Lab1>python manage.py shell
Python 3.8.5 (default, Sep  3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.19.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: from polls.models import Question, Choice

# 조건 표현에는 filter() 함수 및 키워드 인자를 사용한다.
# startswith와 같은 연산자를 붙일 때는 __(밑줄 2개)를 사용한다.
In [2]: Question.objects.filter(id=1)
Out[2]: <QuerySet [<Question: 오늘의 맛집?>]>

In [3]: Question.objects.filter(question_text__startswith='What')
Out[3]: <QuerySet [<Question: What's new?>]>

# 올해 생성된 질문을 조회해 본다.
In [4]: from django.utils import timezone

In [6]: current_year = timezone.now().year

In [7]: Question.objects.filter(pub_date__year=current_year)
Out[7]: <QuerySet [<Question: 오늘의 맛집?>, <Question: 당신의 취미는?>, <Question: What's new?>]>

# id 값을 잘못 지정하면 익셉션이 발생한다.
In [8]: Question.objects.get(id=5)
---------------------------------------------------------------------------DoesNotExist
Traceback (most recent call last)<ipython-input-8-68e0132266e9> in <module>
----> 1 Question.objects.get(id=5)
...(생략)...
DoesNotExist: Question matching query does not exist.

# 지금부터는 Choice 모델에 관련된 명령들이다.
In [12]: q = Question.objects.get(pk=1)

# 위 질문 레코드에 연결된 답변 항목을 모두 조회한다.
In [13]: q.choice_set.all()
Out[13]: <QuerySet [<Choice: 국밥>, <Choice: 카레2>, <Choice: 쌀국수>]>

# 질문의 답변 항목 3개를 생성해 본다.
In [14]: q.choice_set.create(choice_text='Pizza', votes=0)
Out[14]: <Choice: Pizza>

In [15]: q.choice_set.create(choice_text='짜장면', votes=0)
Out[15]: <Choice: 짜장면>

In [16]: c = q.choice_set.create(choice_text='라면', votes=0)

In [17]: c.question
Out[17]: <Question: 오늘의 맛집?>

In [18]: q.choice_set.all()
Out[18]: <QuerySet [<Choice: 국밥>, <Choice: 카레2>, <Choice: 쌀국수>, <Choice: Pizza>, <Choice: 짜장면>,
<Choice: 라면>]>

In [19]: q.choice_set.count()
Out[19]: 6

# 밑줄 2개(__)를 사용하여 객체 간의 관계를 표현할 수 있다.
In [21]: Choice.objects.filter(question__pub_date__year=current_year)
```

```
Out[21]: <QuerySet [<Choice: 국밥>, <Choice: 카레2>, <Choice: 쌀국수>, <Choice: 축구>, <Choice: 음악듣기>,
<Choice: Pizza>, <Choice: 짜장면>, <Choice: 라면>]>

In [22]: c = q.choice_set.filter(choice_text__startswith='What')

# choice_set 중에서 한 개의 답변 항목을 삭제할 수 있다.
In [24]: c.delete()
Out[24]: (0, {})
```

4.3 템플릿 시스템

- 템플릿 코드에 if 태그, for 태그 등이 있지만, 이들은 파이썬 프로그래밍 언어의 문법과는 다른 것이며, 템플릿 시스템에서만 사용되는 고유의 문법이다.
- 장고의 템플릿 시스템은 템플릿 문법으로 작성된 템플릿 코드를 해석하여 템플릿 파일로 결과물을 만들어준다. 이렇게 템플릿 코드를 템플릿 파일로 해석하는 과정을 장고에서는 렌더링이라고 한다.

4.3.1 템플릿 변수

- 템플릿 코드에서는 변수를 사용할 수 있다. 변수는 다음과 같은 형식으로 사용한다.

```
{{ variable }}
```

4.3.2 템플릿 필터

- 필터란 일반적인 용어로 어떤 객체나 처리 결과에 추가적으로 명령을 적용하여 해당 명령에 맞게 최종 결과를 변경하는 것을 말한다. 필터는 아래의 예시처럼 파이프(|) 문자를 사용한다.

