

장고 Form/ModelForm 제대로 알고 쓰기

당신의 파이썬/장고 페이스메이커가 되겠습니다. ;)

EP06. Form Validation

Form 유효성 검사가 수행되는 시점

```
# myapp/views.py
def post_new(request):
    if request.method == 'POST':
        form = PostForm(request.POST, request.FILES)
        if form.is_valid(): # 유효성 검사가 수행됩니다.
            form.save()
            # SUCCESS 후 처리
    else:
        form = PostForm()
    # ...
```

유효성 검사 호출 로직

form.is_valid() 호출 당시

1. form.full_clean() 호출

1. 각 필드 객체 별로

- 각 `필드객체.clean()` 호출을 통해 각 필드 Type에 맞춰 유효성 검사

2. Form 객체 내에서

- 필드 이름 별로 `Form객체.clean_필드명()` 함수가 있다면 호출해서 유효성 검사
- `Form객체.clean()` 함수가 있다면 호출해서 유효성 검사

2. 에러 유무에 따른 True/False 리턴

Form에서 수행하는 2가지 유효성 검사

1. Validator 함수를 통한 유효성 검사

- 값이 원하는 조건에 맞지 않을 때, ValidationError 예외를 발생
 - 주의: 리턴값은 사용되지 않습니다.

2. Form 클래스 내 clean, clean_ 멤버함수를 통한 유효성 검사 및 값 변경

- 값이 원하는 조건에 맞지 않을 때, ValidationError 예외를 발생
- 리턴값을 통해 값 반환

ValidationError 예외가 발생되면 ...

```
@html_safe
class BaseForm:
    # 중략
    def _clean_fields(self):
        for name, field in self.fields.items():
            if field.disabled:
                value = self.get_initial_for_field(field, name)
            else:
                value = field.widget.value_from_datadict(self.data, self.files, self.add_prefix(name))
            try:
                if isinstance(field, FileField):
                    initial = self.get_initial_for_field(field, name)
                    value = field.clean(value, initial)
                else:
                    value = field.clean(value)
                self.cleaned_data[name] = value
                if hasattr(self, 'clean_%s' % name):
                    value = getattr(self, 'clean_%s' % name)()
                    self.cleaned_data[name] = value
            except ValidationError as e:
                self.add_error(name, e)
```

<https://github.com/django/django/blob/2.1/django/forms/forms.py#L385>

Validator

함수형/클래스형 Validator (1)

- 함수형
 - 유효성 검사를 수행할 값 인자를 1개 받은 Callable Object
- 클래스형
 - 클래스의 인스턴스가 Callable Object

<https://docs.djangoproject.com/en/2.1/ref/validators/>

함수형/클래스형 Validator (2)

```
@deconstructible
class RegexValidator:
    regex = ''
    # 중략

    def __call__(self, value):
        regex_matches = self.regex.search(str(value))
        invalid_input = regex_matches if self.inverse_match else not regex_matches
        if invalid_input:
            raise ValidationError(self.message, code=self.code)

integer_validator = RegexValidator(
    _lazy_re_compile(r'^-?\d+\Z'),
    message=_('Enter a valid integer.'),
    code='invalid',
)

def validate_integer(value):
    return integer_validator(value)
```

<https://github.com/django/django/blob/2.1/django/core/validators.py#L152>

Validator 지정 #1

모델 필드 정의 시에 지정

```
import re
from django.db import models
from django.forms import ValidationError

def phone_number_validator(value):
    if not re.match(r'^010[1-9]\d{7}$'):
        raise ValidationError('{} is not an phone number'.format(value))

class Profile(models.Model):
    phone_number = models.CharField(max_length=11,
                                    validators=[phone_number_validator])

class ProfileForm(forms.ModelForm):
    class Meta:
        model = Profile
        fields = '__all__'
```

Validator 지정 #2

Form 필드 정의 시에 지정

```
import re
from django import forms
from django.forms import ValidationError

def phone_number_validator(value):
    if not re.match(r'^010[1-9]\d{7}$'):
        raise ValidationError('{} is not an phone number'.format(value))

class ProfileForm(forms.Form):
    phone_number = forms.CharField(validators=[phone_number_validator])
```

Validator 지정 #3

ModelForm이지만, Form Field 직접 지정

```
import re
from django.db import models
from django.forms import ValidationError

def phone_number_validator(value):
    if not re.match(r'^010[1-9]\d{7}$'):
        raise ValidationError('{} is not an phone number'.format(value))

class Profile(models.Model):
    phone_number = models.CharField(max_length=11)

class ProfileForm(forms.ModelForm):
    phone_number = forms.CharField(validators=[phone_number_validator])
    class Meta:
        model = Profile
        fields = '__all__'
```

