장고 핵심 기능 - Form

장고 Form 클래스 이해

```
$ python manage.py shell
>>> from blog.forms import PostSearchForm
>>> good_data = { 'search_word' : 'test' }
>>> error_data = {'search_word': ''}
>>> initial_data = { 'search_word': 'django'}

# 인자 없이 폼 객체를 만들면 언바인드 폼이 됨
>>> f = PostSearchForm()
>>> f.is_bound
False

# 빈 데이터를 사용해도 바인드 폼이 됨
>>> f = PostSearchForm({})
>>> f.is_bound
True

# 유효성 검사는 False
>>> f.is_valid()
False

# 맞는 데이터를 넣으면 유효성 검사는 True
>>> f = PostSearchForm(good_data)
>>> f.is_valid()
True
```

장고 Form 클래스 이해

```
# 틀린 데이터를 넣으면 유효성 검사는 False
# PostSearchForm에서 search_word 필드가 필수 임
>>> f = PostSearchForm(error_data)
>>> f.is_valid()
False

# 유효성 검사 오류 내역
>>> f.errors
{ 'search_word' : ['This field is required.']}

# 오류 내역을 다른 데이터 형식으로 변환할 수 있음
>>> f.errors.as_data()
{ 'search_word': [ValidationError(['This field is required.'])]}

# 오류 내역을 JSON 형식으로 변환
>>> f.errors.as_json()
'{"search_word": [ { "message": "This field is required",
"code":"required"}]}'
```

장고 Form 클래스 이해

```
# 언바운드 폼인 경우, 유효성 검사는 False이지만 오류는 아님
>>> f = PostSearchForm()
>>> f.is_bound
False
>>> f.is_valid()
False
>>> f.errors
{}

# 폼에 초기 데이터 지정가능. 그래도 언바운드 폼
>>> f = PostSearchForm(initial=initial_data)
>>> f.is_bound
False

# 폼의 내용을 보면 초기 데이터가 입력된 것을 확인
>>> print(f)
<tr><th>....

# has_changed() 메서드는 현재의 데이터가 초기 데이터와 다른지 검사
>>> f.has_changed()
True
```

장고 Form 클래스 이해

```
# 현재의 데이터와 초기 데이터가 동일
>>> f = PostSearchForm(initial_data, initial=initial_data)
>>> f.has_changed()
False

# 현재의 데이터와 초기 데이터가 다름
>>> f = PostSearchForm(good_data, initial=initial_data)
>>> f.has_changed()
True

# 달라진 필드명의 리스트
>>> f.changed_data
['search_word']

# 유효성 검사 전에 cleaned_data를 액세스하면 예외 발생
>>> f.cleaned_data
예외 발생
>>> f.is_valid()
False

# cleaned_data 속성에는 유효성 검사를 통과한 필드만 들어 있음
>>> f.cleaned_data
{ 'search_word': 'test' }
```

장고 Form 클래스 이해

```
# 폼 객체의 주요 기능은 유효성 검사와 HTML 텍스트로 렌더링하는 것
# 폼을 렌더링하고 그 결과를 <p> 태그 형식으로 출력
>>> print(f.as_p())

# 폼을 렌더링하고 그 결과를 <ul> 태그 형식으로 출력
>>> print(f.as_ul())

# 폼을 렌더링하고 그 결과를 <table> 태그 형식으로 출력
>>> print(f.as_table())

# 형식을 지정하지 않으면 <table> 태그 형식으로 출력
>>> print(f)

# auto_id=False로 지정하면 <lable> 태그와 id 속성이 생성되지 않음
>>> f = PostSearchForm(good_data, initial=initial_data, auto_id=False)
>>> print(f)
```

장고 Form 클래스 이해

```
# auto_id 값을 임의의 문자열로 지정
# 디폴트는 auto_id = 'id_%s'
>>> f = PostSearchForm(good_data, initial=initial_data, auto_id='id_for_%s')
>>> print(f)

# auto_id=True로 지정했을 때
>>> f = PostSearchForm(good_data, initial=initial_data, auto_id=True)
>>> print(f)
```

