Представления и шаблоны

Метод **_str_** выводит человекочитаемое название объекта.

```python
# Импортируем класс, который говорит нам о том,
# что в этом представлении мы будем выводить список объектов из БД
from django.views.generic import ListView
from .models import Product


class ProductsList(ListView):
    # Указываем модель, объекты которой мы будем выводить
    model = Product
    # Поле, которое будет использоваться для сортировки объектов
    ordering = 'name'
    # Указываем имя шаблона, в котором будут все инструкции о том,
    # как именно пользователю должны быть показаны наши объекты
    template_name = 'products.html'
    # Это имя списка, в котором будут лежать все объекты.
    # Его надо указать, чтобы обратиться к списку объектов в html-шаблоне.
    context_object_name = 'products'
```

Вот так мы можем использовать дженерик ListView для вывода списка товаров:

1. Создаем свой класс, который наследуется от ListView.
2. Указываем модель, из которой будем выводить данные.
3. Указываем поле сортировки данных модели (необязательно).
4. Записываем название шаблона.
5. Объявляем, как хотим назвать переменную в шаблоне.

```
path('', ProductsList.as_view()),
```

Добавление в urlpatterns представления, созданного с помощью дженерика.

```
path('<int:pk>', ProductDetail.as_view()),
```

Подключение представления, созданного на основе DetailView.

```python
class ProductDetail(DetailView):
    # Модель всё та же, но мы хотим получать информацию по отдельному товару
    model = Product
    # Используем другой шаблон — product.html
    template_name = 'product.html'
    # Название объекта, в котором будет выбранный пользователем продукт
    context_object_name = 'product'
```

Возьмите за привычку после написания **{% if %}** сразу писать **{% endif %}**. Написали **{% for %}** — сразу напишите **{% endfor %}** и так далее.

Писать фильтры и теги в разных файлах необязательно, но при разрастании кодовой базы проекта это может упростить вашу работу.