ДЗ#3: Формы

Основы веб-разработки (первый семестр)

Здесь - ликбез по формам ввода в html и django, в том объеме, в котором оно надо для подготовки третьего ДЗ

Формы в html

Пользовательский ввод с точки зрения html обслуживает две задачи
* предоставление пользователю удобного интерфейса для ввода информации
* структурирование введенных данных по определенным правилам и пересылка их на сервер в определенном формате

Элементарная работа с вводом данных обычно такова - показываем пользователю форму ввода, а внутри нее - некоторое количество именованных полей для ввода и кнопку submit. Пользователь вводит данные, жмет submit ("Отправить", "Найти", "Создать пост" итп), данные отправляются на сервер в определенном формате, сервер их обрабатывает и возвращает ответ.

По шагам - вот так:
1) Пользователь зашел на страницу, в ней может быть несколько форм
2) Пользователь ввел данные, и жмакнул кнопку submit-а
3) Браузер смотрит, в какую именно форму входит нажатая кнопка submit-а
4) Браузер ищет в этой форме все элементы ввода, забирает из них введенные пользователем данные
5) Для каждого элемента ввода браузер знает из html, под каким именем данные этого элемента должны быть отправлены на сервер (это определяется аттрибутами элемента ввода)
6) Браузер проверяет, каким именно образом нужно отправить собранные данные на сервер и на какую ссылку (это определяется аттрибутами формы)
7) Происходит запрос на сервер

Принципиально можно разделить ввод данных на два типа запросов:

GET-запрос

В этом случае браузер передает данные прямо в ссылке, в той ее части, которая называется querystring.
В случае работы с GET-запросами сервер обычно сразу возвращает html, который браузер и показывает пользователю, плюс ту же самую форму ввода, заполненную теми данными, которые пользователь вводил. Ближайший пример - поисковики. Тот же гугл, к примеру. Изначально мы видим поле ввода, рядом кнопку "Найти". Оба они входят в одну html-форму. Мы что-то вводим, жмакаем "Найти", браузер формирует урл типа google.com/?q=nanana и показывает нам ту

Прямой эфир

Марина Даньшина 17 минут назад
Расписание занятий → Расписание пересдач 0

Екатерина Рогушкова 8 часов назад
Получение значков достижений на портале (ачивки) 2

Ольга Августан Вчера в 18:41
Опросы → Получение значков достижений на портале (ачивки) 2

Ольга Августан Вчера в 13:19
Общие вопросы → Завершение семестра: последние итоговые встречи 0

Ольга Августан Вчера в 13:12
Завершение семестра: итоговые встречи по дисциплинам 10

Марина Даньшина 25 Мая 2016, 14:01
Разработка на Java (первый семестр) → Забыли зонтик 0

Ksenia Sternina 25 Мая 2016, 13:38
Домашнее задание - Юзабилити-тестирование 4

Летяго Владимир 25 Мая 2016, 01:40
Разработка приложений на Android - 1 (второй семестр) → Контрольный рубеж 0

Летяго Владимир 24 Мая 2016, 22:45
Домашнее задание №3 10

Вячеслав Ишутин 24 Мая 2016, 22:31
Разработка на C++ (открытый курс) → Итоговая встреча по дисциплине 0

Вячеслав Ишутин 24 Мая 2016, 22:22
Разработка на C++ (открытый курс) → Итоговое занятие. Презентация курсовых проектов. 0

Александр Агафонов 24 Мая 2016, 19:17
Мастер-класс Ильи Стыценко онлайн «Использование OAuth 2 в приложениях на Django» 1

Марина Даньшина 23 Мая 2016, 20:59
Мероприятия → Мастер-класс Ильи Стыценко онлайн «Использование OAuth 2 в приложениях на Django» 1

Ольга Августан 23 Мая 2016, 17:56
Разработка приложений на iOS - 1 (второй

же самую форму, где поле поискового запроса уже сразу заполнено введенным нами "nanana" + то, что нашлось по этому запросу.

