

1번ppt) Django - the web framework

Django

Django is high-level python web framework that encourages rapid development and clean, pragmatic design. It takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel.

- Web Framework
- Django Intro
- 요청과 응답
- Template
- HTML Form
- URL

Django는 빠른 개발과 깨끗하고 실용적인 디자인을 장려하는 고급 python 웹 프레임워크입니다. 웹 개발의 번거로움을 많이 덜어주기 때문에 번거로운 작업 없이 앱 작성에 집중할 수 있습니다.

Web framework

- Django : 파이썬 웹 프레임워크
- web(world wide web)
 - 인터넷에 연결된 컴퓨터를 통해 정보를 공유할 수 있는 전 세계적인 정보 공간
- static web page (정적 웹 페이지)
 - 일반적으로 HTML, CSS, JavaScript로 작성됨
 - 서버에 미리 저장된 파일이 사용자에게 그대로 전달되는 웹 페이지
 - 서버가 정적 웹 페이지에 대한 요청을 받은 경우
 서버는 추가적인 처리 과정 없이 클라이언트에게 응답을 보냄
 - 모든 상황에서 모든 사용자에게 동일한 정보를 표시
 - flat page 라고도 함
- dynamic web page (동적 웹 페이지)
 - 서버 사이드 프로그래밍 언어(Python, Java, C++ 등)가 사용되며,
 파일을 처리하고 데이터베이스와의 상호작용이 이루어짐
 - 웹 페이지에 대한 요청을 받은 경우 서버는 추가적인 처리 과정 이후
 클라이언트에게 응답을 보냄
 - 동적 웹 페이지는 방문자와 상호작용하기 때문에 페이지 내용은 그때그때 다름
- framework (=Application framework)
 - 프로그래밍에서 특정 운영 체제를 위한 응용 프로그램 표준 구조를 구현하는
 클래스와 라이브러리 모임
 - 재사용할 수 있는 수많은 코드를 프레임워크로 통합 → 재사용성
- web framework
 - 웹 페이지를 개발하는 과정에서 겪는 어려움을 줄이는 것이 주 목적
 - 데이터 베이스 연동, 템플릿 형태의 표준, 세션 관리, 코드 재사용 등의 기능을 포함
 - Application framework의 일종

Django를 사용해야 하는 이유

- 검증된 python 언어 기반의 web framework,
대규모 서비스에도 안정적이며 오랫동안 세계적인 기업들에 의해 사용됨
 - 스포티파이, 인스타그램, 드롭박스, 딜리버리 히어로
- Django는 다소 독선적인 framework이다.

framework architecture

- 소프트웨어 공학에서 사용 되는 **디자인 패턴** 중 하나
- 사용자 인터페이스로부터 프로그램 로직을 분리하여 애플리케이션의 시작적 요소나 이면에서 실행되는 부분을 서로 영향 없이 쉽게 고칠 수 있는 애플리케이션을 만들 수 있음

= MVC design pattern (model-view-controller)

- Django는 MTV Pattern 패턴이라고 함

MTV Pattern (model)

