Django intro(0308 - 0309)

1 - 1) Dynamic web

- static web: 미리 저장된 정적파일(HTML, CSS, JS) 등을 제공
- dynamic web: 사용자의 요청에 따라 서버에서 추가적으로 실행
- 프로토콜: 요청과 응답(ex. 네이버 사이트에 요청을 보내면 화면에 응답이 오는 것과 비슷)

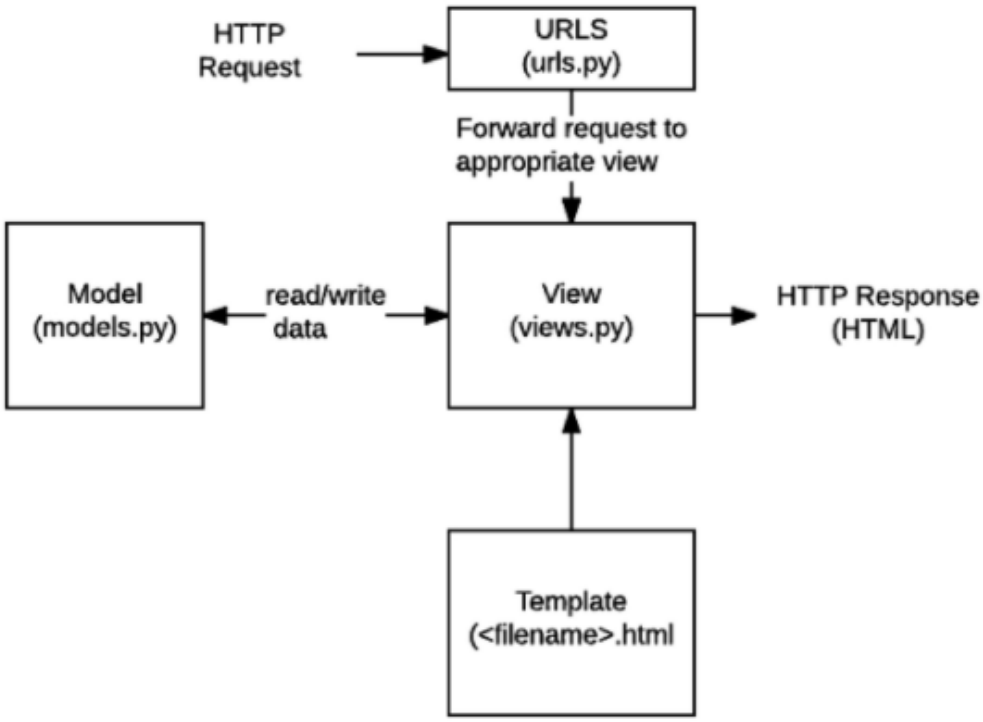
1 - 2) Django how

- django는 파이썬으로 작성된 오픈소스 웹 어플리케이션 프레임워크
    - 프레임워크 = 틀에 박힌 일
    - 프레임워크이므로 중간중간 양념을 칠 수는 있겠지만 디자인 패턴이 가지고 있는 흐름은 따라야 함

- 모델 - 뷰 - 컨트롤러 모델 패턴을 따름
    - 모델: 어플리케이션의 정보(데이터)
    - 뷰: 사용자 인터페이스 요소
    - 컨트롤러: 데이터와 비즈니스 로직 사이의 상호동작 관리

- django는 MTV
    - Model = Model: 데이터베이스 관리
    - View = Template: 레이아웃(화면)
    - Controller = View: 중심 컨트롤러(심장)

