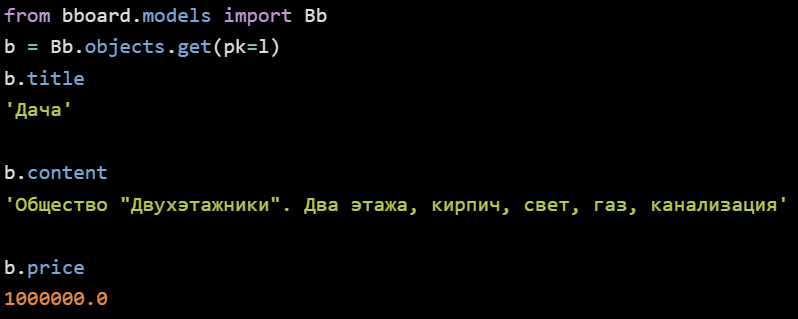
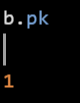
**Тема занятия № 11: Модуль 7. Выборка данных**

1. Извлечение значений из полей записи

Извлечение значений из полей записи получить значения полей записи можно из атрибутов класса модели, представляющих эти поля:

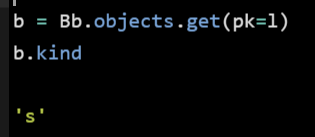


Атрибут класса рк хранит значение ключа текущей записи:

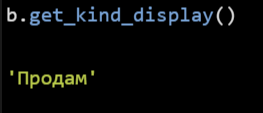


Им удобно пользоваться, когда в модели есть явно созданное ключевое поле с именем, отличным от стандартного id, — нам не придется вспоминать, как называется это поле.

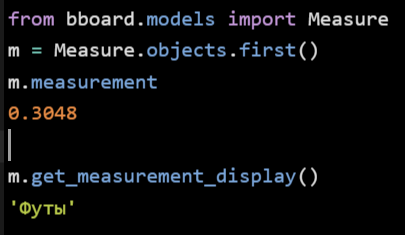
Получение значений из полей со списком если просто обратиться к какому-либо полю со списком, мы получим значение, которое непосредственно хранится в поле, а не то, которое должно выводиться на экран:



Чтобы получить ’’экранное” значение, следует вызвать у модели метод с именем вида get\_<имя\_поля>\_dispiay (), который это значение и вернет:



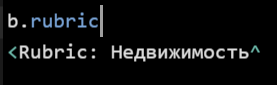
Это же касается полей, у которых перечень допустимых значений задан в виде объекта последовательности:



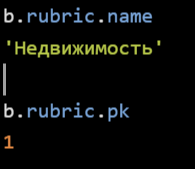
1. Доступ к связанным записям

Доступ к связанным записям средства, предназначенные для доступа к связанным записям и создаваемые самим фреймворком, различаются для разных типов связей.

Для связи ”один-со-многими" из вторичной модели можно получить связанную запись первичной модели посредством атрибута класса, представляющего поле внешнего ключа:



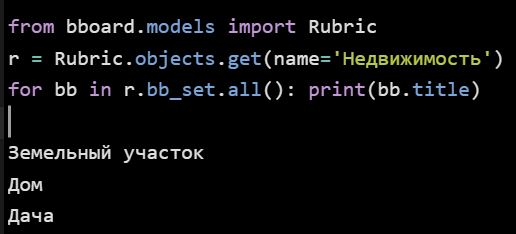
Можно получить значение любого поля связанной записи:



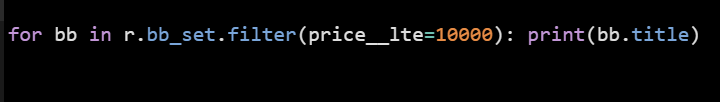
В классе первичной модели будет создан атрибут с именем вида <имя связанной вторичной модели>\_Bb^ Он хранит диспетчер обратной связи, представленный экземпляром класса relatedmanager, который является производным от класса диспетчера записей Manager и, таким образом, поддерживает все его методы.