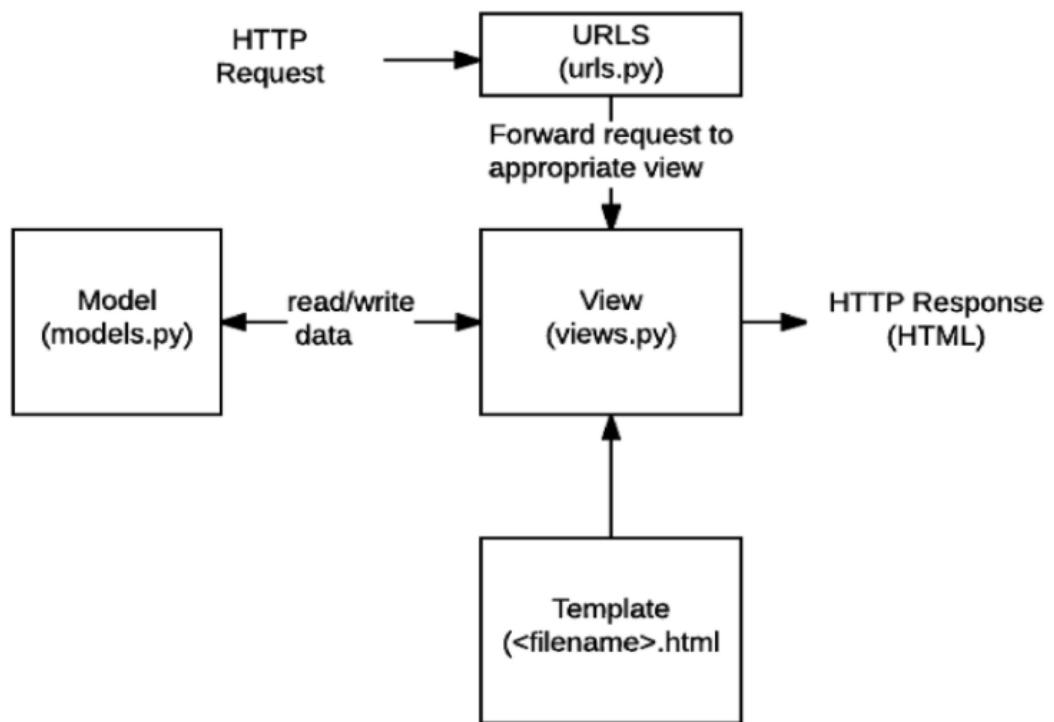
- Model
 - 응용프로그램의 데이터 구조를 정의하고 데이터베이스의 기록을 관리(추가, 수정, 삭제)
- Template
 - 파일의 구조나 레이아웃을 정의
 - 실제 내용을 보여주는 데 사용(presentation)
- View
 - HTTP 요청을 수신하고 HTTP 응답을 반환
 - Model을 통해 요청을 충족시키는데 필요한 데이터에 접근
 - template에게 응답의 서식 설정을 맡김

MVC Pattern 과 MTV Pattern의 차이

MVC Pattern	MTV (Django)
M _{odel}	M _{odel}
V _{iew}	T _{emplate}
C _{ontroller}	V _{iew}

- 어떤 식으로 동작하는지 매우 중요

MTV Pattern



Django intro

- Django 설치 전 가상환경 생성 및 활성화
- 현재 3.2가 LTS임 (버전 명시 하지 않으면 4.0으로 깔리니 주의)

• 일반적인 경우보다 장기간에 걸쳐 지원하도록 고안된 소프트웨어의 버전

• 컴퓨터 소프트웨어의 제품 수명주기 관리 정책

- Long Term Support (장기 지원 버전)
- 배포자는 LTS 확정을 통해 장기적이고 안정적인 지원을 보장함

1	가상환경 구축하기	<code>\$ python -m venv venv</code>
2	가상환경 실행시키기	<code>\$ source venv/Scripts/activate</code>
3	pip list로 목록 확인하기	<code>\$ pip list</code>
• Django 설치		

1	버전에 맞는 django 설치	<pre>\$ pip install django==3.2.12</pre>
1'	requirements.txt 설치 재귀(리컬시브) 사용	<pre>\$ pip install -r requirements.txt</pre>
	● 프로젝트 생성 *프로젝트 이름에는 Python이나 Django에서 사용중인 키워드를 피해야 한다. ‘-’ (하이픈)도 사용할 수 없다. ex. Django, text, class, django-test 등	
1	프로젝트 생성	<ul style="list-style-type: none"> · <code>django-admin startproject <프로젝트명></code> . <pre>\$ django-admin startproject firstpj .</pre>
2	Django 서버 시작하기(활성화)	<pre>\$ python manage.py runserver</pre>
	메인 페이지 로켓 확인	 <p>The install worked successfully! Congratulations!</p> <p>You are seeing this page because <code>DEBUG=True</code> in your settings file and you have not configured any URLs.</p>
	Application 생성 (복수형 권장)	<pre>\$ python manage.py startapp articles</pre>
	앱 등록 (INSTALLED_APPS에 등록) 반드시 생성 후 등록 (꼭) - INSTALLED_APPS에 먼저 작성(등록)하고 생성하려면 앱이 생성되지 않음	

```
# settings.py

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

- 프로젝트에서 앱을 사용하기 위해서는 반드시 INSTALLED_APPS 리스트에 추가해야 함
- **INSTALLED_APPS**
 - Django installation에 활성화 된 모든 앱을 지정하는 문자열 목록

앱 등록 시 주의 사항

