

장고 Form/ModelForm 제대로 알고 쓰기

당신의 파이썬/장고 페이스메이커가 되겠습니다. ;)

EP10. built-in Class Based View를 통한 Form 처리

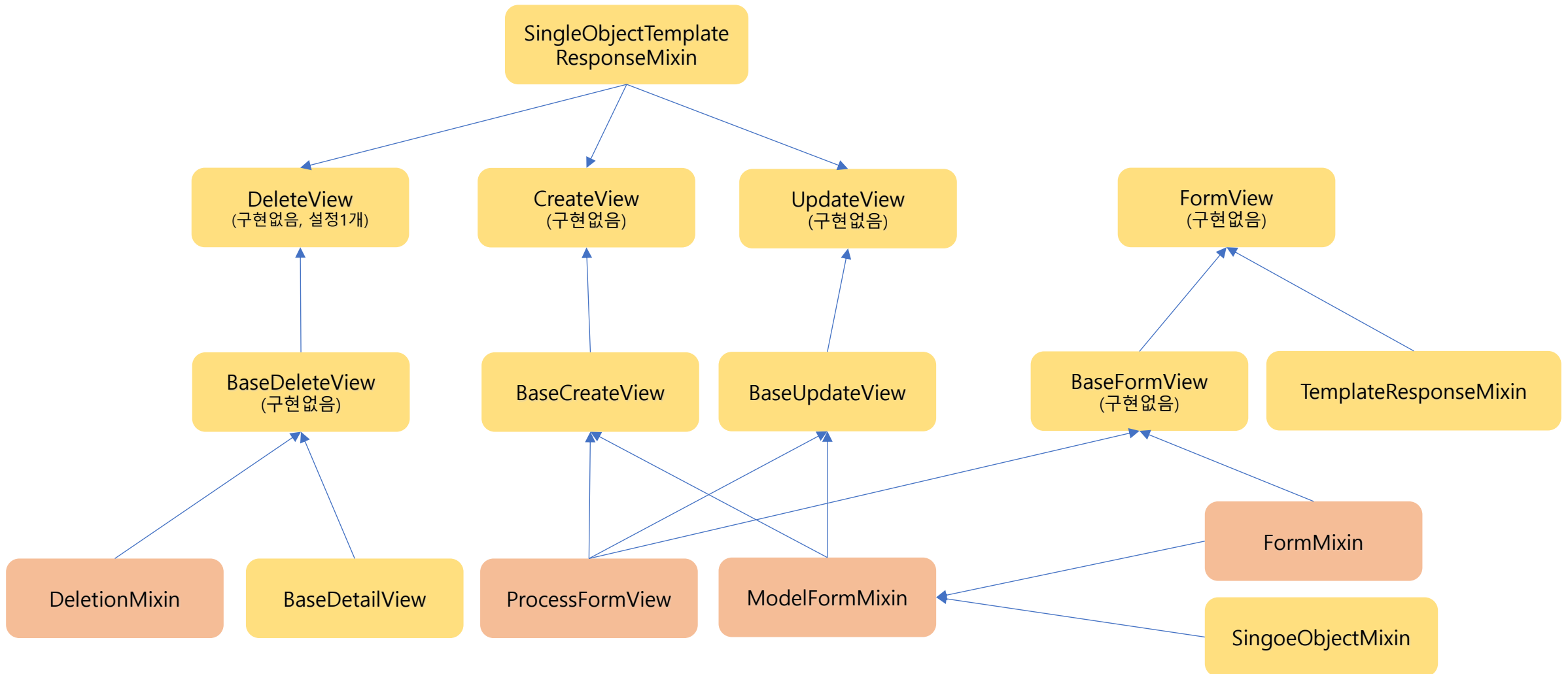
Built-in CBV API

Generic editing views

- FormView
 - TemplateResponseMixin, BaseFormView
- CreateView
 - SingleObjectTemplateResponseMixin
 - BaseCreateView ← ModelFormMixin, ProcessFormView
- UpdateView
 - SingleObjectTemplateResponseMixin
 - BaseUpdateView ← ModelFormMixin, ProcessFormView
- DeleteView
 - SingleObjectTemplateResponseMixin
 - BaseDeleteView ← DeletionMixin, BaseDetailView