장고 Form 클래스 이해

```
# 폼의 각 필드를 액세스
# 장고 용어로 BoundField라고 하며, 폼의 각 필드에 대한 HTML 출력을 제어
>>> print(f['search_word'])

# 필드에 대한 <label> 태그 부분을 출력
>>> print(f['search_word'].label_tag())

# <label> 태그의 레이블 출력
>>> print(f['search_word'].label)
Search Word

# <input> 태그의 value 속서
>>> print(f['search_word'].value())
test

# 초기값이 설정된 언바인딩 폼에서 <input> 태그의 value 속성
>>> f = PostSearchForm(initial=initial_data)
>>> print(f['search_word'].value())
django

# BoundField 클래스의 속서및 메서드 확인 --> 공식 문서 참고
```

일반 폼 정의

❖ Model 클래스

```
class Album(models.Model):
    name = models.CharField('NAME', max_length=30)
    description = models.CharField('One Line Description', max_length=100,
                                   blank=True)

class Phto(models.Model):
    album = models.ForeignKey(Album, on_delete=models.CASCADE)
    title = mdoels.CharField('TITLE', max_length=30)
    description = models.TextField('Photo Description', blank=True)
    image = ThumbnailImageField('IMAGE', upload_to='photo/%Y/%m')
    upload_dt = models.DateTimeField('UPLOAD DATE', auto_now_add=True)
```

일반 폼 정의

❖ Form 클래스

```
from django import forms

class PhotoForm(forms.Form):
    album = forms.ModelChoiceField(queryset=Album.objects.all())
    title    = froms.CharField(label='TITLE', max_length=30)
    description = forms.CharField(label='Photo Description',
                                  widget=forms.Textarea, required=False)
    image    = ImageField(label='IMAGE')
    upload_dt   = forms.DateTimeField(label='UPLOAD DATE')
```

일반 폼 정의

❖ 모델 필드와 폼 필드간 매핑 룰

- 모델의 ForeignKey 필드는 폼의 MdoelChoiceField 필드로 매핑
  - 선택 항목들은 queryset 속성으로 지정
- 모델의 CharField필드는 폼의 CharField 필드로 매핑
  - 모델의 verbose_name 속성은 폼의 label 속서으로 매핑
  - max_length 속성도 그대로 매핑
- 모델의 TextField 필드는 폼의 CharField 필드로 매핑
  - widget 속성을 forms.Textarear로 지정
  - 모델 정의에서 blank=True이면 폼 필드는 required=False가 됨
- 모델의 ImageField 필드는 폼의 ImageField 필드로 매핑
- 모델의 upload_dt 필드는 자동으로 채워지는 속성(auto_now_add)이므로 폼에는 정의하지 않음

모델 폼 정의

❖ 모델 폼 정의

- 모델과 연관 없는 폼의 경우 정의
- 기존 모델이 있는 경우 이를 기반으로 모델 폼 정의

- 폼 필드를 정의하지 않고 모델의 필드를 이용

모델 폼 정의

❖ ModelForm 클래스 방식

```
from django import forms

class PhotoForm(forms.ModelForm):
    class Meta:
        model = Photo
        fields = ['title', 'image', 'description']
        # fields = '__all__'
        # exclude = ['description']
```

모델 폼 정의

❖ modelform_factory 함수 방식

```
from django.forms.models import modelform_factory
from photo.models import Photo

PhotoForm = modelform_factory(Photo, fields = '__all__')
```

- modelform_factory(model, form=ModelForm, fields=None, exclude=None, formfield_callback=None, widgets=None, localized_fields=None, labels=None, help_texts=None, error_messages=None, field_classes=None)
  - fields: 리턴하는 ModelForm에 포함될 필드
  - exclude: 리턴하는 ModelForm에 제외될 필드
  - formfield_callback : 모델의 필드를 받아서 폼 필드를 리턴하는 콜백 함수 지정
  - widgets: 모델 필드와 위젯을 매핑한 사전
  - localized_fields: 로컬 지역값이 필요한 필드를 리스트로 지정
  - labels: 모델 필드와 레이블을 매핑한 사전
  - help_texts: 모델 필드와 설명 문구를 매핑한 사전
  - error_messages: 모델 필드와 에러 메시지를 매핑한 사전
  - field_classes: 모델 필드와 폼의 필드 클래스를 매핑한 사전