POST-запрос

В этом случае браузер передает данные в теле запроса.
Случай работы с POST-запросами можно разделить на несколько этапов, рассмотрим их на примере создания например поста на условном хабре.

* Пользователь зашел (GET-запрос) на страничку создания поста.
* Сервер видит, что запрос пришел методом GET, а значит нужно просто отрисовать пустую формочку. Возвращает html с формой, полями "Заголовок" и "Текст" и кнопкой submit.
* Пользователь вводит все данные и жмакает "Создать" (submit, POST-запрос)
* Запрос уходит обычно на ту же страничку создания поста, но на этот раз методом POST. Сервер видит это и проверяет, правильно ли введены данные.
** Если данные не верны (например, забыли ввести заголовок) - сервер возвращает html, где отрисовывается та же самая форма с теми же полями, и сообщением об ошибке ("Введите заголовок"). Все правильно введенные поля должны быть уже предзаполнены (чтобы юзеру не пришлось заново вводить текст поста)
** Если данные верны, сервер возвращает ему http-ответ с перенаправлением на страницу только что созданного поста.

По последнему пункту - перенаправление делается для исключения дубликатов. Потому что, если браузер получает сразу html после POST-запроса, а пользователь жмакнет F5 (обновить страницу) - то серверу прилетит опять POST-запрос с теми же данными. В случае перенаправления, браузер приходит на конечную ссылку уже GET-запросом, и можно сколько угодно обновлять страницу.

Итак, для работы с вводом данных в html используются следующие теги:

Тег form

На странице по дефолту данный тег никак отдельно не отрисовывается, он служит только для группировки нескольких элементов ввода и кнопки submit в одну форму.
Аттрибуты:

action

Ссылка, на которую нужно переслать данные, отправленные с помощью этой формы. В случае отсутствия аттрибута - данные отправляются на ту же ссылку, где форма находится изначально (обычно так и делается)

method

Либо get либо post. Определяет, GET или POST запрос будет отправлен на сервер

enctype

семестр) → Завершение семестра 0

Ольга Августан 23 Мая 2016, 17:16
Итоговая встреча по дисциплине 2

Ольга Августан 23 Мая 2016, 17:07
Основы веб-разработки (первый семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 17:06
Основы мобильной разработки (первый семестр) → Итоговая встреча по дисциплине 2

Ольга Августан 23 Мая 2016, 17:05
Проектирование интерфейсов мобильных приложений (второй семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 17:02
Проектирование СУБД (второй семестр) → Итоговая встреча по дисциплине 0

Ольга Августан 23 Мая 2016, 16:59
Общие вопросы → Завершение семестра: итоговые встречи по дисциплинам 10

Весь эфир

Блоги

Основы веб-разработки (первый семестр) 26,04
Разработка на C++ (открытый курс) 16,97
Основы мобильной разработки (первый семестр) 14,70
Разработка на Java (первый семестр) 13,58
Общие вопросы 5,71
Разработка приложений на Android - 1 (второй семестр) 5,66
Стажировка 4,52
Проектирование интерфейсов мобильных приложений (второй семестр) 3,42
Разработка приложений на iOS - 1 (второй семестр) 2,29
Проектирование СУБД (второй семестр) 2,26

Определяет способ кодирования данных перед отправкой на сервер. Нас здесь интересует только один аспект - если в форме есть элемент загрузки файла (например, юзер загружает свою аватарку или картинку к посту) - enctype должен быть равен multipart/form-data, в остальных же случаях можно его не указывать

Примеры:

<form>...</form> - всё по дефолту. будет произведен GET-запрос на ту же страницу, где мы сейчас
<form action="/search/">...</form> - будет произведен GET-запрос на страницу /search/
<form method="post">...</form> - будет произведен POST-запрос туда же, где мы сейчас
<form method="post" enctype="multipart/form-data">...</form> - будет произведен POST-запрос туда же, где мы сейчас, плюс вместе с данными можно загрузить какие-либо файлы

