10장 심화학습

다양한 view의 세계

Create View 와 Form View는 뭐가 다를까?

```
class BaseCreateView(ModelFormMixin, ProcessFormView):
    def get(self, request, *args, **kwargs):
        self.object = None
        return super(BaseCreateView, self).get(request, *args, **kwargs)

    def post(self, request, *args, **kwargs):
        self.object = None
        return super(BaseCreateView, self).post(request, *args, **kwargs)
```

```
class BaseFormView(FormMixin, ProcessFormView):
    """
    A base view for displaying a form
    """
```

ModelFormMixin이 하는 일

```
class ModelFormMixin(FormMixin, SingleObjectMixin):
    """
    A mixin that provides a way to show and handle a modelform in a request.
    """
    fields = None

    def get_form_class(self):
    def get_form_kwargs(self):
    def get_success_url(self):
    def form_valid(self, form):
```

Create View 와 Update View의 차이

```
class BaseCreateView(ModelFormMixin, ProcessFormView):
    self.object = self.get_object()

class BaseUpdateView(ModelFormMixin, ProcessFormView):
    self.object = None
```

> ## Class based view 쓰세요. 두번 쓰세요
>
> '''목적에 맞게'''

Django extra views

상황에 따라 대단히 유용한 CBV들을 제공해 준다.

- FormSet and ModelFormSet views - The formset equivalents of FormView and ModelFormView.
- InlineFormSetView - Lets you edit formsets related to a model (uses inlineformset_factory)
- CreateWithInlinesView and UpdateWithInlinesView - Lets you edit a model and its relations
- GenericInlineFormSetView, the equivalent of InlineFormSetView but for GenericForeignKeys
- Support for generic inlines in CreateWithInlinesView and UpdateWithInlinesView
- Support for naming each inline or formset with NamedFormsetsMixin
- SortableListMixin - Generic mixin for sorting functionality in your views
- SearchableListMixin - Generic mixin for search functionality in your views

Mixin

```python
class Waffle:
    # 와플을 넣어드립니다.

class Icecream:
    # 아이스크림을 넣어드립니다.

class MixedWhat(Waffle, Icecream):
    # 이것은 와플 아이스크림일까요? 아이스크림 와플콘일까요?
```

Mixin은 다중 상속이다.

다중상속은...?

MRO

```
Python has the same structure as Perl, but, unlike Perl, includes it in the
syntax of the language. The order of inheritance affects the class semantics.
Python had to deal with this upon the introduction of new-style classes, all
of which have a common ancestor, object. Python creates a list of classes
using the C3 linearization (or Method Resolution Order (MRO)) algorithm. That
algorithm enforces two constraints: children precede their parents and if a
class inherits from multiple classes, they are kept in the order specified in
the tuple of base classes (however in this case, some classes high in the
inheritance graph may precede classes lower in the graph[8]). Thus, the
method resolution order is: D, B, C, A.[9]
```

위키백과-다중상속의 다이아몬드 프라블럼

손으로 풀어봅시다 MRO 알고리즘

```
                          6
                         ---
Level 3                 | O |                   (more general)
                      /  ---  \
                     /    |    \                      |
                    /     |     \                     |
                   /      |      \                    |
                  ---    ---    ---                   |
Level 2        3 | D | 4| E |  | F | 5                |
                  ---    ---    ---                   |
                   \  \ _ /       |                   |
                    \    / \ _    |                   |
                     \  /      \  |                   |
                      ---      ---                    |
Level 1            1 | B |    | C | 2                 |
                      ---      ---                    |
                        \      /                      |
                         \    /                      \ /
                           ---
Level 0                 0 | A |                 (more specialized)
                           ---
```

```
L[O] = O
L[D] = D O
L[E] = E O
L[F] = F O
```

그래서 그랬나 봅니다. CBV vs FBV

Why are mixins so bad

```
I think mixins are what people use when they can't use decorators or some
other form of composition to solve a problem. Mixins aren't inherently bad
they just obfuscate where functionality is derived from a bit too much.

Inheritance is useful, but is often over used. Having a few base classes
doesn't indicate a problem, but when code you're relying on is 4 levels of
inheritance away then things get confusing. Mixins cause a similar problem,
namely you can't easily tell where some piece of functionality came. This
generally results in developers traversing through lots of files.

Decorators on the other hand are very visible. It is clear where
functionality is coming from. You can layer decorators on top of one another
and mix/match them too!
```

Decorators on the other hand are very visible

다중 상속 다루기

1. 인터페이스 상속과 구현 상속을 구분한다.
2. ABC를 이용해서 인터페이스를 명확히 한다.
3. 코드를 재사용하기 위해 믹스인을 사용한다.
4. 이름을 통해 믹스인임을 명확히 한다.
5. ABC가 믹스인이 될 수는 있지만, 믹스인이라고 해서 ABC인 것은 아니다.
6. 두 개 이상의 구상클래스에서 상속받지 않는다.
7. 사용자에게 집합 클래스를 제공한다.
8. 클래스 상속보다 객체 구성을 사용하라.

> **다중 상속의위협과 유해성이 지나치게 확대 포장되어 있다. 사실 나는 다중 상속과 관련해서 그다지 큰 문제를 겪지 않았다.**
>
> by 레나르트 레제브로 님 ; in 전문가를 위한 파이썬, 한빛미디어

ABC란?

Python ABC(Abstract Base Class) 추상화 클래스

```
class BaseClass:
    def function_1(self):
        raise NotImplementedError()

    def function_2(self):
        raise NotImplementedError()
```

```
import abc

class BaseClass:
    __metaclass__ = abc.ABCMeta

    @abc.abstractmethod
    def function_1(self):
        pass

    @abc.abstractmethod
    def function_2(self):
        pass
```

1. 인스턴스화 할 수 없다.

```
>>> from BaseClass import BaseClass
>>>
>>> base = BaseClass()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: Can't instantiate abstract class BaseClass with abstract
methods function_1, function_2
```

2. 에러 발생 시점이 다르다.

```
function in TestClass!!
Traceback (most recent call last):
    ...
    raise NotImplementedError()
NotImplementedError
```

```
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    _m = getattr(__import__(cls_name, fromlist=[]), cls_name)()
TypeError: Can't instantiate abstract class TestClass with abstract
methods function_2
```