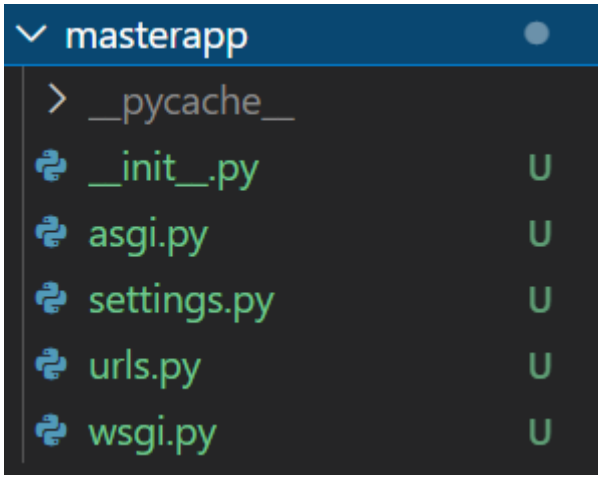
- django how
    - 아래 그림 참고



- 프로젝트 생성하기

```
django-admin startproject 'projectname' # 프로젝트 생성
python manage.py runserver # 사이트 확인
```

1 - 3) master_app 기본상태 알아보기



- **init**.py
  - 디렉토리를 패키지로 접근할 수 있도록 한다.

- settings.py
  - 웹사이트들의 설정을 담고 있다.
  - 세부사항을 담당한다.

- urls.py
  - 사용자의 요청을 맞닥뜨리는 곳

    - request를 알맞은 view로 연결해준다.
    - django는 기본으로 admin url을 제공한다.

- wsgi.py
    - 배포에 도움을 주는 파일

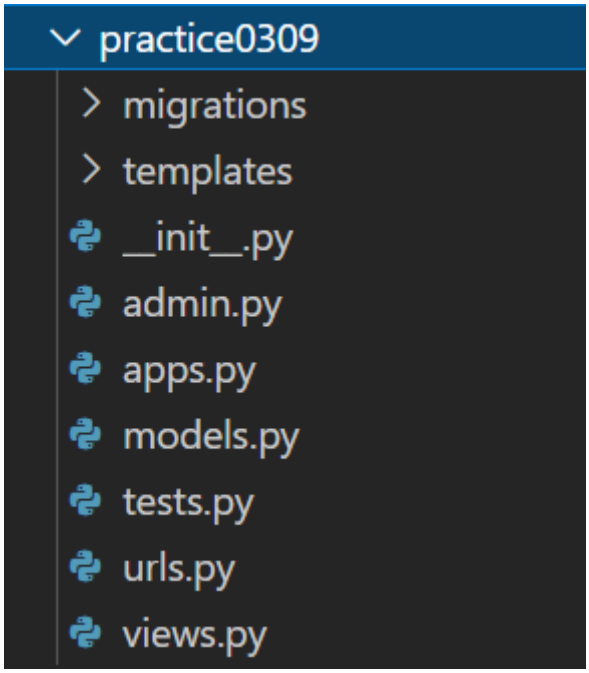
1 - 4) app 생성하기

- app 생성하기

```
python manage.py startapp 'app name'
```

    - app을 생성한 후
    - settings.py의 installed apps에서 출생신고 필수!(순서: 생성 → 출생신고)
    -
```
# Application definition
INSTALLED_APPS = [
    # my apps
    'workshop0308',
    'practice0309',
    'workshop0309',
```

- app 내부



    - test.py: test 코드를 작성하는 곳
    - views.py: mtv 중 view
    - models.py: mtv 중 model
    - apps.py: app에 대한 정보

1 - 5) 내가 이해한대로 정리해보기!

- url
  - 요청을 받는 곳
  - 요청과 view 함수와의 mapping이라고 생각하자
  - 요청을 받아야 하므로 'urlpatterns = []' 와 같은 것이 필요
  - 요청 경로가 어떤데? ~~게 생긴 경로면 ~~로 이동하자!

