**Тема занятия № 47: Модуль 29. Миксины в Django**

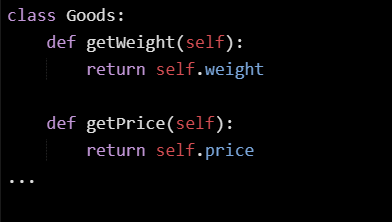
1. Миксины

Вообще, миксины были придуманы для возможности единообразного оперирования объектами. Представьте, что разрабатывается интернет-магазин и для каждого товара система предполагает получение следующих стандартных свойств:

* идентификатор;
* габариты;
* вес;
* цена.

Для материальных товаров все эти характеристики имеют смысл.

Но для информационных, таких как электронная книга или приложение для смартфона, габариты и вес не определены. Тем не менее, система магазина в целом, обращается к этим свойствам объектов (товаров). Как сделать так, чтобы каждый объект, представляющий товар, имел по умолчанию все нужные свойства? Конечно, мы можем непосредственно в классе их прописать:

****

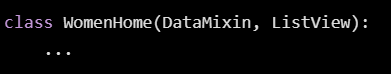
Но это начинает плохо работать, при наличии разветвленной иерархии объектов, когда каждый класс товара представляется отдельным классом:

Cars, Toys, Books, …

Тогда пришлось бы в каждом прописывать эти методы, что нехорошо. Поэтому, как вы уже догадались, все это выносится в базовый класс, например, Goods. Но, иногда, наборы разрозненных классов не имеют единого базового, либо базовый класс находится на таком уровне абстракции, что в него писать такие конкретные методы – прямой путь к мешанине кода. Вот как раз для таких случаев, когда нужно дополнительно к уже существующей иерархии добавить какие-либо общие для разнородных классов данные и/или методы и применяется механизм миксинов.

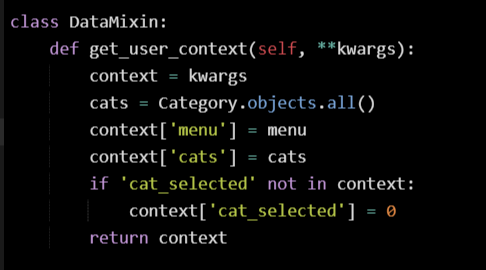
В разных языках программирования миксины реализуются по-разному. В частности, в Python, благодаря наличию механизма множественного наследования, примеси можно добавлять в виде отдельного базового класса.

Например, наши классы представлений можно дополнительно унаследовать от класса DataMixin, который мы отдельно определим. Причем, этот класс лучше прописывать первым в списке наследования:

****

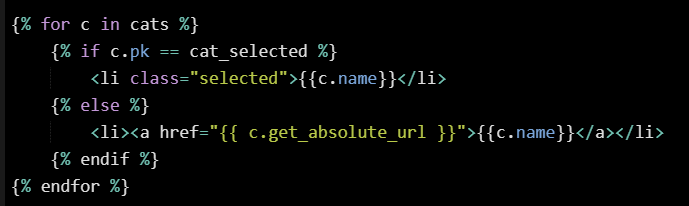
Так как в нем могут быть атрибуты и методы, которые, затем, используются конструктором следующего класса – ListView. То есть, в Python, класс, записанный первым, первым и обрабатывается. Поэтому данные базового класса DataMixin переопределят (при необходимости) атрибуты следующего класса ListView.

Давайте теперь определим наш класс DataMixin и уберем дублирование кода из классов представлений. Первый вопрос, где прописать этот класс? Обычно, в Django все дополнительные, вспомогательные классы объявляют в отдельном файле приложения utils.py. Мы так и поступим. Создадим этот файл и в нем запишем класс DataMixin, следующим образом:

****

Обратите внимание, я перенес сюда и главное меню, т.к. оно используется напрямую классом DataMixin. И вначале идет импорт моделей, т.к. мы используем класс Category для получения всех категорий.