모델 폼 정의

❖ 제너릭 뷰에서 폼 정의

- CreateView, UpdateView
  - 테이블의 레코드를 생성하거나 변경하는 역할
  - 뷰와 관련된 모델이 있어야 하고 레코드에 담을 데이터를 입력 받을 폼이 필요
  - ModelForm의 기능을 내부에 포함

```
class PhotoCreateView(CreateView)
    model = Photo
    fields = '__all__'

class PhotoUpdateView(UpdateView):
    model = Photo
    fields = '__all__'
```

  - ModelForm에서 사용하는 Meta 클래스를 사용하지 않고, 간단하게 model과 fields 속성을 정의
    --> 명시적으로 모델 폼을 정의하지 않아도 제너릭 뷰 내부적으로 적절한 모델 폼을 만들고 관련 뷰 처리를 수행

폼셋 정의

❖ 폼셋(Formset)

- 폼의 집합
- 일반 폼을 여러 개 묶어서 하나의 폼으로 취급
- `BaseFormSet` 클래스를 상속
- `formset_factory()` 함수를 사용하여 생성

폼셋 정의

❖ formset_factory 함수

```
from django.forms.formsets import formset_factory
from blog.forms import PostSearchForm

PostSearchFormSet = formset_factory(PostSearchForm)
```

폼셋 정의

❖ formset_factory 함수

```
formset_factory(form, formset=BaseFormSet, extra=1, can_order=False,
    can_delete=False, max_num=None, validate_max=False, min_num=None,
    validate_min=False)
```

- form: 폼셋을 만들 때 베이스가 되는 폼 지정
- formset:
  - 폼셋을 만들 때 상속받기 위한 부모 클래스를 지정
  - 일반적으로 BaseFormSet 클래스를 사용
  - 변경 시 BaseFormSet 클래스를 오버라이딩해 기능을 변경 후 사용
- extra: 폼셋을 보여줄 때 빈 폼을 몇 개 포함할지 지정. 디폴트는 1개
- can_order: 폼셋에 포함된 폼들의 순서 변경할지 여부 지정
- can_delete: 폼셋에 포함된 폼들의 일부를 삭제할 수 있는지 여부를 지정
- max_num: 폼셋에 포함된 폼들의 일부를 삭제할 수 있는지 여부를 지정
- valdiate_max:
  - True면 폼셋에 대한 유효성 검사를 수행할 때 max_num에 대한 검사 실시
  - 삭제 표시가 된 폼을 제외한 폼의 개수가 max_num 보다 작거나 같아야 유효성 검사 통과
- min_num: 폼셋을 보여줄 때 포함될 폼의 최소 개수 지정
- validate_min:
  - True면 폼셋에 대한 유효성 검사를 수행할 때 min_num에 대한 검사도 실시
  - 삭제 표시가 된 폼을 제외한 폼의 개수가 min_num 보다 크거나 같아야 유효성 검사를 통과

폼셋 정의

❖ 모델 폼셋 정의

- 모델 폼과 폼셋의 특징을 둘 다 갖고 있는 폼
- 데이터베이스 모델에 기초해 모델 폼을 만듦
- 그 모델 폼을 여러 개 묶은 것이 모델 폼 셋

- modelformset_factory() 함수로 폼셋을 정의
  - 모델 폼의 modelform_factory() 함수와 formset_factory() 함수를 합쳐 놓은 것
  - 모델 폼셋을 만들 때 BaseModelFormSet 클래스를 상속해서 정의 가능

```
from django.forms import modelformset_factory
from photo.models import Photo

PhotoFormSet = modelformset_factory(Photo, fields='__all__')
```

인라인 폼셋 정의

❖ 인라인 폼셋 정의

- 메인 폼에 종속된 폼셋
- 주종 관계는 테이블의 관계가 1:N 관계에서 외래 키로 연결된 경우로부터 비롯
- 1 테이블에 대한 폼을 메인 폼
- N 테이블에 대한 폼을 인라인 폼셋

- BaseInlineFormSet 클래스 상속 또는
- inlineformset_factory() 함수 사용(*)

인라인 폼셋 정의

❖ photo/views.py

```
class AlbumPhotoCV(LoginRequiredMixin, CreateVeiw):
   model = Album
   fields = ['name', 'descriptions']
```

❖ photo/forms.py

```
PhotoInlineFormSet = inlineformset_factory(Album, Photo,
    fields = ['image', 'title', 'description'], extra=2)
```