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

# <<masterapp의 url.py>>

urlpatterns = [
    path('admin/', admin.site.urls),    # 기본 제공되는 관리자 사이트
    # workshop0308/라는 경로가 오면
    # workshop0308.urls 라는 곳으로 이동('workshop0308' app의 url로 이동)
    # include: 접두사가 workshop0308이면 무조건 workshop0308.urls로 forward
하겠다
    path('workshop0308/', include('workshop0308.urls')),

]
```

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

# <<workshop0308의 url.py>>

urlpatterns = [
    # workshop0308/lotto 라는 경로가 오면 workshop0303의 views.py로 이동해
    # view에 정의된 lotto 함수를 실행
    path('lotto/', views.lotto)
]
```

- 포워딩
  - 요청을 받은 후 url을 지나
  - request를 view에 적합하게 만드는 과정

- view
  - control을 하는 곳이므로
  - def로 실행할 행동을 정의한다

```python
from django.shortcuts import render
import random

def lotto(request):
    lotto = random.sample(range(1, 46), 6)
    context = {
        'lotto': sorted(lotto),
        'greeting': 'Hello World!',
    }
    return render(request, 'workshop0308/lotto.html', context)

# view함수의 인자와 render의 첫번째 인자는 꼭 request여야 한다!
# lotto 함수의 리턴 > workshop0308/lotto.html에 context를 넘겨주는 것
```

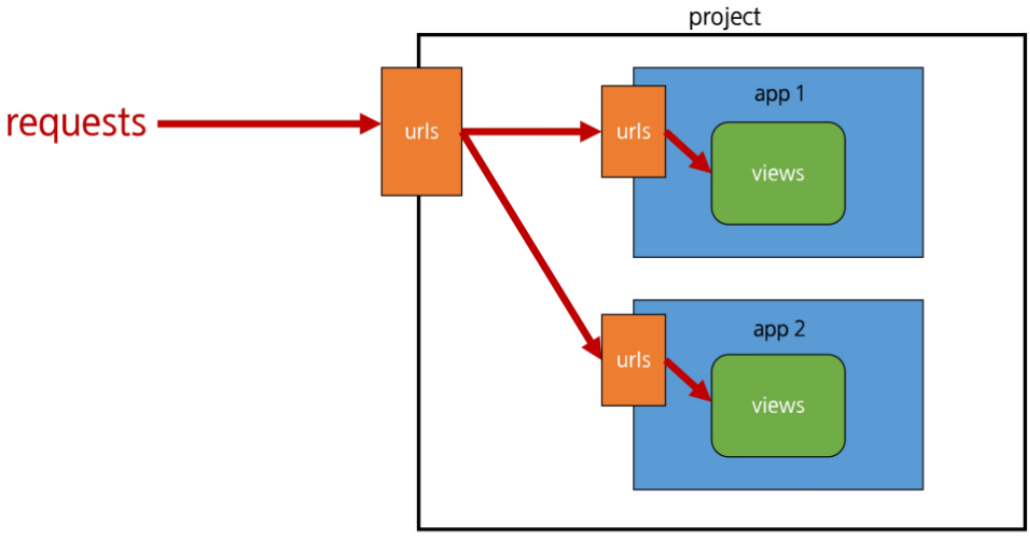
- template
  - 사용자에게 보여주는 곳이므로
  - html 파일이다

```python
<<masterapp의 base.html>>
# app과 path, view 수가 증가할 수록
# 마스터 앱에서 모든 url을 관리하기 힘들어진다.
# 그래서 각 app에 urls.py를 작성해서 사용하는 것이 좋다.

# 즉, base.html은 공통 기본틀만 잡고
# 남은 세부사항은 각 app의 url에서 틀을 상속받아 세부적으로 작성

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>
    {% block title %}                # 포트를 열어준다
    {% endblock title %}             # 포트를 닫는다
  </title>
</head>
<body>
  This is BASE TEMPLATE

  {% block header %}
  {% endblock header %}

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

</body>
</html>
```

```python
<<workshop0308 app의 lotto.html>>
{% extends 'base.html' %}   # base.html을 상속받았다는 의미

# base.html에서 열렸던 title 포트에 들어갈 내용
```

```
{% block title %}
  pick lotto
{% endblock title %}

# base.html에서 열렸던 content 포트에 들어갈 내용
{% block content %}
  <h1>Workshop0308</h1>
  <h2>{{ greeting }}</h2>
  <p>이번주 당신의 행운번호는</p>
  <ul>
    {% for number in lotto %}        # lotto는 view에서 가져온 context의
lotto
      <li>{{ number }}</li>          # 딕셔너리는 html에 들어오며 자동 해체되
므로
    {% endfor %}                     # context['lotto'] 이런식으로 접근할 필
요 없다
  </ul>

{% endblock content %}
```