빌트인 Validators

- RegexValidator
- EmailValidator
- URLValidator
- `validate_email`
- `validate_slug`
- `validate_unicode_slug`
- `validate_ipv4_address`, `validate_ipv6_address`,
`validate_ipv46_address`
- `validate_comma_separated_integer_list`
- `int_list_validator`

<https://docs.djangoproject.com/en/2.1/ref/validators/#built-in-validators>

빌트인 Validators

- MaxValueValidator
- MinValueValidator
- MaxLengthValidator
- MinLengthValidator
- DecimalValidator
- FileExtensionValidator : 파일 확장자 허용 여부
 - 주의 : 확장자만으로 정확히 그 포맷 임을 확정할 수는 없습니다.
- validate_image_file_extension
 - 이미지 확장자 여부. Pillow 설치 필수
- ProhibitNullCharactersValidator : 문자열에 '\x00' 포함여부

<https://docs.djangoproject.com/en/2.1/ref/validators/#built-in-validators>

모델 필드에 디폴트 적용된 validators

- `models.EmailField` (`CharField`)
 - `validators.validate_email` 적용
- `models.URLField`
 - `validators.URLValidator()` 적용
- `models.GenericIPAddressField`
 - `validators.ip_address_validators` 적용
- `models.SlugField`
 - `validators.validate_slug` 적용

https://github.com/django/django/blob/2.1/django/db/models/fields/__init__.py

Form clean 멤버함수

Form clean 멤버함수에게 기대하는 것

1. "필드별 Error 기록" 혹은 "Non 필드 Error 기록"
 - 값이 조건에 안 맞으면 ValidationError 예외를 통해 오류 기록
 - 혹은 add_error(필드명, 오류내용) 직접 호출을 통해 오류 기록
2. 원하는 포맷으로 값 변경
 - 리턴값을 통해 값 변경하기

멤버 함수별, 검사/변경의 책임

- `clean_필드명()` 멤버함수
 - 특정 필드별 검사/변경의 책임
 - `ValidationError` 예외 발생 시, 해당 필드 `Error`로 분류
- `clean()` 멤버함수
 - 다수 필드에 대한 검사/변경의 책임
 - `ValidationError` 예외 발생 시, `non_field_errors`로 분류
 - `add_error(...)` 함수를 통해 필드별 `Error` 기록도 가능

언제 validators를 쓰고, 언제 clean을?

가급적이면 모든 validators는 모델에 정의하고, ModelForm을 통해 모델의 validators 정보도 같이 가져오세요.

clean이 필요할 때

- 특정 Form에서 1회성 유효성 검사 루틴이 필요할 때
- 다수 필드값에 걸쳐서, 유효성 검사가 필요할 때
- 필드 값을 변경할 필요가 있을 때
 - validator는 값만 체크할 뿐, 값을 변경할 수는 없습니다.

샘플 코드

```

# myapp/models.py
class GameUser(models.Model):
    server = models.CharField(max_length=10)
    username = models.CharField(max_length=20)

# myapp/forms.py
class GameUserSignupForm(forms.ModelForm):
    class Meta:
        model = GameUser
        fields = ['server', 'username']

    def clean_username(self):
        'username 필드값의 좌/우 공백을 제거하고, 최소 3글자 이상 입력되었는 지 체크'
        username = self.cleaned_data.get('username', '').strip()
        if len(username) < 3:
            raise forms.ValidationError('3글자 이상 입력해주세요.')
        # 이 리턴값으로 self.cleaned_data['username'] 값이 변경됩니다.
        # 좌/우 공백이 제거된 (strip) username으로 변경됩니다.
        return username

```

(다음 페이지에 이어서)

(이전 페이지에 이어서)

```
def clean(self):
    cleaned_data = super().clean()
    if self.check_exist(cleaned_data['server'], cleaned_data['username']):
        # clean내 ValidationError는 non_field_errors 로써 노출
        raise forms.ValidationError('서버에 이미 등록된 username입니다.')
    return cleaned_data

def check_exist(self, server, username):
    return GameUser.objects.filter(server=server, username=username).exists()
```