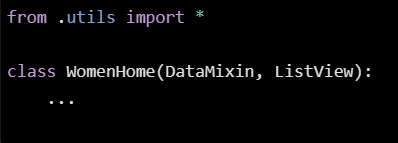
Если вы помните, мы категории в шаблоне base.html сейчас отображаем с помощью созданного нами тега show\_categories. Это был искусственный пример, демонстрирующий возможность создания пользовательских тегов, теперь я его уберу и вместо него буду использовать переменную cats, которую передадим в шаблон. Соответственно, в шаблоне вернем строчки для отображения рубрик:

****

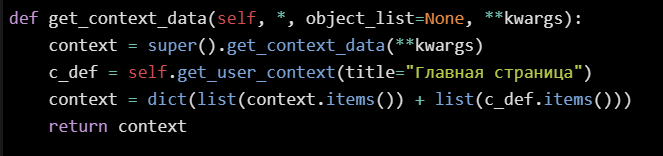
Итак, что же делает класс DataMixin? Смотрите, в нем объявлен вспомогательный метод get\_user\_context() для формирования контекста шаблона по умолчанию. Также, при необходимости, мы можем передавать ему именованные аргументы, которые также будут помещаться в контекст. Благодаря этому методу, нам не придется в классах представлений каждый раз прописывать ссылки на главное меню и категории.

Итак, класс миксин объявлен, осталось прописать его в качестве базового у классов представлений. Для этого в файле views.py мы импортируем модуль utils.py:

И унаследуем класс WomenHome также и от DataMixin:

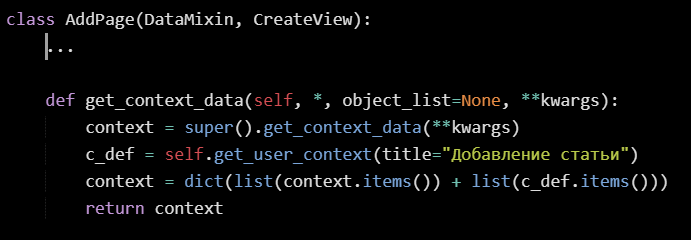
****

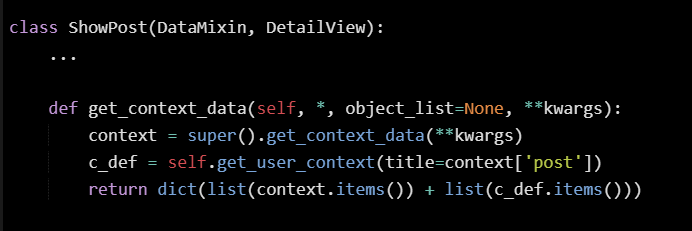
Осталось изменить метод get\_context\_data(), следующим образом:

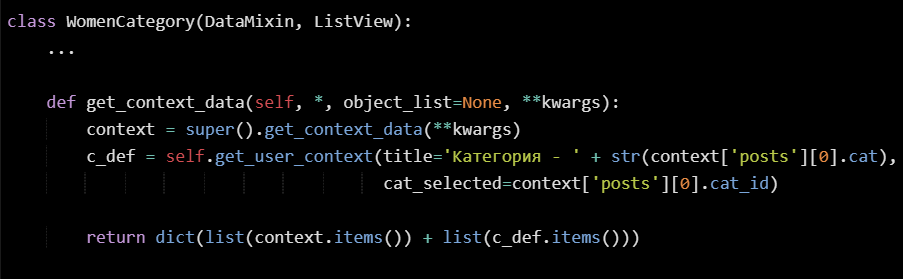


Смотрите, мы здесь вызываем метод get\_user\_context() класса DataMixin, указав, дополнительно параметр title. Получаем сформированный словарь c\_def со всеми стандартными ключами и объединяем его со словарем context. В конце, возвращаем объединенные данные. Все, дублирование в методе get\_context\_data() устранено.

По аналогии, меняем и все остальные классы представлений:



****

****

Все, переходим на сайт и видим, что страницы отображаются также как и ранее, но теперь все работает совместно с классом DataMixin. Вот пример того, как миксины в Django позволяют устранять дублирование кода в классах представлений.

Конечно, в класс DataMixin можно прописывать не только методы, но и общие атрибуты, если они есть, то есть, выносить любую общую информацию.