Тег input

Применяется для отрисовки большинства элементов ввода.
Аттрибуты:

name

Определяет, под каким именем введенные данные отправятся на сервер

value

Определяет, какими именно данными элемент ввода будет заполнен изначально

type

Определяет, а как собственно будет отрисован элемент ввода. Возможные значения смотрите здесь http://htmlbook.ru/html/input/type

required

Определяет, обязательно ли элемент должен быть заполнен для отправки формы. Если такой аттрибут есть у элемента, то браузер не даст отправить формочку на сервер, пока мы не заполним его чем-либо

Примеры:

<input type="text" name="searchstring"> - данные лягут в запрос под именем searchstring, рисуем тупо текстовое поле
<input type="text" name="searchstring" required=""> - то же самое, но мы не сможем отправить форму, пока не заполним это поле ввода чем-нибудь
<input type="text" name="searchstring" required="" value="где у преподавателя кнопка паузы?"> - то же самое, но изначально поле будет заполнено текстом из аттрибута value
<input type="file" name="avatar"> - будет отрисована кнопка "Загрузить файло", сам файл отправится на сервер под именем avatar

<input type="submit" value="Отправить комментарий"> - та самая submit-кнопка для формы, на кнопке будет написано "Отправить комментарий"
<input type="hidden" name="post_id" value="13"> - пользователь вообще не увидит этого поля на странице, но в запрос под именем post_id добавится значение "13"

Имейте ввиду, в случае с input type="file" мы не можем предзаполнить его ничем - security, все дела. Иначе мы могли бы отрисовать страничку, которая изначально заставляла бы юзера загрузить какой-либо файл с компьютера, без его ведома.

Тег select

Используется для отрисовки списков, из которых юзер может что-либо выбрать. Чтобы определить, из каких именно вариантов пользователь что-то выбирает - в тег select вкладывается какое-то количество тегов option
Аттрибуты:

required

Аналогично тегу input

name

Аналогично тегу input

multiple

Определяет, может ли пользователь выбрать несколько вариант, или только один

Тег option

Вкладывается в тег select, означает один конкретный вариант для выбора.
Аттрибуты:

value

Значение, которое отправится на сервер, если будет выбран этот option. Само же текстовое представление варианта вкладывается текстом внутрь тега

selected

При наличии такого аттрибута в теге он будет выбран заранее

Примеры:

```
<select name="category">
   <option value="13">Бизнес</option>
   <option value="15" selected>Авто</option>
</select>
```

Пользователь выберет какую-
либо категорию, и ее id отправится на сервер под именем category. Изначально вы

```
<select multiple name="user">
   <option value="10">Вася</option>
   <option value="11">Петя</option>
   <option value="12">Коля</option>
   <option value="13">Люда</option>
```

</select>
Пользователь выберет одного и других пользователей, и их id отправятся на серве то вроде "user=10&user=13"). Изначально не выбран никто

Тег textarea

Применяется для ввода многострочного текста, поддерживает в том числе аттрибуты required, name. Если нужно заранее заполнить чем-либо данный элемент ввода - содержимое просто вкладывается прямо внутрь тега
Пример приводить не буду, на нём ломается верста технотрека :) Ищите здесь http://htmlbook.ru/html/textarea

Тег label

Позволяет отрисовать кусок текста, при нажатии на который фокус ввода будет перемещен в какой-либо элемент ввода. Для этого элементу ввода дается атрибут id, а тегу label соответствующий ему атрибут for. Проще всего на примере:

<label for="title_input">Введите заголовок:</label>
<input id="title_input" type="text" name="title">
Будет отрисовано поле ввода для заголовка поста, а рядом метка "Введите заголовок". Если жмакнем на эту метку - активируется поле ввода