```
INSTALLED_APPS = [
    # Local apps
    'articles',

    # Third party apps
    'djangorestframework',

    # Django apps
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
]
```

- 해당 순서를 지키지 않아도 수업 과정에서는 문제가 없지만, 추후 advanced 한 내용을 대비하기 위해 지키는 것을 권장

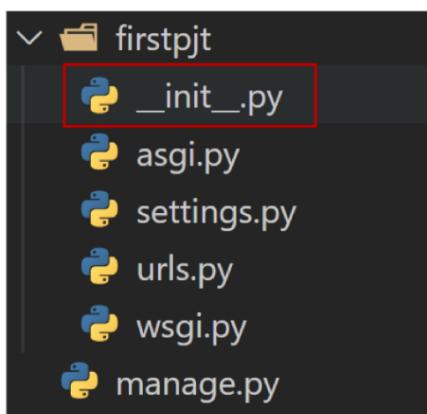
참고) LTS : Long Term Support (장기 지원 버전) - 컴퓨터 소프트웨어의 제품 수명주기 관리 정책, 배포자는 LTS 확정을 통해 장기적이고 안정적인 지원을 보장함.

✓ Project 과 Application의 차이

• Project

- Project(이하 프로젝트)는 Application(이하 앱)의 집합 (collection of apps)
- 프로젝트에는 여러 앱이 포함될 수 있음
- 앱은 여러 프로젝트에 있을 수 있음

| 프로젝트 구조 (1/6)



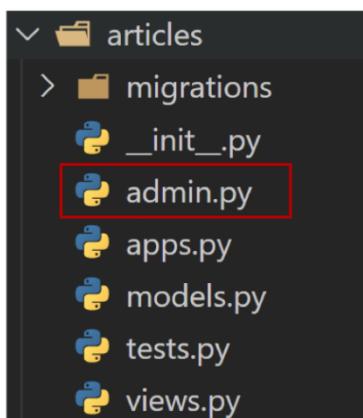
- `__init__.py`
- `asgi.py`
- `settings.py` : 애플리케이션의 모든 설정을 포함
- `urls.py` : 사이트 URL과 적절한 views의 연결을 지정
- `wsgi.py`
- `manage.py` : Django 프로젝트와 다양한 방법으로 상호작용하는 command라인 유ти리티

```
# manage.py Usage  
$ python manage.py <command> [options]
```

• Application

- 앱은 실제 요청을 처리하고 페이지를 보여주고 하는 등의 역할을 담당
- 하나의 프로젝트는 여러 앱을 가짐
- 일반적으로 앱은 하나의 역할 및 기능 단위로 작성함

Application 구조



- `admin.py` : 관리자용 페이지를 설정하는 곳
- `apps.py` : 앱의 정보가 작성된 곳
- `models.py` : 앱에서 사용하는 model을 정의하는 곳
- `tests.py` : 프로젝트의 테스트 코드를 작성하는 곳
- `views.py` : view 함수들이 정의되는 곳

요청과 응답

1. URLs

```
# urls.py

from django.contrib import admin
from django.urls import path
from articles import views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
]
```

- HTTP 요청(request)을
알맞은 view로 전달

2. View

```
# views.py

from django.shortcuts import render

def index(request):
    return render(request, 'index.html')
```

- HTTP 요청을 수신하고 HTTP 응답을
반환하는 함수 작성
- Model을 통해 요청에 맞는
필요 데이터에 접근
- Template에게 HTTP 응답 서식을 맡김

3. Templates

- 실제 내용을 보여주는데 사용되는 파일
- 파일의 구조나 레이아웃을 정의 (ex:HTML)
- **template 파일 경로 기본 값 - app폴더 안의 template 폴더로 지정**

```
<!-- articles/templates/index.html-->  
<h1>만나서 반가워요!</h1>
```

- 실제 내용을 보여주는데 사용되는 파일
- 파일의 구조나 레이아웃을 정의
(ex. HTML)
- Template 파일 경로의 기본 값은
app 폴더 안의 templates 폴더로
지정되어 있음

4. 추가 설정

| 추가 설정 (1/2)

• LANGUAGE_CODE

- 모든 사용자에게 제공되는 번역을 결정
- 이 설정이 적용 되려면 USE_I18N이 활성화되어 있어야 함
- [language-identifiers](#)

```
# settings.py  
  
LANGUAGE_CODE = 'ko-kr'  
  