<https://docs.djangoproject.com/en/2.1/ref/class-based-views/>

Form CBV 관계도



ProcessFormView

```
class ProcessFormView(View):
    """Render a form on GET and processes it on POST."""
    def get(self, request, *args, **kwargs):
        """Handle GET requests: instantiate a blank version of the form."""
        return self.render_to_response(self.get_context_data())

    def post(self, request, *args, **kwargs):
        """
        Handle POST requests: instantiate a form instance with the passed
        POST variables and then check if it's valid.
        """
        form = self.get_form()
        if form.is_valid():
            return self.form_valid(form)
        else:
            return self.form_invalid(form)

    # PUT is a valid HTTP verb for creating (with a known URL) or editing an
    # object, note that browsers only support POST for now.
    def put(self, *args, **kwargs):
        return self.post(*args, **kwargs)
```

```
class FormMixin(ContextMixin):
    # 생략 ...

    def get_context_data(self, **kwargs):
        """Insert the form into the context dict."""
        if 'form' not in kwargs:
            kwargs['form'] = self.get_form()
        return super().get_context_data(**kwargs)
```

<https://github.com/django/django/blob/2.1/django/views/generic/edit.py#L129>

Create 구현의 다양한 예

구현#1

```
def post_new(request):
    if request.method == 'POST':
        form = PostForm(request.POST, request.FILES)
        if form.is_valid():
            object = form.save()
            return redirect(object)
    else:
        form = PostForm()
    return render(request, "myapp/post_form.html", {
        "form": form,
    })
```

```
<form action="" method="post">
    {% csrf_token %}
    <table>
        {{ form }}
    </table>
    <input type="submit" />
</form>
```

구현#2

```
from django.shortcuts import resolve_url
from django.views.generic import FormView
from .forms import PostForm

class PostCreateView(FormView):
    form_class = PostForm
    template_name = 'myapp/post_form.html'

    def form_valid(self, form):
        self.object = form.save() # CBV ModelFormMixin에서 구현된 부분
        return super().form_valid(form)

    def get_success_url(self):
        # 주의: Post모델에 get_absolute_url() 멤버함수 구현 필요
        return resolve_url(self.object)
        # return self.post.get_absolute_url() # 대안 1
        # return reverse('blog:post_detail', args=[self.post.id]) # 대안 2

post_new = PostCreateView.as_view()
```

구현#3

```
from django.views.generic import CreateView
from .forms import PostForm

class PostCreateView(CreateView):
    form_class = PostForm

post_new = PostCreateView.as_view()
```

구현#4

```
from django.views.generic import CreateView
from .models import Post

class PostCreateView(CreateView):
    model = Post

post_new = PostCreateView.as_view()
```

CreateView와 UpdateView

```
class CreateView(SingleObjectTemplateResponseMixin, BaseCreateView):  
    """  
    View for creating a new object, with a response rendered by a template.  
    """  
    template_name_suffix = '_form'
```

```
class BaseCreateView(ModelFormMixin, ProcessFormView):  
    """  
    Base view for creating a new object instance.  
    Using this base class requires subclassing to provide a response mixin.  
    """  
    def get(self, request, *args, **kwargs):  
        self.object = None  
        return super().get(request, *args, **kwargs)  
  
    def post(self, request, *args, **kwargs):  
        self.object = None  
        return super().post(request, *args, **kwargs)
```

```
class UpdateView(SingleObjectTemplateResponseMixin, BaseUpdateView):  
    """View for updating an object, with a response rendered by a template."""  
    template_name_suffix = '_form'
```

```
class BaseUpdateView(ModelFormMixin, ProcessFormView):  
    """  
    Base view for updating an existing object.  
    Using this base class requires subclassing to provide a response mixin.  
    """  
    def get(self, request, *args, **kwargs):  
        self.object = self.get_object()  
        return super().get(request, *args, **kwargs)  
  
    def post(self, request, *args, **kwargs):  
        self.object = self.get_object()  
        return super().post(request, *args, **kwargs)
```