Где почитать

Заходите на http://htmlbook.ru/ и ищите там прямо по имени элемента или аттрибута - все неплохо расписано

Валидация данных средствами html

html-теги form, input, select и textarea предоставляют некоторые инструменты для валидации данных. Например, select заставляет пользователя выбрать один из заранее определенных вариантов, аттрибут required заставляет обязательно ввести или выбрать что-либо.
НО! Браузер - лишь инструмент. Всегда (ВСЕГДА!) есть возможность отправить на ваш сервер любой http-запрос, с любыми данными внутри. Поэтому, правильность введенных данных всегда должна проверяться на сервере. Немного психану:

ВСЕГДА

ПРОВЕРЯЙТЕ

ДАННЫЕ НА СЕРВЕРЕ

!!!111 АЗАЗАЗА

Об этом следующий раздел.

Проверка введенных данных средствами Django. Формы в django.

В Django есть пакет forms, в котором содержится всё, что касается валидации данных на сервере. Но его возможности этим не ограничиваются - формы в django помимо валидации умеют как минимум еще и отрисовать себя на странице, и запомнить ошибки, которые пользователь допустил во время ввода. Кроме того, django.forms занимается еще одной очень важной задачей - приводит прилетевшие к нам данные в нужные структуры языка python. Дело в том, что все, введенное пользователем, изначально существует именно в текстовом виде. Это свойство протокола http - даже если в формочке было супе-мега-убер-поле-ввода для выбора даты и времени - в запросе все равно будет валяться что-нибудь типа "2016-03-20 15:59" именно в текстовом виде. Или даже если пользователь вводит в форму число через <input type="number"> - все равно веб-сервер получит это число в виде текстового его представления, и перед нами стоит задача превратить его в python-овый integer, и дату и время - в python-овый datetime. Всё это - очень важный и трудоемкий кусок работы, именно поэтому везде, где мы принимаем данные от пользователя - нужно использовать пакет forms. Везде. Допустим, вам кажется, что "а чо, там же просто тупо текста кусок, давайте вытащим напрямую из request.GET (request.POST)". Имейте ввиду - вам только кажется. Вы сэкономите полминуты, если забьете на описание формы в django. Но потом пользователь введет что-нибудь такое, что оно ударит вам по лбу. Обязательно введет. И оно обязательно ударит. И сервер упадет, и вам будет плохо, и вы все равно вставите туда форму.
Ладно, ближе к делу.

Описание формы

Первое, что мы делаем для работы с формами в Django - импортируем сам модуль forms:

```
from django import forms
```

Лучше всего сделать это в отдельном файлике внутри приложения, обычно его называют forms.py. И описывают там формы, которые приложение использует. Затем создаем класс формы, как наследник класса forms.Form

```
class MyCoolForm(forms.Form):
    ...
```

После чего наполняем форму нужными нам полями.
Для этого в пакете forms есть несколько классов, обозначающих работу с каким-либо типом данных (текстовые данные, числа, файлы и тому подобное).

Например, forms.CharField работает с текстовыми данными, forms.IntegerField - с целыми числами, forms.DateTimeField - с датами и временем, и тому подобное.
То есть - класс поля определяет то, с какими данными это поле будет работать. А имя этого поля должно совпадать с именем, под которым данные прилетают к нам от браузера.
Например, если мы ждем из браузера заголовок создаваемого поста под именем title - мы делаем вот что

```
class MyCoolForm(forms.Form):
    title = forms.CharField()
```

Сделали форму, которая ждет от пользователя title и интерпретирует его как прост

Думаю, из примера понятно, как описывается форма.