TIME_ZONE = 'Asia/Seoul'
```

• TIME_ZONE

- 데이터베이스 연결의 시간대를 나타내는 문자열 지정
- USE_TZ가 True이고 이 옵션이 설정된 경우 데이터베이스에서 날짜 시간을 읽으면,
UTC 대신 새로 설정한 시간대의 인식 날짜&시간이 반환 됨
- USE_TZ가 False인 상태로 이 값을 설정하는 것은 error가 발생하므로 주의
- [List of tz database time zones](#)

• USE_I18N

- Django의 번역 시스템을 활성화해야 하는지 여부를 지정

• USE_L10N

- 데이터의 지역화 된 형식(localized formatting)을 기본적으로 활성화할지 여부를 지정
- True일 경우, Django는 현재 locale의 형식을 사용하여 숫자와 날짜를 표시

• USE_TZ

- datetimes가 기본적으로 시간대를 인식하는지 여부를 지정
- True일 경우 Django는 내부적으로 시간대 인식 날짜 / 시간을 사용

Template

- 데이터를 제어하는 도구이자 표현에 관련된 로직
- Django에서 사용하는 **built-in template system** (Django template language)

Django Template Language (DTL)

- Django template에서 사용하는 built-in template system
- 조건, 반복, 변수 치환, 필터 등의 기능을 제공
- 단순히 Python이 HTML에 포함 된 것이 아니며,
프로그래밍적 로직이 아니라 **프레젠테이션을 표현하기 위한 것**
- Python처럼 일부 프로그래밍 구조(if, for 등)를 사용할 수 있지만,
이것은 해당 Python 코드로 실행되는 것이 아님

DTL Syntax 4가지 (VFTC : Variable, Filters, Tags, Comments)

1. Variable {{variable}}

| DTL Syntax (1/4) – Variable

```
 {{ variable }}
```

- render()를 사용하여 views.py에서 정의한 변수를 template 파일로 넘겨 사용하는 것
- 변수명은 영어, 숫자와 밑줄(_)의 조합으로 구성될 수 있으나 밑줄로는 시작 할 수 없음
 - 공백이나 구두점 문자 또한 사용할 수 없음
- dot(.)를 사용하여 변수 속성에 접근할 수 있음
- render()의 세번째 인자로 {'key': value} 와 같이 딕셔너리 형태로 넘겨주며,
여기서 정의한 key에 해당하는 문자열이 template에서 사용 가능한 변수명이 됨

2. Filters : {{variable|filter}} 60개의 built-in template filters를 제공

DTL Syntax (2/4) – Filters

```
 {{ variable|filter }}
```

- 표시할 변수를 수정할 때 사용
- 예시)
 - name 변수를 모두 소문자로 출력
- 60개의 built-in template filters를 제공
- chained가 가능하며 일부 필터는 인자를 받기도 함

```
 {{ variable|truncatewords:30 }}
```

3. Tags : {%- tag %} {%- if %} {%- endif %} 약 24개의 built-in template tags를 제공

DTL Syntax (3/4) – Tags

```
{%- tag %}
```

- 출력 텍스트를 만들거나, 반복 또는 논리를 수행하여 제어 흐름을 만드는 등 변수보다 복잡한 일들을 수행
- 일부 태그는 시작과 종료 태그가 필요
- 약 24개의 built-in template tags를 제공

```
{%- if %}{%- endif %}
```

4. Comments 한 줄 주석 : {# #} / 여러 줄 주석 : {%- comment %} 주석 주석 {%- endcomment %}

DTL Syntax (4/4) – Comments

```
{# #}
```

- Django template에서 라인의 주석을 표현하기 위해 사용
- 아래처럼 유효하지 않은 템플릿 코드가 포함될 수 있음

```
{# {% if ... %} text {% else %} #}
```

- 한 줄 주석에만 사용할 수 있음 (줄 바꿈이 허용되지 않음)
- 여러 줄 주석은 {%- comment %}와 {%- endcomment %} 사이에 입력

```
{%- comment %}  
주석  
주석  
{%- endcomment %}
```

🌈 코드 작성 순서 (데이터의 흐름에 맞추어 작성) 🌈

- urls.py -> views.py -> templates 순

✓ [실습] DTL Syntax (p58 ~ 63)

[실습] DTL Syntax (1/6) – “Variable”

```
# urls.py

urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', views.index),
    path('greeting/', views.greeting),
]
```

- urls.py -> views.py -> templates 순으로 작성