Диспетчер обратной связи, в отличие от диспетчера записей, манипулирует только записями, связанными с текущей записью первичной модели.

Посмотрим, чем торгуют в рубрике ’’Недвижимость”:

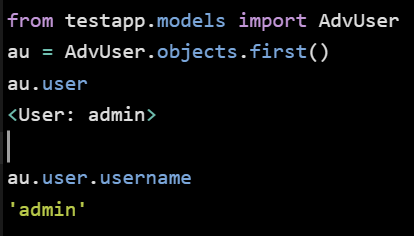


Выясним, есть ли там что-нибудь дешевле 10 000 руб.:

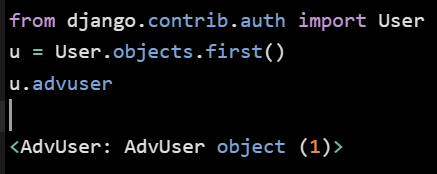


Похоже, что ничего...

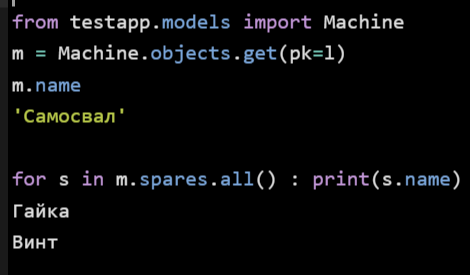
В случае связи "один-с-одним" всё гораздо проще. Из вторичной модели можно получить доступ к связанной записи первичной модели через атрибут класса, представляющий поле внешнего ключа:

****

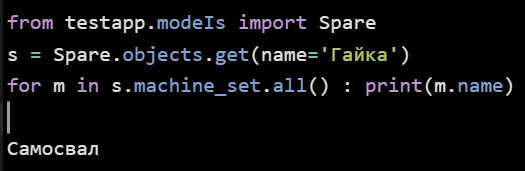
Из первичной модели можно получить доступ к связанной записи вторичной модели через атрибут класса, имя которого совпадает с именем вторичной модели:



В случае связи "многие-со-многими" через атрибут класса ведущей модели, представляющий поле внешнего ключа, доступен диспетчер обратной связи, представляющий набор связанных записей ведомой модели:



В ведомой модели будет присутствовать атрибут класса <имя связанной ведущей модели>\_set. Его можно использовать для доступа к записям связанной ведущей модели. Пример:

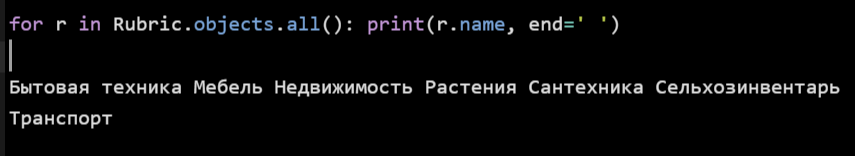
****

1. Выборка записей

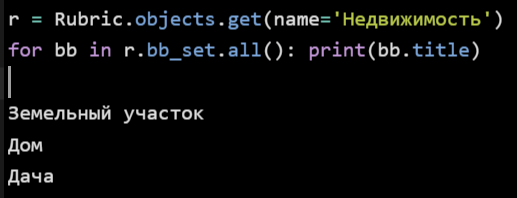
Теперь выясним, как выполнить выборку из модели записей — как всех, так и лишь тех, которые удовлетворяют определенным условиям.

Все модели поддерживают атрибут класса objects. Он хранит диспетчер записей (экземпляр класса Manager), который позволяет манипулировать всеми записями, хранящимися в модели.