<https://github.com/django/django/blob/2.1/django/views/generic/edit.py#L175>

```

class ModelFormMixin(FormMixin, SingleObjectMixin):
    """Provide a way to show and handle a ModelForm in a request."""
    fields = None

    def get_form_class(self):
        """Return the form class to use in this view."""
        if self.fields is not None and self.form_class:
            raise ImproperlyConfigured(
                "Specifying both 'fields' and 'form_class' is not permitted."
            )
        if self.form_class:
            return self.form_class
        else:
            if self.model is not None:
                # If a model has been explicitly provided, use it
                model = self.model
            elif getattr(self, 'object', None) is not None:
                # If this view is operating on a single object, use
                # the class of that object
                model = self.object.__class__
            else:
                # Try to get a queryset and extract the model class
                # from that
                model = self.get_queryset().model

            if self.fields is None:
                raise ImproperlyConfigured(
                    "Using ModelFormMixin (base class of %s) without "
                    "the 'fields' attribute is prohibited." % self.__class__.__name__
                )

            return model_forms.modelform_factory(model, fields=self.fields)

```

```

def get_form_kwargs(self):
    """Return the keyword arguments for instantiating the form."""
    kwargs = super().get_form_kwargs()
    if hasattr(self, 'object'):
        kwargs.update({'instance': self.object})
    return kwargs

def get_success_url(self):
    """Return the URL to redirect to after processing a valid form."""
    if self.success_url:
        url = self.success_url.format(**self.object.__dict__)
    else:
        try:
            url = self.object.get_absolute_url()
        except AttributeError:
            raise ImproperlyConfigured(
                "No URL to redirect to. Either provide a url or define "
                "a get_absolute_url method on the Model."
            )
    return url

def form_valid(self, form):
    """If the form is valid, save the associated model."""
    self.object = form.save()
    return super().form_valid(form)

```

FormMixin

```
class FormMixin(ContextMixin):
    """Provide a way to show and handle a form in a request."""
    initial = {}
    form_class = None
    success_url = None
    prefix = None

    def get_initial(self):
        """Return the initial data to use for forms on this view."""
        return self.initial.copy()

    def get_prefix(self):
        """Return the prefix to use for forms."""
        return self.prefix

    def get_form_class(self):
        """Return the form class to use."""
        return self.form_class

    def get_form(self, form_class=None):
        """Return an instance of the form to be used in this view."""
        if form_class is None:
            form_class = self.get_form_class()
        return form_class(**self.get_form_kwargs())
```

```
def get_form_kwargs(self):
    """Return the keyword arguments for instantiating the form."""
    kwargs = {
        'initial': self.get_initial(),
        'prefix': self.get_prefix(),
    }

    if self.request.method in ('POST', 'PUT'):
        kwargs.update({
            'data': self.request.POST,
            'files': self.request.FILES,
        })
    return kwargs

def get_success_url(self):
    """Return the URL to redirect to after processing a valid form."""
    if not self.success_url:
        raise ImproperlyConfigured("No URL to redirect to. Provide a success_url.")
    return str(self.success_url) # success_url may be lazy

def form_valid(self, form):
    """If the form is valid, redirect to the supplied URL."""
    return HttpResponseRedirect(self.get_success_url())

def form_invalid(self, form):
    """If the form is invalid, render the invalid form."""
    return self.render_to_response(self.get_context_data(form=form))

def get_context_data(self, **kwargs):
    """Insert the form into the context dict."""
    if 'form' not in kwargs:
        kwargs['form'] = self.get_form()
    return super().get_context_data(**kwargs)
```

<https://github.com/django/django/blob/2.1/django/views/generic/edit.py#L10>

DeleteView

```
class DeleteView(SingleObjectTemplateResponseMixin, BaseDeleteView):
    """
    View for deleting an object retrieved with self.get_object(), with a
    response rendered by a template.
    """
    template_name_suffix = '_confirm_delete'

class BaseDeleteView(DeletionMixin, BaseDetailView):
    """
    Base view for deleting an object.
    Using this base class requires subclassing to provide a response mixin.
    """
```