```
# views.py

def greeting(request):
    return render(request, 'greeting.html', {'name': 'Alice'})
```

```
<!-- greeting.html -->

<p>안녕하세요 저는 {{ name }} 입니다.</p>
```

[실습] DTL Syntax (2/6) – “Variable”

```
# views.py

def greeting(request):
    foods = ['apple', 'banana', 'coconut',]
    info = {
        'name': 'Alice'
    }
    context = {
        'foods': foods,
        'info': info,
    }
    return render(request, 'greeting.html', context)
```

```
<!-- greeting.html -->
```

```
<p>안녕하세요 저는 {{ info.name }} 입니다.</p>
<p>제가 좋아하는 음식은 {{ foods }} 입니다.</p>
<p>{{ foods.0 }}을 가장 좋아합니다.</p>
```

```
<a href="/index/">뒤로</a>
```

[실습] DTL Syntax (3/6) – “Variable”

```
# urls.py

urlpatterns = [
    ...,
    path('dinner/', views.dinner),
]
```

```
<!-- dinner.html -->
```

```
<h1>오늘 저녁은 {{ pick }}!</h1>
<a href="/index/">뒤로</a>
```

```
# views.py

import random

def dinner(request):
    foods = ['족발', '햄버거', '치킨', '초밥']
    pick = random.choice(foods)
    context = {
        'pick': pick,
    }
    return render(request, 'dinner.html', context)
```

[실습] DTL Syntax (4/6) – “Filters”

```
# views.py

def dinner(request):
    foods = ['족발', '햄버거', '치킨', '초밥',]
    pick = random.choice(foods)
    context = {
        'pick': pick,
        'foods': foods,
    }
    return render(request, 'dinner.html', context)
```

```
<!-- dinner.html -->
```

```
<h1>오늘 저녁은 {{ pick }}!</h1>
<p>{{ pick }}은 {{ pick|length }}글자</p>
<p>{{ foods|join:", " }}</p>
```

```
<a href="/index/">뒤로</a>
```

[실습] DTL Syntax (5/6) – “Tags”

```
<!-- dinner.html -->

<h1>오늘 저녁은 {{ pick }}!</h1>
<p>{{ pick }}은 {{ pick|length }}글자</p>

<p>메뉴판</p>
<ul>
    {% for food in foods %}
        <li>{{ food }}</li>
    {% endfor %}
</ul>

<a href="/index/">뒤로</a>
```

[실습] DTL Syntax (6/6) – “Comments”

```
<!-- dinner.html -->

{# 이것은 주석입니다. #}

{% comment %}
    <p>여러 줄</p>
    <p>주석</p>
    <p>입니다.</p>
{% endcomment %}
```

Template inheritance (템플릿 상속) - for 재사용성

- 템플릿 상속은 기본적으로 코드의 재사용성에 초점을 맞춤
- 템플릿 상속을 사용하면 사이트의 모든 공통 요소를 포함하고, 하위 템플릿이 재정의 override 할 수 있는 블록을 정의하는 기본 “skeleton” 템플릿을 만들 수 있음

Template inheritance – “tags”

```
{% extends '' %}
```

- 자식(하위)템플릿이 부모 템플릿을 확장한다는 것을 알림
- 반드시 템플릿 최상단에 작성 되어야 함

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

- 하위 템플릿에서 재지정(overridden)할 수 있는 블록을 정의
- 즉, 하위 템플릿이 채울 수 있는 공간

[실습] Template inheritance (p66 ~ 68)

1. app_name/templates 디렉토리 외 템플릿 추가 경로 설정

```
TEMPLATES = [
{
    'BACKEND': 'django.template.backends.django.DjangoTemplates',
    'DIRS': [BASE_DIR / 'templates'],
    'APP_DIRS': True,
    'OPTIONS': {
```

2. base.html 작성

```
<nav>
    ...
</nav>
{% block content %}
{% endblock %}
<!-- bootstrap CDN -->
</body>
</html>
```

bootstrap 및 간단한 navbar 작성

3. index.html 변경

```
<!-- index.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>만나서 반가워요!</h1>
    <a href="/greeting/">greeting</a>
    <a href="/dinner/">dinner</a>
    <a href="/dtl-practice/">dtl-practice</a>
{% endblock %}
```

Template Tag - "include"