Метод ан (), поддерживаемый классом Manager, возвращает набор из всех записей модели в виде экземпляра класса Queryset. Последний обладает функциональностью последовательности и поддерживает итерационный протокол. Так что мы можем просто перебрать записи набора и выполнить над ними какие-либо действия в обычном цикле for.. .in. Пример:



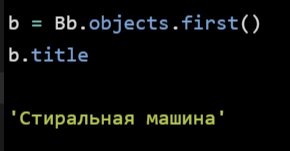
Класс relatedmanager является производным от класса Manager, следовательно, тоже поддерживает метод all(). Только в этом случае возвращаемый им набор будет содержать лишь связанные записи. Пример:



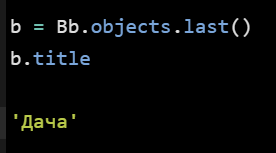
**Извлечение одной записи**

Ряд методов позволяют извлечь из модели всего одну запись:

□ first () — возвращает первую запись набора или None, если набор пуст:

****

□ last () — возвращает последнюю запись набора или None, если набор пуст:



Оба эти метода учитывают сортировку набора записей, заданную либо вызовом метода order by (), либо в параметре ordering модели;

□ earliest ([<имя поля 1>, <имя поля 2> . . . <имя поля л>]) —возвращает запись, у которой значение даты и времени, записанное в полях с указанными именами, является наиболее ранним.

Предварительно выполняется временная сортировка записей по указанным полям. По умолчанию записи сортируются по возрастанию значений этих полей.

Чтобы задать сортировку по убыванию, имя поля нужно предварить символом ’’Минус”.

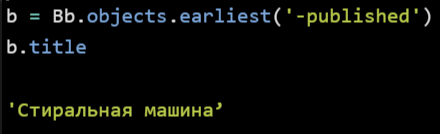
Сначала проверяется значение, записанное в поле, имя которого указано первым. Если это значение одинаково у нескольких записей, то проверяется значение следующего поля и т.д.

Если в модели указан параметр get iatest by, задающий поля для просмотра, то метод можно вызвать без параметров. Если ни одной подходящей записи не нашлось, возбуждается исключение Doesnotexist.

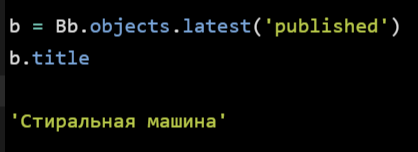
Ищем самое раннее из оставленных на сайте объявлений:



А теперь найдем самое позднее, для чего укажем сортировку по убыванию:

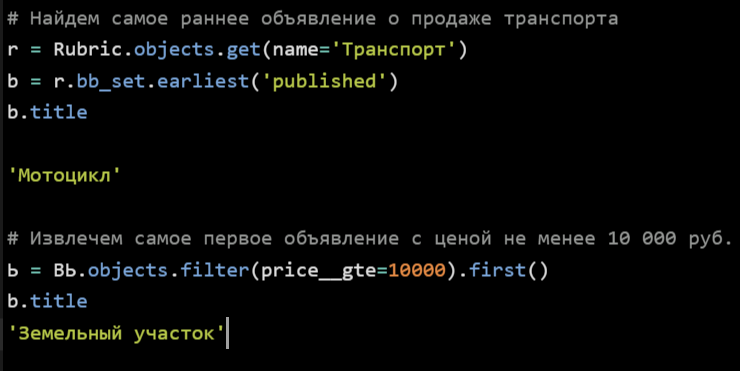


□ latest ([<имя поля 1>, <имя поля 2> . . . <имя поля п>]) — ТО же самое, ЧТО Earliest (), но ищет запись с наиболее поздним значением даты и времени:



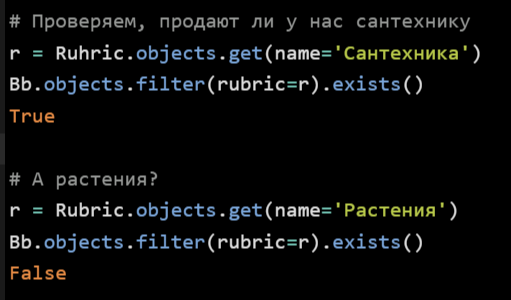
Все ЭТИ методы поддерживаются классами Manager, relatedmanager И queryset.