Миксины фреймворка Django

В Django есть стандартные миксины, которые можно использовать совместно с классами представлений. Использовать их достаточно просто, я покажу пример одного такого миксина:

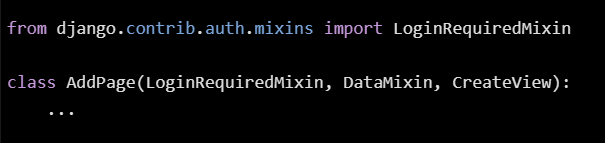
LoginRequiredMixin

который позволяет ограничить доступ к странице для неавторизованных пользователей. Подробную информацию об этом классе можно посмотреть по этой ссылке:

https://djbook.ru/rel3.0/topics/auth/default.html

Давайте вначале выполним его импорт в файле views.py:

А, затем, добавим в класс AddPage:

****

Причем, прописывать желательно самым первым, т.к. он имеет наибольшую важность. Хотя, в нашем случае первые два миксина можно записывать в любом порядке, они никак между собой не пересекаются.

По идее все. Если теперь попробовать выйти из админки (то есть, стать не зарегистрированным пользователем) и перейти на добавление поста, то увидим страницу с кодом 404. Давайте улучшим этот поведение, сделаем его более дружественным. Для этого, в классе AddPage (после добавления миксина LoginRequiredMixin) можно прописать специальный атрибут:



который указывает адрес перенаправления для незарегистрированного пользователя. В данном случае, мы его отправляем в админ-панель. Переходим на главную страницу, затем, на добавление статьи и автоматом перенаправляемся на форму авторизации.

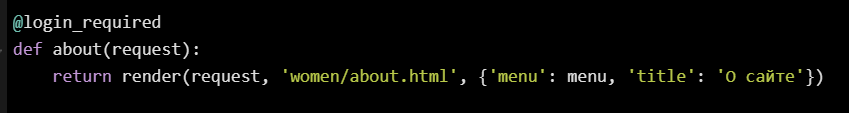
Конечно, прописывать конкретный URL-адрес – это не лучшая практика, поэтому, давайте, воспользуемся функцией reverse\_lazy для формирования маршрута по его имени:



Также, вместо перенаправлений, можно генерировать страницу с кодом 403 – доступ запрещен. Для этого достаточно прописать атрибут:

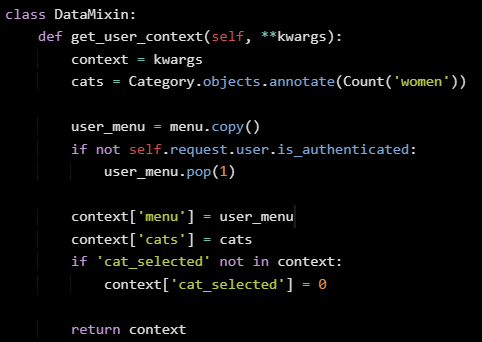


Если похожий функционал нужно реализовать для функций представлений, а не классов, то здесь уже используется декоратор login\_required, например, так:

****

Теперь эта страница доступна только авторизованным пользователям. Я уберу его, т.к. он здесь не к месту. Это просто демонстрация того, как ограничить доступ, работая с функциями представлениями.

Наконец, давайте сделаем отображение пункта «Добавить статью» только для авторизованных пользователей. Для этого я в классе DataMixin буду удалять этот пункт, если пользователь не авторизован:

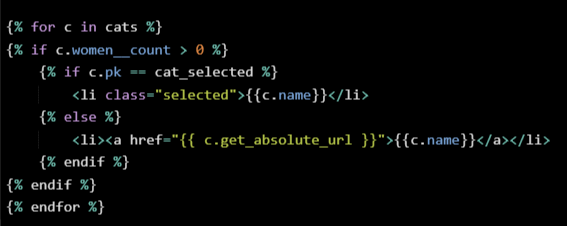
****

Здесь используется объект request, у которого имеется объект user, а у того, в свою очередь, специальный булевый атрибут is\_authenticated, указывающий на авторизацию текущего пользователя (если True – авторизован, False – в противном случае).

Ну и в заключение этого занятия, давайте сделаем вывод только тех рубрик, которые содержат статьи. Для этого мы будем выбирать рубрики с использованием агрегирующей функции:



Как работает эта строчка мы говорили на предыдущем занятии. Далее, в шаблоне base.html пропишем проверку при выводе рубрик:

****

Все, теперь у нас появляются только те рубрики, у которых есть статьи, что более логично.