위의 상황을 그림으로 나타낸 것이 아래 모습이다



2 - 1) DTL(장고 템플릿 언어)

- DTL 이란?
  - django template에서 사용하는 built-in template system
  - 조건, 반복, 변수치환, 필터 등의 기능 제공
  - 파이썬처럼 프로그래밍 구조를 사용할 수는 있으나 python 코드로 실행되는 것 X
  - Python이 html에 포함된 것 X, 프로그래밍적 로직 X, 프레젠테이션 표현을 위함

- DTL syntax
  - variable
    - {{ variable }}

```
예시)
{% block content %}
  <h1>{{ greeting }}</h1>   # greeting은 view에서 넘어온 딕셔너리 key값 중 하나
{% endblock content %}
```

- render()을 사용 > views.py에서 정의한 변수 → template 파일로
- 변수명: 영어, 숫자, 대쉬 ㅇ, 공백과 구두점 X
- . 을 사용하여 변수 속성에 접근 가능
- render()의 세번째 인자: {'key': value} 와 같이 딕셔너리 형태 / 여기서 key → template 에서 사용 가능한 변수명

```
예시)
def lotto(request):
    lotto = random.sample(range(1, 46), 6)
    context = {
        'lotto': sorted(lotto),
    }
    return render(request, 'workshop0308/lotto.html', context)

# render의 세번째 인자인 context 딕셔너리에서 사용된 key는
# workshop0308앱의 lotto.html라는 템플릿에서 사용 가능한 변수명이 된다
```

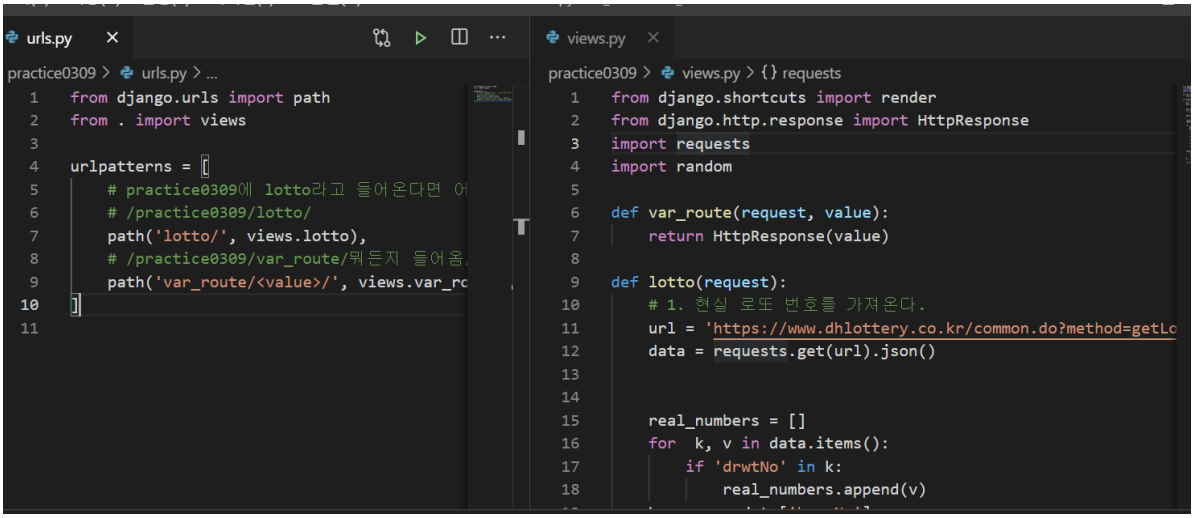
- filters
  - {{ variable | filter }}
  - ex) {{ name | lower}}: name 변수를 소문자로
  - 변수를 수정할 때 사용
  - 약 60개의 built-in template filters 제공
  - chained 가능
  - 일부 필터는 인자를 받기도 함
- tags
  - { % tags %}
  - 출력 텍스트 or 제어 흐름 만듦 등
  - 변수보다 복잡한 일 수행
  - 일부 태그는 시작과 종료태그 필요
  - 약 24개의 built-in template tags 제공
- comments
  - {# lorem ipsum #}
  - 줄의 주석을 표현하기 위해
  - 줄 바꿈 허용 X
  - 여러줄 주석은 {% comment %}와 {% endcomment %} 사이에 입력