Следовательно, мы можем вызывать их также у набора связанных записей:

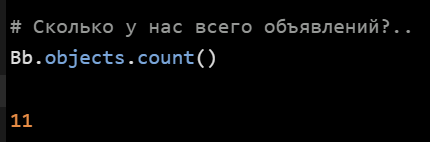


Получение числа записей в наборе два следующих метода позволят получить число записей, имеющихся в наборе, а также проверить, есть ли там записи:

□ exists () — возвращает True, если в наборе есть записи, и False, если набор записей пуст:



□ count () — возвращает число записей, имеющихся в наборе:



Эти методы выполняются очень быстро, поэтому для проведения простых проверок рекомендуется применять именно их.

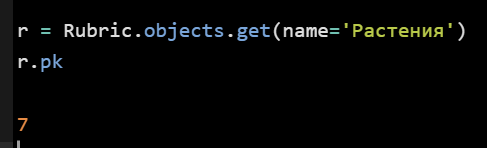
**Поиск одной записи**

Для поиска записи ПО заданным условиям служит метод get (<условия поиска>). Условия поиска записываются в виде именованных параметров, каждый из которых представляет одноименное поле. Значение, присвоенное такому параметру, задает искомое значение для поля.

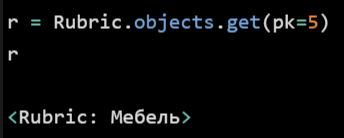
Если совпадающая с заданными условиями запись нашлась, она будет возвращена в качестве результата. Если ни одной подходящей записи не было найдено, то будет возбуждено исключение doesnotexist, класс которого является вложенным в класс модели, чья запись не была найдена. Если же подходящих записей оказалось несколько, возбуждается исключение multipleobjectsreturned ИЗ МОДУЛЯ Dj ango.core.exceptions.

Пара типичных примеров:

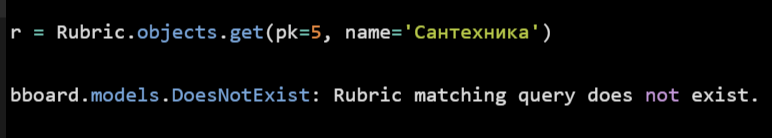
□ найдем рубрику ’’Растения”:



□ найдем рубрику с ключом 5:



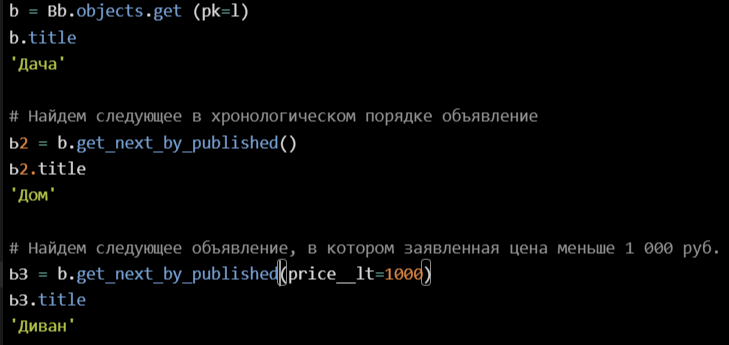
Если в методе get () указать сразу несколько условий поиска, то они будут объединяться по правилам логического И. Для примера найдем рубрику с ключом 5 И названием ’’Сантехника”:



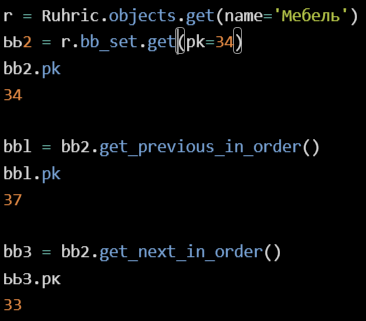
Такой записи нет, и мы получим исключение doesnotexist.