```
{% include '' %}
```

- 템플릿을 로드하고 현재 페이지로 렌더링
- 템플릿 내에 다른 템플릿을 "포함(including)"하는 방법

✓ [실습] Template Tag - "include" (p70 ~ 71)

The screenshot shows the VS Code interface. On the left, the Explorer sidebar displays a project structure with a 'templates' folder containing 'dj_nav.html' (which is highlighted with a red box). Other files like 'base.html', 'db.sqlite3', 'manage.py', and 'requirements.txt' are also listed. On the right, the editor pane shows the content of 'dj_nav.html':

```
dj_nav.html
templates > dj_nav.html
1 <nav class="navbar navbar-light bg-light">
2   <div class="container-fluid">
3     <a class="navbar-brand" href="#">Navbar</a>
4   </div>
5 </nav>
6
```

templates/ 디렉토리에 _nav.html 생성 후 base.html에 있던 네비게이션 코드 가져오기

```
<title>Document</title>
</head>
<body>
  {% include '_nav.html' %}
  <div class="container">
    {% block content %}
    {% endblock %}
  </div>
  <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
</body>
</html>
```

include tag를 통해 base.html에 _nav.html 포함시키기

*[주의] 파일명 앞에 _(언더바)는 단순히 include되는 템플릿이라는 것을 분류하기 위함이며, 특수한 기능이나 규칙을 포함하지 않음

Django template system (feat. Django 설계 철학)

1. 표현과 로직(view)를 분리
2. 중복 배제

HTML Form

HTML form element (핵심 속성 - *action, method*)

- 웹에서 사용자 정보를 입력하는 여러 방식(text, button, checkbox, file, hidden, image, password, radio, reset, submit)을 제공하고,
사용자로부터 할당된 데이터를 서버로 전송하는 역할을 담당
- 핵심 속성(attribute)
 - **action** : 입력 데이터가 전송될 URL 지정
 - **method** : 입력 데이터 전달 방식 지정

HTML input element (핵심 속성 - *name*,)

- 사용자로부터 데이터를 입력 받기 위해 사용
- type 속성에 따라 동작 방식이 달라짐
- 핵심 속성(attribute)
 - **name**
 - 중복 가능, 양식을 제출했을 때 name이라는 이름에 설정된 값을 넘겨서 값을 가져올 수 있음
 - 주요 용도는 GET/POST 방식으로 서버에 전달하는 파라미터(name은 key, value는 value)로 매핑하는 것
 - GET 방식에서는 URL에서 ?key=value&key=value 형식으로 데이터를 전달함

HTML label element

- 사용자 인터페이스 항목에 대한 설명(caption)을 나타냄
- label을 input 요소와 연결하기
 1. input에 id 속성 부여
 2. label에는 input의 id와 동일한 값의 for 속성이 필요

- label과 input 요소 연결의 주요 이점

- 시각적인 기능 뿐만 아니라 화면 리더기에서 label을 읽어 사용자가 입력해야 하는 텍스트가 무엇인지 더 쉽게 이해할 수 있도록 돕는 프로그래밍적 이점도 있음
- label을 클릭해서 input에 초점(focus)를 맞추거나 활성화(activate) 시킬 수 있음

HTML for attribute

- for 속성의 값과 일치하는 id를 가진 문서의 첫 번째 요소를 제어
 - 연결 된 요소가 labelable elements인 경우 이 요소에 대한 labeled control이 됨
- “labelable elements”
 - label 요소와 연결할 수 있는 요소
 - button, input(not hidden type), select, textarea ...

HTML id attribute

- 전체 문서에서 고유(must be unique)해야 하는 식별자를 정의
- 사용 목적
 - linking, scripting, styling 시 요소를 식별

HTTP

- HyperText Transfer Protocol
- 웹에서 이루어지는 모든 데이터 교환의 기초
- 주어진 리소스가 수행 할 작업을 나타내는 request methods를 정의
- HTTP request method 종류
 - GET, POST, PUT, DELETE ...

HTTP request method - GET

- 서버로부터 정보를 조회하는 데 사용
- 데이터를 가져올 때만 사용해야 함
- 데이터를 서버로 전송할 때 body가 아닌 Query String Parameters를 통해 전송
- 우리는 서버에 요청을 하면 HTML 문서 파일 한장을 받는데,
이때 사용하는 요청의 방식이 GET

[실습] Throw & Catch (p83 ~ 84)