- 템플릿 상속
  - 코드의 재사용성에 초점
  - {% extends %}
    - 자식(하위) 템플릿이 부모 템플릿을 확장한다는 뜻
    - 템플릿 최상단 위치: 주석 밑도 안된다!
  - {% block %}

- 하위 템플릿에서 재지정할 수 있는 블록을 정의
- 하위 템플릿이 채울 수 있는 공간

3 - 1) variable routing

- value를 변수처럼 사용 → 꼭 value를 인자로 함께 넘겨주어야 한다
- 디폴트 값은 str이지만 '타입:변수명'과 같은 형태로 쓸 수도 있다(예시 <int: value>)



3 - 2) element

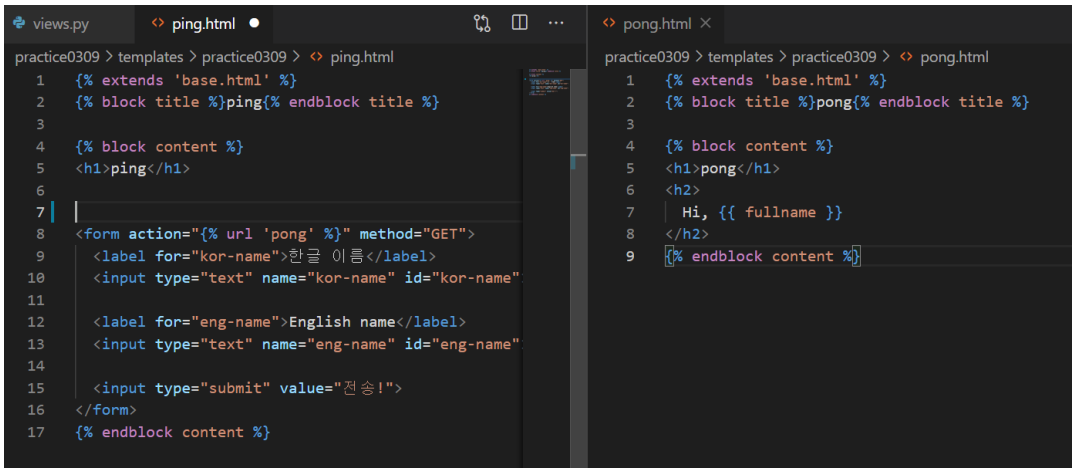
- form element
  - 사용자로부터 할당된 데이터를 서버로 전송하는 역할
  - action: 입력 데이터가 전송될 URL 지정, 수신인
  - method: 입력 데이터 전달 방식 지정, 공개 or 비공개
- input element
  - 사용자로부터 데이터 입력
  - 종이에 포스트잇을 붙여서 넘겨주는 것과 비슷하다
  - urls.py ping.html

practice0309 > templates > practice0309 > ping.html

```
{% extends 'base.html' %}
{% block title %}ping{% endblock title %}

{% block content %}
<h1>ping</h1>
<form action="" method="">
  <label for="kor-name">한글이름</label>
  <input type="text" id="kor-name">

  <label for="eng-name">English name</label>
  <input type="text" id="eng-name">
</form>

{% endblock content %}
```