Если В модели есть ХОТЯ бы ОДНО поле типа datefieid ИЛИ datetimefield, ТО модель получает поддержку методов С именами вида get\_next\_by\_<MM>j поля>() и Get\_previous\_by\_<HMH поля> (). Формат вызова у обоих методов одинаковый:

Get\_next\_by\_<WM# поля> | get\_previous\_by\_<Hzvw поля> ( [ <условия поиска>] ) первый метод возвращает запись, чье поле с указанным именем хранит следующее в порядке увеличения значение даты, второй метод — запись с предыдущим значением. Если указаны условия поиска, то они также принимаются во внимание. Примеры:



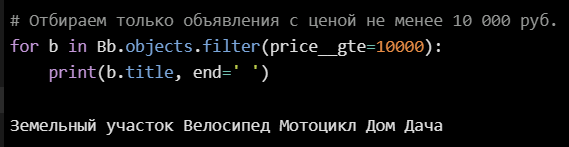
Если текущая модель является вторичной, и у нее было задано произвольное переупорядочивание записей, связанных с одной и той же записью первичной модели (т.е. Был указан параметр order\_with\_respect\_to). То эта вторичная модель получает поддержку методов get\_next\_in\_order () и get\_Previous in order о. Они вызываются у какой-либо записи вторичной модели: первый метод возвращает следующую в установленном порядке запись, а второй — предыдущую. Пример:



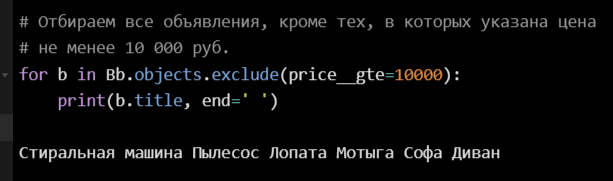
**Фильтрация записей**

Для фильтрации записей Django предусматривает два следующих метода — диаметральные противоположности друг друга:

□ filter{<условия фкльтрации>) — отбирает из текущего набора только записи, удовлетворяющие заданным условиям фильтрации. Условия фильтрации задаются точно в таком же формате, что и условия поиска в вызове метода get о

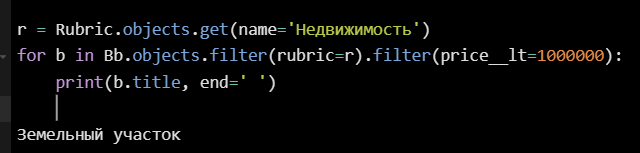
****

□ exclude (<условия фильтрации>) — то же самое, что filter о, но, наоборот, отбирает записи, не удовлетворяющие заданным условиям фильтрации:

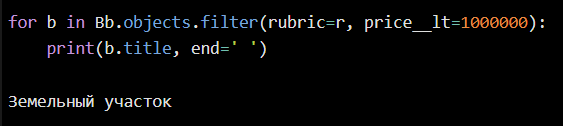


Поскольку оба эти метода поддерживаются классом Queryset, мы можем "сцеплять"

Их вызовы друг с другом. Для примера найдем все объявления о продаже недвижимости с ценой менее 1 000 000 руб.:

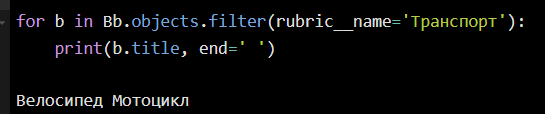


Впрочем, такой запрос на фильтрацию можно записать и в одном вызове метода Filter():

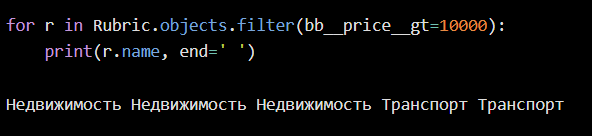


Фильтрация по значениям полей связанных записей

Чтобы выполнить фильтрацию записей вторичной модели по значениям полей из первичной модели, условие фильтрации записывается в формате <имя поля внешнего ключа>\_\_<имя поля первичной модели>. Для примера выберем все объявления о продаже транспорта:

