

장고 차근차근 시작하기

Second Edition

당신의 파이썬/장고 페이스메이커가 되겠습니다. ;)

EP19. 장고 템플릿 언어 (Django Template Language)

왜 템플릿을 사용하는 가?

- 코드 만으로 직접 복잡한 문자열을 조합하기 까다롭습니다.
 - 조합한 문자열이 조금만 복잡해져도 코드가 산으로 갑니다.
- 복잡한 문자열을 좀 더 편리하기 조합할 수 있도록 도와주는 라이브러리
 - 단지 HTML 응답 뿐만이 아니라, 다양한 문자열 조합에 사용할 수 있습니다.
 - 이메일, 푸시메세지, SMS 메세지 등의 조합

```
context = {  
    'name': 'Chinseok Lee',  
    'total': 3100000,  
    'item_list': [  
        {'name': '아이폰 X S', 'price': 1500000},  
        {'name': '아이폰 X Max', 'price': 1600000, 'is_discount': True},  
    ],  
}
```

안녕하세요. Chinseok Lee님.

총 3100000원 주문하셨습니다.

+ 아이폰 X S
- 1500000원

+ 아이폰 X Max
- 1600000원
- 할인적용

감사합니다.

템플릿을 사용한다면?

안녕하세요. {{ name }}님.

총 {{ total }}원 주문하셨습니다.

```
{% for item in item_list %}
+ {{ item.name }}
  - {{ item.price }}원
  {% if item.is_discount %}    - 할인적용{%
endif %}
{% endfor %}
```

감사합니다.

```
from django.shortcuts import render
from django.template.loader import render_to_string
```

```
def item_list(request):
    # context = ...
    return render(request,
        "app/item_list.html",
        context)
```

```
def item_list2(request):
    # context = ...
    rendered_str = render_to_string("app/item_list.html",
        context)
    return HttpResponse(rendered_str)
```

“Stupid 장고 템플릿 언어”의 철학

- 장고의 기본 철학
 - 풍성한 (Fat) Model
 - 비즈니스 로직이 없는 (Stupid) Template
 - 간결한 (Thin) View
- 템플릿 기능에 제한을 둌으로서, 비즈니스 로직을 템플릿 단에 구현함을 방지
 - 비즈니스 로직은 Model 에. 그리고 Form/ModelForm을 통한 유효성 검사 및 저장을 권장.
 - 다른 템플릿 엔진을 씀으로서, 이러한 제약에서 벗어날 수 있으나, 비권장.

템플릿 엔진 활용 주요코드

- `django.shortcuts.render` \Rightarrow `HttpResponse`
- `django.template.loader.render_to_string` \Rightarrow `str` ([공식문서](#))

```
# django 2.1.2 코드 : django/shortcuts.py
from django.http import HttpResponse
from django.template import loader
```

```
def render(request, template_name, context=None, content_type=None, status=None, using=None):
    content = loader.render_to_string(template_name, context, request, using=using)
    return HttpResponse(content, content_type, status)
```

```
# 활용코드
def index(request):
    return render(request, 'myapp/index.html', {
        'name': 'Ask Company',
    })
```

장고의 빌트인 백엔드

- `django.template.backends.django.DjangoTemplates`
 - 장고에서의 일반적인 템플릿 엔진
 - 대개의 장고 라이브러리에서 사용하는 템플릿 엔진
 - 함수 호출은 가능하되, 인자없는 함수만 가능.
 - 함수 호출 시에 소괄호를 쓰지 않습니다. Callable Object라면 템플릿 엔진에서 알아서 호출.
- `django.template.backends.jinja2.Jinja2`
 - 부분 지원