```
# name 변수값의 모든 문자를 소문자로 바꿔주는 필터이다.
{{ name|lower }}

# 필터 체인: text 변수값 중에서 특수 문자를 이스케이프해주고, 그 결과 스트링에 HTML <p> 태그를 붙여준다.
{{ text|escape|linebreaks }}

# 필터 인자: bio 변수값 중에서 앞에 30개의 단어만 보여주고, 줄 바꿈 문자는 모두 없애준다.
{{ bio|truncatewords:30 }}

# 만일 list가 ['a','b','c']라면 결과는 "a // b // c"가 된다.
{{ list|join:" // " }}

# value 변수값이 False이거나 없는 경우, "nothing"으로 보여준다.
{{ value|default:"nothing" }}

# value 변수값의 길이를 반환한다.
{{ value|length }}

# value 변수값에서 HTML 태그를 모두 없애준다. 그러나 100% 보장하는 것은 아니다.
{{ value|striptags }}
```

```
# value 변수값이 1아 아니면 복수 접미사 s를 붙여준다.
{{ value|pluralize }}
{{ value|pluralize:"es" }}

# 더하기 필터이다.
{{ value|add:"2" }}

# first="5", second="10"이라면 결과는 15가 된다.
{{ first|add:second }}
```

4.3.3 템플릿 태그

- 템플릿 태그는 {% tag %} 형식을 가지며, 어떤 태그는 시작 태그와 끝 태그 둘 다 있어야 한다.

- {% for %} 태그
  - 리스트에 담겨 있는 항목을 순회하면서 출력할 수 있다.

```
<ul>
{% for athlete in athlete_list %}
    <li>{{ athlete.name }}</li>
{% endfor %}
</ul>
```

- {% if %} 태그
  - 변수를 평가하여 True이면 바로 아래의 문장이 표시된다.

```
{% if athelete_list %}
    Number of athletes: {{ athlete_list}length }}
{% elif ... %}
{% else %}
{% endif %}
```

- {% csrf_token %} 태그
  - POST 방식의 <form>을 사용하는 템플릿 코드에서는 CSRF(Cross Site Request Forgery) 공격을 방지하기 위하여 사용해야 한다. 폼 데이터에는 악의적인 스크립트 문장이 들어 있을 수도 있기 때문이다.

```
<form action="." method="post">{% csrf_token %}
```

- {% url %} 태그
  - 소스에 URL을 하드코딩하는 것을 방지하기 위한 것이다.

```
<form action="{% url 'polls:vote' question.id %}" method="post">
<form action="/polls/3/vote/" method="post">
```

- {% with %} 태그

- 특정 값을 변수에 저장해두는 기능을 한다.

```
{% with total=business.employees.count %}
    {{ total }} people works at business
{% endwith %}
```

4.3.4 템플릿 주석

■ 템플릿 코드에서도 주석문을 사용할 수 있다.
- 한줄 주석문: {# #} 형식, 예) {# greeting #}hello
- 여러줄 주석: {% comment %}, 예) {%

```
// 한줄 주석문은 {# #} 형식을 따른다.
{# greeting #}hello

// 여러줄 주석문은 {% comment %} 태그를 사용한다.
{% comment "Optional note" %}
<p>Commented out text here</p>
{% endComment %}
```

4.3.5 HTML 이스케이프

```
name = "<b>username"
name = "&lt;b&gt;username"

Hello, {{ name }}   --> Hello, <b>username

# 자동 이스케이프 기능 적용
# 첫번째 방법
Hello, {{ name|safe }}

# 두번째 방법
{% autoecape off %}
Hello, {{ name }}
{% endautoescape %}
```

4.3.6 템플릿 상속

■ 템플릿 상속을 통해서 템플릿 코드를 재사용할 수 있고, 사이트의 루앤필을 일관성 있게 보여줄 수 있다. 부모 템플릿은 템플릿의 뼈대를 만들어주고 {% block %} 태그를 통해 하위로 상속해줄 부분을 지정해주면, 자식 템플릿은 부모 템플릿의 뼈대는 그대로 재사용하고 {% block %} 부부만 채워주면 된다.

■ 부모 템플릿
- 06: title 블록
- 11~16: sidebar 블록
- 21: content 블록

[ch5/templates/base.html]

```
<!DOCTYPE html>
<html lang="en">
<head>
    {% load static %}
    <link rel="stylesheet" href="{% static 'admin/css/base.css' %}" />
    <title>{% block title %}My Amazing Site{% endblock %}</title>
</head>