Дальнейшие действия с классом формы таковы:

Создание экземпляра формы в запросе

Очень просто - инициализируем класс формы, просто вызывая его.
Имеем ввиду, что django.forms.Form дает нам достаточно функционала, чтобы не верстать руками в html все поля ввода для пользователя.
Итак, форму можно инициализировать со следующими параметрами:

data

Если этот параметр отсутствует, форма не пытается валидировать данные вообще. То есть, мы просто используем ее для того, чтобы отрисовать данные в html, который хотим вернуть пользователю (о том, как отрисовать форму в шаблоне я расскажу ниже).
Если же мы передаем этот параметр, то в нем должен лежать либо request.GET либо request.POST - в этом случае форма мы сможем проверить на правильность данные, переданные нами от пользователя либо в querystring (GET-запрос), либо в теле запроса (POST-запрос).
Например:

```
myform = MyCoolForm()
```

Просто создали экземпляр формы, сможем передать его в контекст шаблона и отрисовать все поля ввода, просто вызвав например {{ myform }}

```
myform = MyCoolForm(data=request.GET)
```

Создали экземпляр формы, объявив ему, что пользователь прислал данные в GET-параметрах запроса

```
myform = MyCoolForm(data=request.POST)
```

То же самое, но данные ждем из тела запроса (POST-запрос)

И request.GET и request.POST - это наследники класса QueryDict. QueryDict - это просто словарь, умеющий дополнительно некоторые полезные штуки. Не скажу какие, пока вроде бы не нужно, да и гуглится это.

files

В случае, если мы в html рисовали пользователю формочку, которая позволяет загружать файлы - эти файлы прилетят нам из запроса в request.FILES. А в форму мы должны отправить их параметром files. Пример:

myform = MyCoolForm(data=request.POST, files=request.FILES)
Ждем от пользователя какое-то количество данных+загруженные файлики.

initial

Этот параметр мы передаем в форму, если хотим предзаполнить ее какими-либо значениями. Это нужно в том случае, если мы редактируем какой-либо объект - ведь нам нужно сначала показать, что у него уже заполнено, и дать пользователю изменить эти значения. Например:

myform = MyCoolForm(initial={"title": "Пост о том о сём"})
Создали экземпляр формы для того, чтобы просто показать форму пользователю.
Или например так:
myform = MyCoolForm(initial={"title": mypost.title})
Или так:
myform = MyCoolForm(data=request.POST, initial={"title": mypost.title})

Итак, экземпляр формы мы создали. Что с ним можно делать дальше? Как минимум - показать в шаблоне

Использование формы в шаблоне

Созданную форму можно показать в шаблоне, показать отдельные поля, показать ошибки... Всё можно!
Для примера предположим, что мы передали нашу форму в контекст шаблона в переменной myform, а в самой форме есть только поле title для текстовых данных.

Отрисовка формы целиком

Очень просто:

```
<form>
   {{ myform }}
   <input type="submit" value="Создать пост">
</form>
```

В этом случае у формы вызывается метод __unicode__, который собирает все ее поля в том порядке, в котором они описаны в классе формы, и рисует все нужные теги, включая тег label. Сам же тег form и submit-кнопку мы должны рисовать

самостоятельно. Почему? Потому что к валидации данных сервером это не относится, а значит должно оставаться только в верстке.
Кстати, формы в django можно отрисовывать с помощью разных методик - в виде списка, в виде таблицы и просто в виде серии параграфов (тег p). Для этого служат методы формы as_p, as_table, as_ul. Просто вызываем их в шаблоне:

```
<form>
   {{ myform.as_p }}
   {{ myform.as_table }}
   {{ myform.as_ul }}
</form>
```