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [],  
        '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',  
            ],  
        },  
    },  
]
```

장고에서는 2개 템플릿 엔진만 쓸 수 있나요?

- 다른 템플릿 엔진도 물론 사용가능합니다만, 장고에서는 Django Template Engine에 한해서 다양한 기능을 지원하고 있습니다. Template Engine 별로 템플릿 문법이 다릅니다. Jinja2의 경우, 장고 템플릿의 superset 템플릿입니다.
 - Mako : <https://www.makotemplates.org>
 - Genshi : <https://genshi.edgewall.org>
 - HamIPy : <https://github.com/jessemiller/HamIPy>
- 한 프로젝트에서 다수의 템플릿 엔진을 활성화할 수 있으나,
 - 하나의 render 수행 시에는 하나의 템플릿 엔진만 선택적으로 사용됩니다.

CBV에서의 템플릿 처리

- `django.template.response.SimpleTemplateResponse` 클래스에서 로직 구현
 - 특정 CBV에서 `template_name` 인자 지정을 통한, 템플릿 지정
 - CBV 종류에 따라, 모델 이름을 따라 `template_name` 자동 지정
 - ex) `ListView`에서의 `Post` 모델 → “앱이름/post_list.html”
 - 각 CBV마다의 기본 템플릿 경로를 활용하시면 (관례에 기반), 코드를 줄이실 수 있습니다.

```
from django.http import HttpResponse

class SimpleTemplateResponse(HttpResponse):
    # 중략

    @property
    def rendered_content(self):
        template = self.resolve_template(self.template_name)
        context = self.resolve_context(self.context_data)
        content = template.render(context, self._request)
        return content
```


settings.TEMPLATES 설정 리스트

- BACKEND : 템플릿 엔진 지정
- DIRS : 템플릿을 둘 디렉토리 경로 리스트 by FileSystem Loader
- APP_DIRS : 앱별 templates 경로 추가 여부 by App Directory Loader
- OPTIONS / context_processors
 - 템플릿 내에서 디폴트 참조할 변수목록을 제공하는 함수 목록
 - 인자로 request를 받고, dict을 반환