****

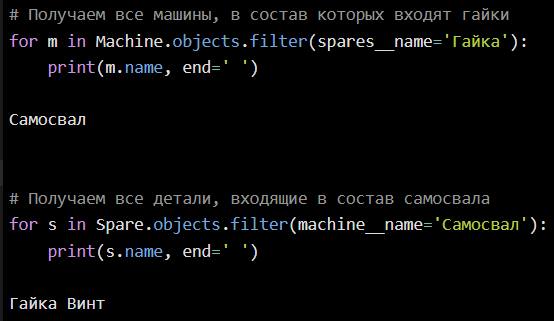
Для выполнения фильтрации записей первичной модели по значениям из полей вторичной модели следует записать условие вида <имя вторичной модели> <имя поля вторичной модели>. В качестве примера выберем все рубрики, в которых есть объявления о продаже с заявленной ценой более 10 000 руб.:

****

Мы получили три одинаковые записи ’’Недвижимость” и две— ’’Транспорт”, поскольку в этих рубриках хранятся, соответственно, три и два объявления, удовлетворяющие заявленным условиям фильтрации. О способе выводить только уникальные записи мы узнаем позже.

Как видим, в подобного рода условиях фильтрации мы можем применять и модификаторы.

Все это касалось связей "один-со-многими" и "один-с-одним". А что же связи ”многие-со-многими”? В них действуют те же самые правила. Убедимся сами:

****

Сравнение co значениями других полей До этого момента при написании условий фильтрации мы сравнивали значения, хранящиеся в полях, с константами. Но иногда бывает необходимо сравнить значение из одного поля записи со значением другого ее поля.

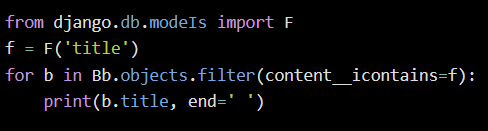
Для этого предназначен класс с именем F, объявленный в модуле django.db.models.

Вот формат его конструктора:

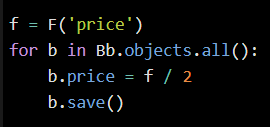
F (<имя поля модели, с которым должно выполняться сравнение^имя поля модели записывается в виде строки.

Получив экземпляр этого класса, мы можем использовать его в правой части любого условия.

Вот пример извлечения объявлений, в которых название товара встречается в тексте его описания:

****

Экземпляры класса f можно использовать не только при фильтрации, но и для занесения нового значения в поля модели. Например, так можно уменьшить цены во всех объявлениях вдвое:

****

**Сложные условия фильтрации**

Класс q из модуля django.db.models позволяет создавать более сложные условия фильтрации. Его конструктор записывается в следующем формате:

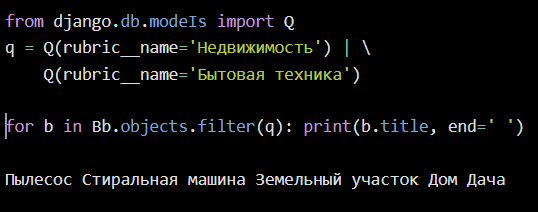
Q (<условие фильтрации>)

Условие фильтрации, одно-единственнос, записывается в таком же виде, как и в вызовах методов filter () И exclude ().

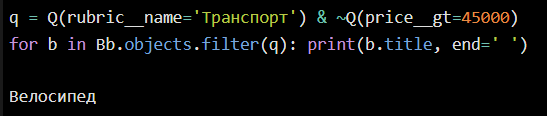
Два экземпляра класса Q, хранящие разные условия, можно объединять посредством операторов & и |, которые обозначают соответственно логическое И и ИЛИ.

Для выполнения логического НЕ применяется оператор Все эти три оператора в качестве результата возвращают новый экземпляр класса Q.