```
# urls.py
urlpatterns = [
    ...,
    path('throw/', views.throw),
]
```

```
# views.py
def throw(request):
    return render(request, 'throw.html')
```

```
# urls.py
urlpatterns = [
    ...,
    path('catch/', views.catch),
]
```

```
# views.py
def catch(request):
    message = request.GET.get('message')
    context = {
        'message': message,
    }
    return render(request, 'catch.html', context)
```

```
<!-- throw.html -->
{% extends 'base.html' %}

{% block content %}
<h1>Throw</h1>
<form action="#" method="#">
    <label for="message">Throw</label>
    <input type="text" id="message" name="message">
    <input type="submit">
</form>
{% endblock %}
```

```
<!-- catch.html -->
{% extends 'base.html' %}

{% block content %}
<h1>Catch</h1>
<h2>여기서 {{ message }}를 받았어!!</h2>
<a href="/throw/">다시 던지러</a>
{% endblock %}
```

URL

Django URLs

- Dispatcher(발송자, 운항관리자)로서의 URL
- 웹 애플리케이션은 URL을 통한 클라이언트의 요청에서부터 시작됨

Variable Routing

- URL 주소를 변수로 사용하는 것
- URL의 일부를 변수로 지정하여 view 함수의 인자로 넘길 수 있음
- 즉, 변수 값에 따라 하나의 path()에 여러 페이지를 연결 시킬 수 있음
 - [사용 예시]
 - `path('accounts/user/<int:user_pk>/', ...)`
 - accounts/user/1 → (1번 user 관련 페이지)
 - accounts/user/2 → (2번 user 관련 페이지)

URL Path converters

URL Path converters

```
path('hello/<str:name>', views.hello),
```

- str
 - '/' 를 제외하고 비어 있지 않은 모든 문자열과 매치
 - 작성하지 않을 경우 기본 값
- int
 - 0 또는 양의 정수와 매치
- slug
 - ASCII 문자 또는 숫자, 하이픈 및 밑줄 문자로 구성된 모든 슬러그 문자열과 매치
 - ex) 'building-your-1st-django-site'
- uuid
- path

[실습] Variable Routing (p90)

```
# urls.py
urlpatterns = [
    ...,
    # path('hello/<str:name>/', views.hello),
    path('hello/<name>/', views.hello),
]
```

```
# views.py
def hello(request, name):
    context = {
        'name': name,
    }
    return render(request, 'hello.html', context)
```

```
<!-- hello.html -->

{% extends 'base.html' %}

{% block content %}
    <h1>만나서 반가워요 {{ name }}님!</h1>
{% endblock %}
```

App URL mapping

- app의 view 함수가 많아지면서 사용하는 path() 또한 많아지고, app 또한 더 많이 작성되기 때문에 프로젝트의 urls.py에서 모두 관리하는 것은 프로젝트 유지보수에 좋지 않음
- 이제는 각 app에 urls.py를 작성하게 됨

[실습] App URL mapping (p92 ~ 94)

```
# firstpjt/urls.py

from articles import views as articles_views
from pages import views as pages_views

urlpatterns = [
    ...,
    path('pages-index', pages_views.index),
]
```

pages 앱 생성 및 등록 후 url 작성

이렇게도 가능하지만..

```
# articles/urls.py

from django.urls import path
from . import views

urlpatterns = [
    path('index/', views.index),
    path('greeting/', views.greeting),
    path('dinner/', views.dinner),
    path('hello/<str:name>/', views.hello),
    path('dtl-practice/', views.dtl_practice),
    path('throw/', views.throw),
    path('catch/', views.catch),
]
```

