환경 파일 작성

```
pip freeze > requirements.txt
pip install -r requirements.txt
```

form

지금까지는, HTML form 또는 input 태그들을 -> 사용자로부터 입력받았음

-> 사용자가 입력한 데이터는 "위험 할 수 있다" -> 위험하다

-> 유효성 검증 필요하다(모두 개발자가 구현하기엔 노력이 필요하다)

-> django에서는 반복되는 검증로직 및 반복 코드를 쉽게 만들어주는 것을 도와줌

-> **Django Form**

form 이라는 것은 유효성 검증의 도구 (하나)

외부의 악의적인, 또는 의도치 않는 공격에 대한 방어 수단

단순화 ,자동화 -> 안전하고 빠르게 코드를 작성할 수 있다

django form의 역할 세가지

1. 랜더링에 필요한 데이터를 준비 & 재구성
2. 데이터에 대한 html form을 생성
3. 클라이언트로부터 받은 데이터를 수신 및 처리

django form class

form내에 filed,widget,label 등을 활용하여 유효성을 검증한다

---

1. form 작성하기

articles / forms.py

```
from django import forms
# 장고로부터 forms를 import한다

class ArticleForm(forms.Form):
    title = forms.CharField(max_length=10)
    content = forms.CharField()
```

2. form 사용하기

articles / views.py

```
...
from .forms import ArticleForm
...

def new(request):
  form = ArticleForm()

  context = {
    'form' : form
  }

  return render(request, 'articles/new.html', context)

...
```

templates/ new.html

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


{% block content %}

<h1 class="text-center">NEW</h1>

<form action="{% url 'articles:create' %}" method="POST">
  {% csrf_token %}
  {% comment %} <label for="title">Title : </label>
  <input type="text" name="title">
  <br/>
  <label for="content">Content : </label>
  <textarea name="content" cols="30" row="5"></textarea>
  <br> {% endcomment %}

  {{form}}
```

```
  <input type="submit">
</form>
<hr>

<a href="{% url 'articles:index' %}">[BACK]</a>

{% endblock  %}
```

Form rendering options

1. as_p -> p 로써, p 처럼 (p태그로 감싸져서 랜더링 됨)
2. as_ul -> li태그로 감싸져서 랜더링 됨 (* ul태그가 필요한 경우 직접 작성)

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


{% block content %}

<h1 class="text-center">NEW</h1>

<form action="{% url 'articles:create' %}" method="POST">
  {% csrf_token %}
  {% comment %} <label for="title">Title : </label>
  <input type="text" name="title">
  <br/>
  <label for="content">Content : </label>
  <textarea name="content" cols="30" row="5"></textarea>
  <br> {% endcomment %}

  {{form.as_p}}

  <input type="submit">
</form>
<hr>

<a href="{% url 'articles:index' %}">[BACK]</a>

{% endblock  %}
```

Django -> HTML input 요소를 표현 2가지 방법

1. Form Filed

-> input에 대한 유효성 검사 로직을 처리하며, 템플릿에 직접 사용

2. Widgets

-> 웹 페이지의 HTML input 요소 랜더링

-> GET /POST 딕셔너리에서 데이터를 추출

-> Widgets은 반드시 Form fileds 안에 할당

Widgets이란?

Django에서 HTML input element를 어떻게 표현할지

HTML 랜더링 처리

- ## 주의사항

- Form Filed와 혼동x
- Form Filed는 input 유효성 검증
- Widgets는 웹페이지에서 "단순한" input element를 어떻게 랜더링 할지

Widgets 사용

articles / forms.py

```
from django import forms
# 장고로부터 forms를 import한다

class ArticleForm(forms.Form):
    title = forms.CharField(max_length=10)
    content = forms.CharField(widget=forms.Textarea)
```

Form filed & Widgets 응용

```
from django import forms
# 장고로부터 forms를 import한다

class ArticleForm(forms.Form):
    title = forms.CharField(max_length=10)
    content = forms.CharField(widget=forms.Textarea)

    REGION_A = "sl"
    REGION_B = "bs"

    REGIONS_CHOICES = [
        (REGION_A, '서울'),
        (REGION_B, '부산'),
    ]



    region = forms.ChoiceField(choices = REGIONS_CHOICES,widget=forms.Select())
```

---

forms의 단점 : forms를 사용하다보면, Model에 정의한 클래스와 유저로 부터 입력받기

위해서 Forms에 재정의 하는경우가 많을 수 있음

->

Model Form

Model form -> "model"을 통해 form class를 생성

- 사용법은 완전히 form 클래스와 동일

model form 사용

articles / forms.py

```
from django import forms
# 장고로부터 forms를 import한다
from .models import Article


# class ArticleForm(forms.Form):
#     title = forms.CharField(max_length=10)
#     content = forms.CharField(widget=forms.Textarea)

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        # fields = ('title',) 필요한것만 가져오거나
        # fields = '__all__'
        # 전체를 가져오거나
        exclude = ('title',)
        # 주의할점 exclude와 fields는 동시에 사용하면 안된다
```

forms 라이브러리에서 ModelForm 상속

정의한 외부클래스 안에, 내부클래스로 Meta <- 만들기

어떤 모델을 기반으로 Form을 생성할지 작성

Meta class

model의 정보를 작성하는 곳

ModelForm 사용하는경우 사용할 모델이 존재해야 함 -> meta class가 구성

해당 model에 정의한 field정보들을 form에 적용하기 위함

[참고] Inner Class(Nested Class -> 중첩 클래스)