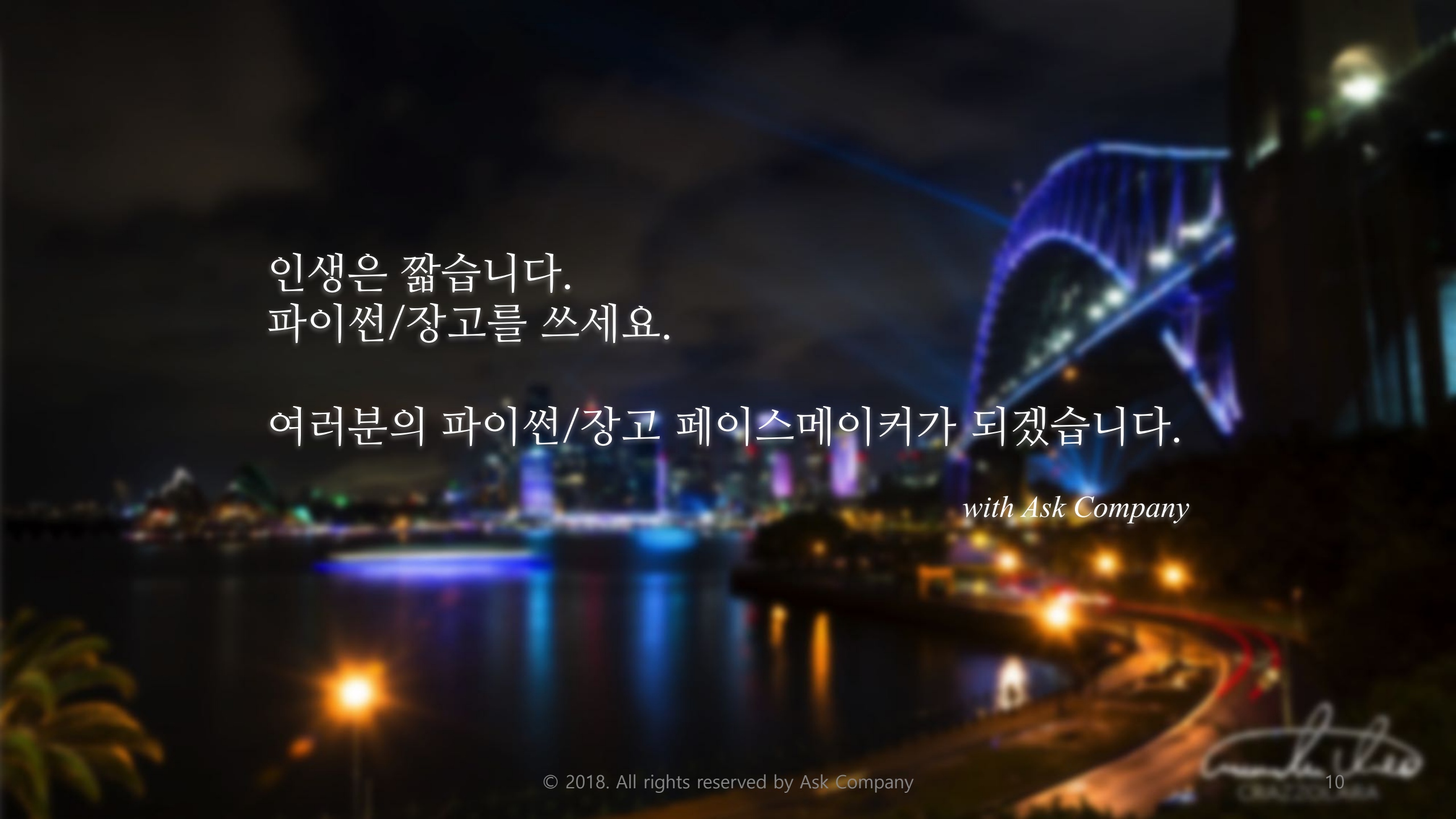
```
<form action="" method="post">
    {% csrf_token %}
    <input type="submit" value="삭제하겠습니다." />
</form>
```

```
class DeletionMixin:
    """Provide the ability to delete objects."""
    success_url = None

    def delete(self, request, *args, **kwargs):
        """
        Call the delete() method on the fetched object and then redirect to the
        success URL.
        """
        self.object = self.get_object()
        success_url = self.get_success_url()
        self.object.delete()
        return HttpResponseRedirect(success_url)

    # Add support for browsers which only accept GET and POST for now.
    def post(self, request, *args, **kwargs):
        return self.delete(request, *args, **kwargs)

    def get_success_url(self):
        if self.success_url:
            return self.success_url.format(**self.object.__dict__)
        else:
            raise ImproperlyConfigured(
                "No URL to redirect to. Provide a success_url.")
```

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