```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [],  
        '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',  
            ],  
        },  
    },  
]
```

최소한의 템플릿 settings

- 각 앱들을 위한 템플릿은 각 “앱/templates/” 경로에 배치
 - 장고 앱은 재사용성에 포커스가 맞춰져 있기 때문.
- 프로젝트 전반적으로 사용할 템플릿은 DIRS에 명시한 경로에 배치

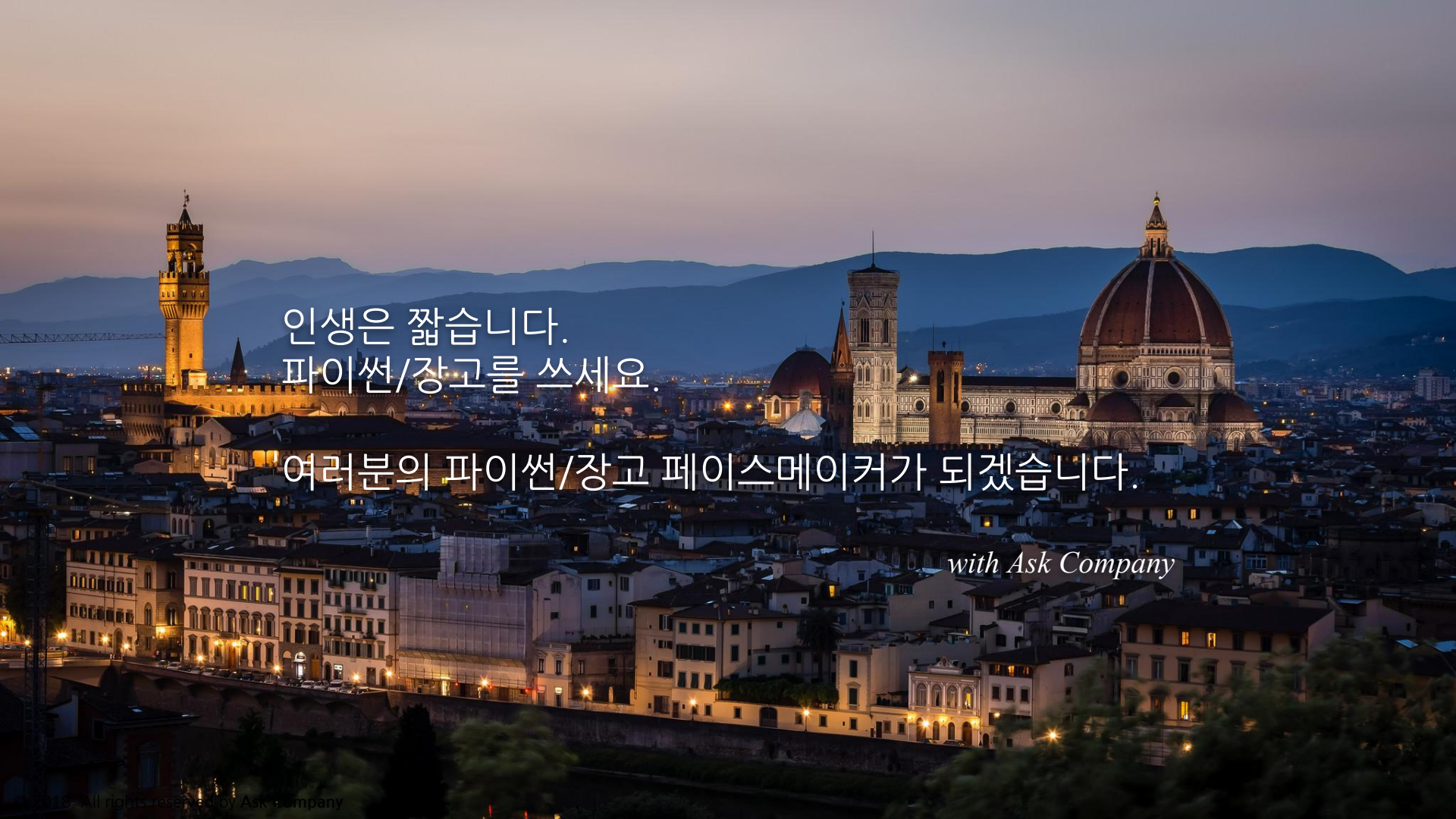
```
TEMPLATES = [  
    {  
        'BACKEND': 'django.template.backends.django.DjangoTemplates',  
        'DIRS': [  
            os.path.join(BASE_DIR, '프로젝트디렉토리명', 'templates'),  
        ],  
        'APP_DIRS': True,  
        'OPTIONS': {  
            'context_processors': [  
              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

장고 템플릿 태그/필터

- **유틸리티 성격**의 장고 템플릿 내에서 호출할 수 있는 함수 목록들
 - Django Template Tag : `{% 태그명 "인자1" "인자2" %}` 와 같은 방식으로 호출
 - 필터보다 범용적인 문법
 - Django Template Filter : `{{ 값|필터1:인자|필터2:인자|필터3 }}` 와 같은 방식으로 호출
- 다양한 빌트인 템플릿 태그/필터가 제공 ([공식문서](#))
 - for/endif, if/endif, include, load, verbatim 등
- 커스텀 템플릿 태그/필터 구현 지원 ([공식문서](#))
 - **주의** : 템플릿 태그/필터에 커스텀 함수를 구현할 수 있다고, 여기에 비즈니스 로직을 넣지는 마세요.
 - 유틸리티 목적으로만 사용하기.

디폴트 에러 템플릿 경로

- django/views/defaults.py 내에서 다음 경로 지정
 - “400.html”, “403.html”, “404.html”, “500.html”
 - 위 파일들을 구현하지 않으면, “기본 흰바탕 까만글씨 에러화면” 출력
 - 실제 서비스에서는 구현을 권장



인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

with Ask Company