클래스 내에 선언된 다른 클래스

관련 클래스들과 그룹화 -> 가독성 & 유지 보수성 ↑

외부 클래스에서 내부클래스로 접근할 수 x

[참고] Meta 데이터

"데이터에 대한 데이터"

[사진] -> (이미지 데이터) -> 데이터(언제 찍었는지(촬영 시간), 몇픽셀인지)

---

create view 수정!

```
def create(request):
  form = ArticleForm(request.POST)
  if form.is_valid():

    # form.save()
    # return redirect('articles:detail', form.pk)
    # 1번방법

    article = form.save()
    return redirect('articles:detail', article.pk)
    # 2번방법
  return redirect('articles:new')


  # 기존 코드
  # title = request.POST.get('title')
  # content = request.POST.get('content')
  # article = Article()
  # article.title = title
  # article.content = content
  # article.save()
  # return redirect('articles:detail', article.pk)
```

is_valid():

유효성 검사를 실행하고, 데이터가 유효한지 boolean값 반환

데이터 유효성 검사를 보장하기 위해 , 장고가 자체적으로 제공

[참고] 유효성 검사

요청한 데이터가 특정 조건에 충족되는지 확인하는 작업

데이터베이스의 각 필드에 올바르지 않은 데이터가 들어가지 않도록 막는작업

---

save()

Form에 바인딩된 데이터 db에 객체를 만들고 저장

ModelForm 하위 클래스 기존의 모델 인스턴스를 키워드인자 "instance"

이것이 제공되면 save() 해당 인스턴스 수정(UPDATE)

제공되지 않은 경우 save() 지정된 모델의 새 인스턴스 만듦(create)

form 유효성이 확인되지 않은경우 save()를 호출하면, 에러를 확인할 수 있음

```
form = ArticleForm(request.POST)

new_article = form.save() -> 새로운 객체 (create)


article = Article.objects.get(pk=1)  ->update
form = ArticleForm(request.POST, instance=article)
form.save()
```

---

create + new 합치기

new는 주석처리 또는 삭제할것

views / create

```
def create(request):
  if request.method == 'POST':
    form = ArticleForm(request.POST)
    if form.is_valid():

      form.save()
      return redirect('articles:detail', form.pk)
      # 1번방법

      # article = form.save()
      # return redirect('articles:detail', article.pk)
      # 2번방법
  else :
    form = ArticleForm()

  context = {
    'form' : form
  }
```

```
    return render(request, 'articles/create.html', context)
```

create.html

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


{% block content %}

<h1 class="text-center">CREATE</h1>

<form action="{% url 'articles:create' %}" method="POST">
  {% csrf_token %}

  {{form.as_p}}

  <input type="submit">
</form>
<hr>

<a href="{% url 'articles:index' %}">[BACK]</a>

{% endblock  %}
```

articles / urls.py

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


app_name = 'articles'
urlpatterns = [
    path("", views.index, name="index"),
    # path('new/', views.new, name="new"),
    path('create/', views.create, name="create"),
    path('<int:pk>/', views.detail, name="detail"),
    path('<int:pk>/delete', views.delete, name="delete"),
    path('<int:pk>/edit', views.edit, name="edit"),
    path('<int:pk>/update', views.update, name="update"),
]
```

---

edit + update

articles / urisl.py

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


app_name = 'articles'
```

```
urlpatterns = [
    path("", views.index, name="index"),
    # path('new/', views.new, name="new"),
    path('create/', views.create, name="create"),
    path('<int:pk>/', views.detail, name="detail"),
    path('<int:pk>/delete', views.delete, name="delete"),
    # path('<int:pk>/edit', views.edit, name="edit"),
    path('<int:pk>/update', views.update, name="update"),
]
```

views.py edit 함수 주석 또는 삭제

articles / detail.html

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

{% block content %}
<h2 class="text-center">DETAIL</h2>

<a href="{% url 'articles:update' article.pk %}" class="btn btn-
primary">EDIT</a>

<h3> {{article.pk}} 번째 글</h3>
<hr>
<p>제목 : {{article.title}}</p>
<p>내용 : {{article.content}}</p>
<p>작성 시각 : {{article.created_at}}</p>
<p>수정 시각 : {{article.updated_at}}</p>
<hr>

<form action="{% url 'articles:delete' article.pk %}" method="POST">
  {% csrf_token %}
  <button class="btn btn-danger">DELETE<button>
</form>


<a href="{% url 'articles:index' %}">[BACK]</a>

{% endblock  %}
```

EDIT.HTML -> UPDATE.HTML

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

```
{% block content %}

<h1 class="text-center">UPDATE</h1>

<form action="{% url 'articles:update' article.pk%}" method="POST">
  {% csrf_token %}
  {{form.as_p}}
  <input type="submit">
</form>
<hr>

<a href="{% url 'articles:index' %}">[BACK]</a>

{% endblock  %}
```

views/ update

```
def update(request, pk):
  article = Article.objects.get(pk = pk)
  if request.method == "POST":
    form = ArticleForm(request.POST, instance=article)
    if form.is_valid():
      form.save()
      return redirect('articles:detail', article.pk)
    # article.title = request.POST.get('title')
    # article.content = request.POST.get('content')
    # article.save()
  else:
    form = ArticleForm(instance=article)
  context = {
    'form' : form ,
    'article' : article
  }
  return render(request,'articles/update.html',context)
```