Пример выборки объявлений о продаже ИЛИ недвижимости, ИЛИ бытовой техники:

****

Пример выборки объявлений о продаже транспорта, в которых цена НЕ больше 45 000 руб.:

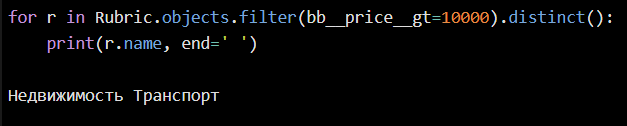
****

**Выборка уникальных записей**

Для вывода только уникальных записей служит метод distinct ():

Distinct ( [<имя поля 1>, <имя поля 2> . . . <имя поля л>]) при использовании СУБД postgresql в вызове метода можно перечислить имена полей, значения которых определят уникальность записей. Если не задавать параметров, то уникальность каждой записи будет определяться значениями всех ее полей.

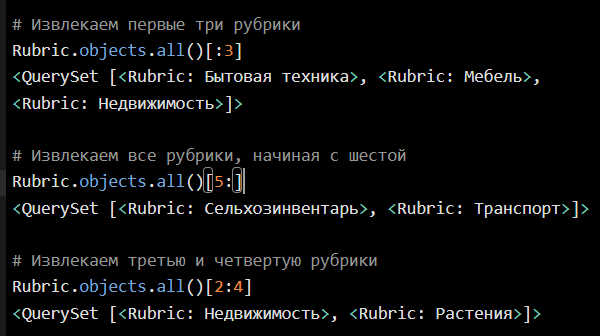
Перепишем пример чтобы он выводил только уникальные записи:



**Выборка указанного числа записей**

Для извлечения указанного числа записей применяется оператор взятия среза [] Python, записываемый точно так же, как и в случае использования обычных последовательностей. Единственное исключение— не поддерживаются отрицательные индексы.

Вот три примера:

****

1. Сортировка записей

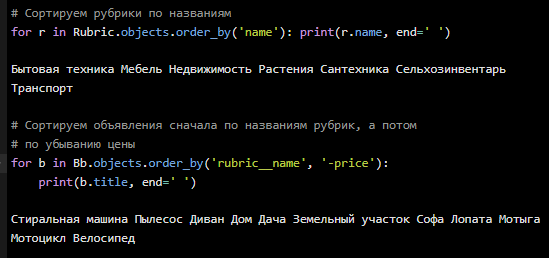
Сортировка записей

Для сортировки записей в наборе применяется метод order by ():

Order\_by( [<имя поля 1>, <имя поля 2> . . . <имя поля п>] )

В качестве параметров указываются имена полей в виде строк. Сначала сортировка выполняется по значению первого поля. Если у каких-то записей оно хранит одно и то же значение, проводится сортировка по второму полю и т.д. По умолчанию сортировка выполняется по возрастанию значения поля. Чтобы отсортировать по убыванию значения, следует предварить имя поля знаком "минус".

Пара примеров:

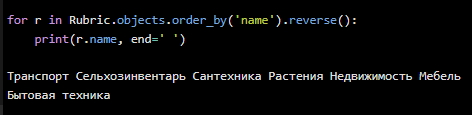
****

Каждый вызов метода order by () отменяет параметры сортировки, заданные в его предыдущем вызове или в параметрах модели (атрибут ordering вложенного класса Meta). Поэтому, если записать:

**Bb.obj ects.order\_by(1 rubric\_\_name1).order\_by(1-price1)**

Объявления будут отсортированы только по убыванию цены. Если передать методу order by о в качестве единственного параметра строку ’?’, то записи будут выстроены в случайном порядке. Однако это может отнять много времени.

Вызов метода reverse () меняет порядок сортировки записей на противоположный:



Транспорт Сельхозинвентарь Сантехника Растения Недвижимость Мебель Бытовая техника

Чтобы отменить сортировку, заданную предыдущим вызовом метода order byo или в самой модели, следует вызвать метод order\_by () без параметров.