<body>
    <div id="sidebar">
        {% block sidebar %}
        <ul>
            <li><a href="/">Project_Home</a></li>
            <li><a href="/admin/">Admin</a></li>
        </ul>
        {% endblock %}
        <br>
    </div>

    <div id="content">
        {% block content %}{% endblock %}
    </div>
</body>
</html>
```

■ 자식 템플릿

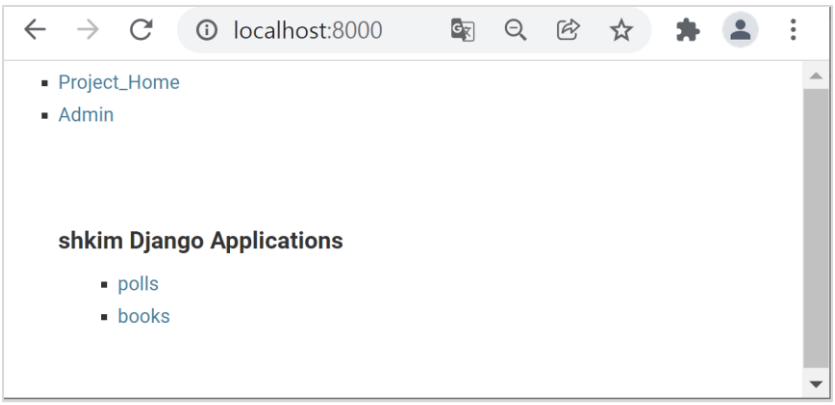
- 01: {% extends %} 태그는 사용하는 태그 중에서 가장 먼저 나와야 한다.
- 07: add 템플릿 필터를 사용하여 애플리케이션명에 필요한 문자열을 붙여주고 있다. urlvar는 polls:index가 된다.
- 12: 가독성을 높이기 위하여 블록명을 기입해도 된다.

[ch5/templates/home.html]

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

{% block content %}
    <h2>shkim Django Applications</h2>
    <ul>
        {% for appname in app_list %}
            {% with appname|add:":"|add:"index" as urlvar %}
                <li><a href="{% url urlvar %}">{{ appname }}</a></li>
            {% endwith %}
        {% endfor %}
    </ul>
{% endblock content %}
```

■ 템플릿 처리 결과



4.4 폼 처리하기

4.4.1 HTML에서의 폼

- 웹 사이트를 개발할 때 사용자로부터 입력을 받기 위해서 폼을 사용한다.
- HTTP 프로토콜 중 폼에서 사용할 수 있는 HTTP 메소드는 GET과 POST뿐이다. 장고는 폼의 데이터를 전송할 때는 POST 방식만을 사용하고 있다.

4.4.2 장고의 폼 기능

- 장고는 폼 처리를 위하여 다음의 3가지 기능을 제공한다.
  - 폼 생성에 필요한 데이터를 폼 클래스로 구조화하기
  - 폼 클래스의 데이터를 렌더링하여 HTML 폼 만들기
  - 사용자로부터 제출된 폼과 데이터를 수신하고 처리하기

4.4.3 폼 클래스로 폼 생성

- 폼 클래스 정의
  - 모든 폼 클래스는 django.forms.Form의 자식 클래스로 생성된다.

```
form django import forms

class NameForm(forms.Form):
    your_name = forms.CharField(label='Your name', max_length=100)
```

- 폼 클래스 렌더링 결과
  - 위의 폼 클래스가 템플릿 시스템에 의해 렌더링되면 다음과 같은 결과가 나온다.

```
<label for="your_name">Your name: </label>
<input id="your_name" type="text" name="your_name" maxlength="100">
```

- 폼 클래스를 템플릿에서 사용

- 랜더링 결과에 <form> 태크나 submit 버튼은 없는데, 이들은 개발자가 직접 템플릿에 넣어줘야 한다.

```
<form action="/your-name/" method="post">
    {% csrf_token %}
    {{ form }}
    <input type="submit" value="Submit" />
</form>
```

4.4.4 뷰에서 폼 클래스 처리

■ p194 참고

4.4.5 폼 클래스를 템플릿으로 변환

■ p195 참고

4.5 클래스형 뷰

■ 뷰는 요청을 받아서 응답을 반환해주는 호출 가능한 객체이다. 장고에서는 뷰를 함수로도 작성할 수 있고 클래스로도 작성할 수 있다. 지금까지 뷰를 함수로 작성하였지만, 사실 함수형 뷰보다 클래스형 뷰가 장점이 많다.