```
# pages/urls.py

from django.urls import path

urlpatterns = [
]
```

각각의 앱 안에 urls.py를 생성하고 프로젝트 urls.py에서
각 앱의 urls.py 파일로 URL 매팅을 위탁

Including other URLconfs

- **include()**

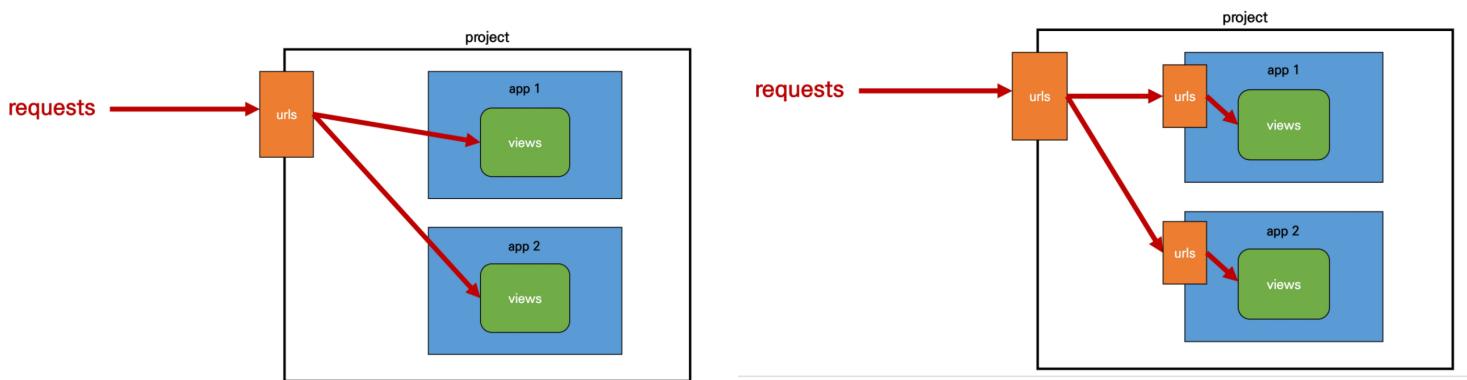
- 다른 URLconf(app1/urls.py)들을 참조할 수 있도록 도움
- 함수 include()를 만나게 되면, URL의 그 시점까지 일치하는 부분을 잘라내고,
남은 문자열 부분을 후속 처리를 위해 include된 URLconf로 전달
- django는 명시적 상대경로(**from .module import ..**)를 권장

```
# firstpjt/urls.py

from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('articles/', include('articles.urls')),
    path('pages/', include('pages.urls')),
]
```

urlpattern은 언제든지 다른 URLconf 모듈을 포함(include)할 수 있음



Naming URL patterns

- 이제는 링크에 url을 직접 작성하는 것이 아니라 path() 함수의 name 인자를 정의해서 사용
- Django Template Tag 중 하나인 url 태그를 사용해서 path() 함수에 작성한 name을 사용할 수 있음
- url 설정에 정의된 특정한 경로들의 의존성을 제거할 수 있음

```
path('index/', views.index, name='index'),
```

```
<a href="{% url 'index' %}>메인 페이지</a>
```

```
# articles/urls.py

urlpatterns = [
    path('index/', views.index, name='index'),
    path('greeting/', views.greeting, name='greeting'),
    path('dinner/', views.dinner, name='dinner'),
    path('dtl-practice/', views.dtl_practice, name='dtl_practice'),
    path('throw/', views.throw, name='throw'),
    path('catch/', views.catch, name='catch'),
    path('hello/<str:name>/', views.hello, name='hello'),
]
```

the 'name' value as called by the `{% url %}` template tag

URL template tag

```
{% url '' %}
```

- 주어진 URL 패턴 이름 및 선택적 매개 변수와 일치하는 절대 경로 주소를 반환
- 템플릿에 URL을 하드 코딩하지 않고도

DRY 원칙을 위반하지 않으면서 링크를 출력하는 방법

[실습] URL template tag 적용하기 (p 102 ~)

```
<!-- index.html -->

{% extends 'base.html' %}

{% block content %}
<h1>만나서 반가워요!</h1>
<a href="{% url 'greeting' %}">greeting</a>
<a href="{% url 'dinner' %}">dinner</a>
<a href="{% url 'dtl_practice' %}">dtl-practice</a>
<a href="{% url 'throw' %}">throw</a>
{% endblock %}

<!-- dinner, dtl_practice, greeting, throw.html-->

<a href="{% url 'index' %}">뒤로</a>
```

```
<!-- throw.html -->

{% extends 'base.html' %}

{% block content %}
<h1>Throw</h1>
<form action="{% url 'catch' %}" method="GET">
...
</form>

<a href="{% url 'index' %}">뒤로</a>
{% endblock %}
```

```
<!-- catch.html -->

{% extends 'base.html' %}

{% block content %}
<h1>Catch</h1>
<h2>여기서 {{ message }}를 받았어!!</h2>
<a href="{% url 'throw' %}">다시 던지려</a>
{% endblock %}
```