개선된 코드

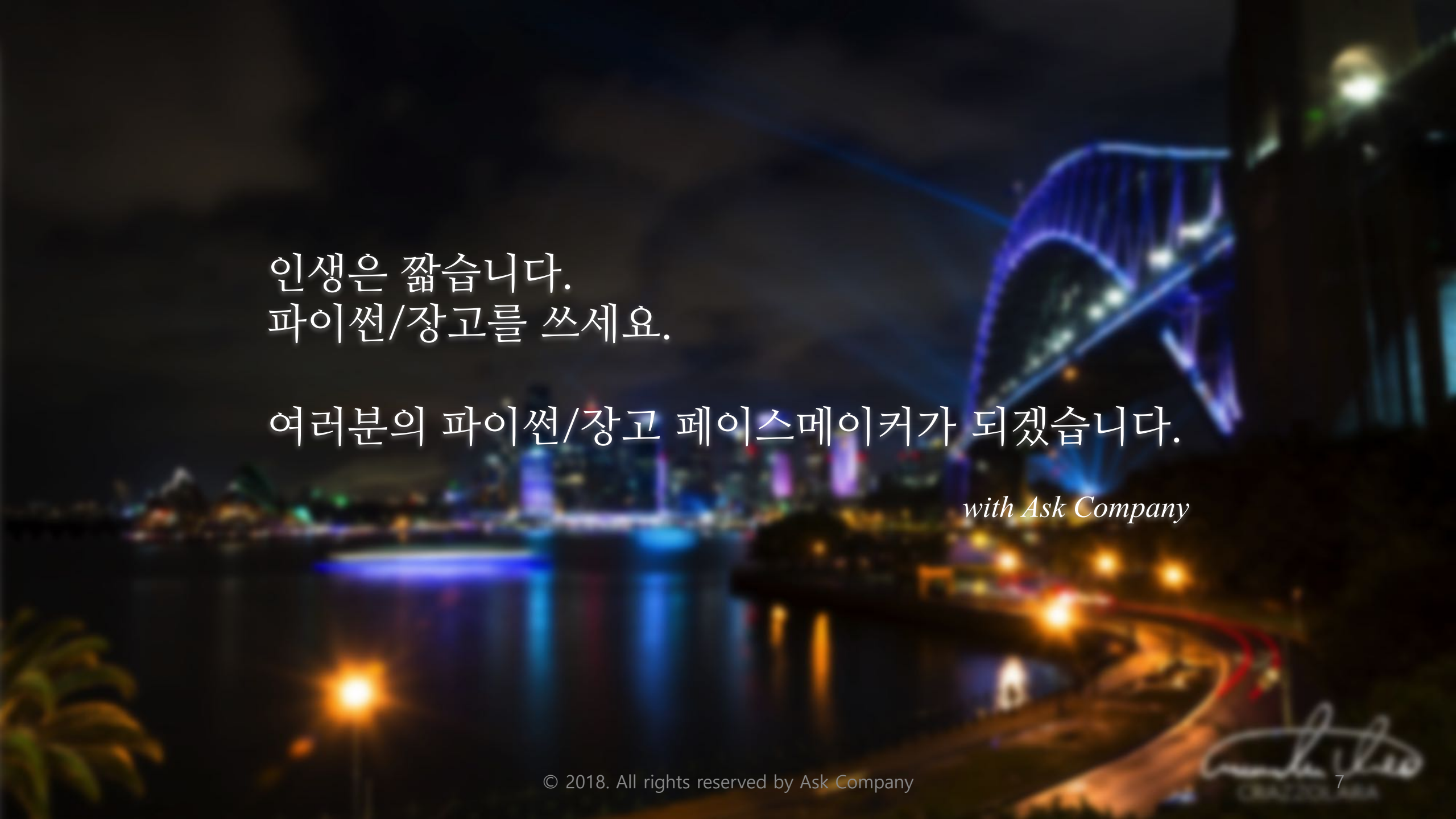
```
from django.core.validators import MinLengthValidator

class GameUser(models.Model):
    server = models.CharField(max_length=10)
    username = models.CharField(max_length=20,
                                validators=[MinLengthValidator(3)])

    class Meta:
        unique_together = [
            ('server', 'username'),
        ]

class GameUserSignupForm(forms.ModelForm):
    class Meta:
        model = GameUser
        fields = ['server', 'username']

    def clean_username(self):
        '''값 변환은 clean함수에서만 가능합니다.
        validator에서는 지원하지 않습니다.'''
        return self.cleaned_data.get('username', '').strip()
```

A nighttime photograph of a cityscape featuring a bridge with blue and white lights. The lights are reflected in the water below. The sky is dark, and the overall scene is illuminated by the city lights.

인생은 짧습니다.
파이썬/장고를 쓰세요.

여러분의 파이썬/장고 페이스메이커가 되겠습니다.

with Ask Company