4.5.1 클래스형 뷰의 시작점

■ 클래스형 뷰를 사용하기 위해서는 가장 먼저 URLconf에서 함수형 뷰가 아니라 클래스형 뷰를 사용한다는 점을 표시해줘야 한다.

4.5.2 클래스형 뷰의 장점 - 효율적인 메소드 구분

■ GET, POST 등의 HTTP 메소드에 따른 처리 기능을 코딩할 때, IF 함수를 사용하지 않고 메소드명으로 구분할 수 있으므로 코드의 구조가 깔끔해진다.

```
# 함수형 뷰로 HTTP GET 메소드 코딩
from django.http import HttpResponse

def my_view(request):
    if request.method == 'GET':
        # 뷰 로직 작성
        return HpptResponse('result')


# 클래스형 뷰로 HTTP GET 메소드 코딩
# views.py
from django.http import HttpResponse
from django.views.generic import View
```

```
class MyView(View):
    def get(self, request):
        # 뷰 로직 작성
        return HttpResponse('result')
```

4.5.3 클래스형 뷰의 장점 - 상속 기능 가능

■ 다중 상속과 같은 객체 지향 기술이 가능하므로 클래스형 제네릭 뷰 및 믹스인 클래스 등을 사용할 수 있고, 이는 코드의 재사용성이나 개발 생산성을 획기적으로 높여준다.

■ 클래스형 뷰 작성 - TemplateView 상속

```
# some_app/urls.py
from django.urls import path
from some_app.views import AboutView

urlpatterns = [
    path('about/', AboutView.as_view()),
]


# some_app/views.py
from django.views.generic import TemplateView

class AboutView(TemplateView):
    template_name = "about.html"
```

```
# 다른 방식으로 TemplateView 제네릭 뷰를 사용할 수도 있다. 아래와 같이 지정해주면 views.py 파일이 필요없다.
# some_app/urls.py
from django.urls import path
from django.views.generic import TemplateView

urlpatterns = [
    path('about/', TemplateView.as_view(template_name="about.html")),
]
```

4.5.4 클래스형 제네릭 뷰

■ 장고에서는 공통된 로직을 미리 개발해 놓고 제공하는 뷰를 제네릭 뷰(generic view)라고 부른다. 제네릭 뷰는 클래스형 뷰로 구성되어 있다.

■ 장고의 제네릭 뷰 리스트

제네릭 뷰 분류	제네릭 뷰 이름	뷰의 기능 또는 역할
Base View	View TemplateView RedirectView	가장 기본이 되는 최상위 제네릭 뷰이다. 다른 모든 제네릭 뷰는 View의 하위 클래스이다. 템플릿이 주어지면 해당 템플릿을 렌더링해준다. URL이 주어지면 해당 URL로 리다이렉트시켜준다.
Generic Display View	ListView DetailView	조건에 맞는 여러 개의 객체를 보여준다. 객체 하나에 대한 상세한 정보를 보여준다.
Generic Edit View	FormView CreateView	폼이 주어지면 해당 폼을 보여준다.

	UpdateView DeleteView	
Generic Date View	ArchiveIndexView YearArchiveView MonthArchiveView WeekArchiveView DayArchiveView TodayArchiveView DateDetailView	조건에 맞는 여러 개의 객체 및 그 객체들에 대한 날짜 정보를 보여준다.

4.5.5 클래스형 뷰에서 폼 처리

- FormView 제너릭 뷰로 폼을 처리
  - FormView 제너릭 뷰를 사용하면 FormView 클래스에 이미 정의되어 있기 때문에 클래스 내에 get(), post() 메소드 정의도 불필요하게 된다. 개발자가 코딩할 내용이 더 단순해진다.

```
from .forms import MyForm
from django.views.generic.edit import FormView

class MyFormView(FormView):
    form_class = MyForm
    template_name = 'form_template.html'
    success_url = '/thanks/'

    def form_valid(self, form):
        # cleaned_data로 관련 로직 처리
        return super(MyFormView, self).form_valid(form)
```

~~4.6 로그 남기기~~