    - id로 input과 label을 묶어준다
  - type 속성에 따라 동작 방식 다름

- name
  - name이라는 이름에 설정된 값을 넘겨서 값을 가져 올 수 있음
  - name은 key, value는 value로
  - get/post 방식으로 서버에 전달
- ping과 pong



views.py / ping.html

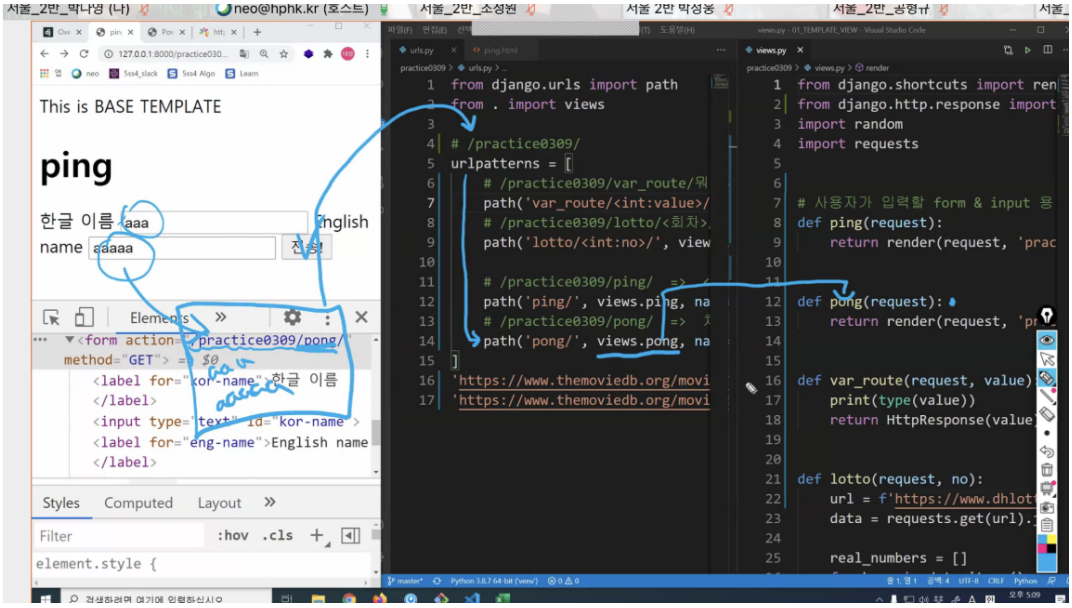
```
{% extends 'base.html' %}
{% block title %}ping{% endblock title %}

{% block content %}
<h1>ping</h1>

<form action="{% url 'pong' %}" method="GET">
  <label for="kor-name">한글 이름</label>
  <input type="text" name="kor-name" id="kor-name"

  <label for="eng-name">English name</label>
  <input type="text" name="eng-name" id="eng-name"

  <input type="submit" value="전송!">
</form>
{% endblock content %}
```

pong.html

```
{% extends 'base.html' %}
{% block title %}pong{% endblock title %}

{% block content %}
<h1>pong</h1>
<h2>
  Hi, {{ fullname }}
</h2>
{% endblock content %}
```

```
# ping → pong: url에 data를 녹여서 같이 보낸다 => form action에 url추가
# {% url 'pong' %}에서 pong을 namespacing이라고 한다.
# namespacing 아니더라도 실제 경로(ex: /ping.html)로도 불러올 수 있다.
# method = GET 이므로 공개상태로 pong으로 간다
```

ping에서 pong으로 가는 경로



3 - 3) method

- GET
  - 모든 데이터가 (URL에)공개 되어 전송
  - 데이터를 가져올때만 사용
  - Query String Parameters를 통해 전송
- POST: 데이터가 URL에 공개되지 않음

** 주의사항 모음 **

- settings가 있는 곳이 마스터앱이다
- 프로젝트 루트는 캐피탈로 써서 헷갈리지 않게 한다. why? 프로젝트 루트에서 code로 열기 안하면 가상환경을 못잡을 수 있다.
-

ntro > settings.py

```
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'intro.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        # default: installed_apps의 app dir/templates를 탐색한다
        # default 이외의 장소에서 찾게 하려면 아래 dirs에 추가해야함(추가 탐색하게 할 디렉토리 입력)
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
```