Обратите внимание, что теги table и ul вокруг формы нужно рисовать самостоятельно - методы формы рисуют только внутреннюю структуру для соответствующих тегов.
Кроме того, если форма в django была инициализирована с параметром data - в аттрибуте errors у нее будут лежать ошибки валидации данных (если пользователь допустил такие ошибки, не ввел например заголовок или в поле для чисел ввел строку итп).
В этом случае в myform.errors будет лежат словарь, ключами которого будут являться названия полей (либо __all__ если это ошибка валидации формы в целом), а значениями будут списки допущенных ошибок (подробнее об ошибках в форме читаем далее). Например:

```
<form>
   {{ myform.errors }} - рендерит ошибки формы как html-список (тег ul)
   А можно и так:
   {% if myform.errors %}
      {% for field_name, errorlist in myform.errors.items %}
         <div>Ошибки в поле {{ field_name }}: {% for error in errorlist %}{{ error }}
{% endfor %}
      {% endfor %}
   {% endif %}
   {{ myform }}
</form>
```

Отрисовка отдельных полей

В шаблоне можно обращаться к отдельным полям формы просто по имени поля. А у отдельного поля формы в шаблоне есть множество полезных штук для отрисовки:

```
<form>
{{ myform.title }} - нарисовали поле ввода для title
{{ myform.title.label }} - нарисовали тег label, работающий с полем title
```

{{ myform.title.errors }} - вывели ошибки, которые допустил пользователь в поле title (аналогично form.errors, только там валяется список ошибок только для данного поля
</form>

Итерация по полям формы

Форма в django поддерживает итерацию (грубо говоря, может трактоваться как список). В этом случае в качестве элементов списка мы получим поля, как если бы обращались к ним через {{ form.имя-поля }}:

```
{% for field in myform %}
    {{ field.label }}
    {{ field }}
    {{ field.errors }}
{% endfor %}
```

Это пожалуй всё, что касается отрисовки форм в шаблонах.

Валидация данных, получение данных из формы

У каждой формы есть метод is_valid. Данный метод при вызове проверяет, верные ли данные поступили в форму (параметры data и files), и возвращает True или False. Кроме того, во время работы этого метода заполняет аттрибут формы cleaned_data - это словарь, ключами в котором являются имена полей, а значениями - данные в этих полях, приведенные к нужным типам языка python (в соответствии с типом поля). В ходе работы метода is_valid заполняются аттрибуты errors (как для формы в целом, так и для отдельных полей).
Алгоритм прост - если форма валидна, то cleaned_data не пуст, а errors пуст. Если форма не валидна, то cleaned_data пуст, а errors не пуст.
Соответственно, в зависимости от результата, который мы получили из is_valid, мы либо используем данные формы, либо не использем. Если это POST-запрос, то в случае валидной формы мы что-то делаем и возвращаем перенаправление на другой урл, либо просто опять возвращаем тот же шаблон, в котором показывали формы - и на этот раз мы нарисуем пользователю, какие ошибки он в ней допустил.
Пример:

```
myform = MyCoolForm(data=request.POST, filed=request.FILES)
if myform.is_valid(): #Ура, все круто, сохраняем объект
    mypost.title = myform.cleaned_data['title']
    mypost.text = myform.cleaned_data['text']
    mypost.save()
```

```
        return redirect... #Возвращаем редирект на страничку объекта
    return render... #Не, что-
то не так, заново рисуем страницу редактирования с этой же формой
```

Собственная валидация данных

Поля формы проверяют соответствие данны тем типам полей, которые описаны у класса формы. Если нам нужно дополнительно проверять что-либо - мы можем вмешиваться в процесс валидации, дополняя его.

Валидация формы в целом

Метод is_valid в ходе своей работы дополнительно вызывает метод clean у формы (именно в нем и происходит составление словаря cleaned_data).
Мы можем перегрузить этот метод, дополнив его так, как нам нужно:

```
class MyForm(forms.Form):
    ...
    text = forms.CharField(label=u'Текст поста')
    is_published = forms.BooleanField(label=u'Опубликовать пост')
    def clean(self):
        data = super(MyForm, self).clean()
        if data.get('is_published' and not data.get('text'):
            raise forms.ValidationError(u'Вы не можете опубликовать пост, пока не заполн
        return data
```

Здесь мы вызываем метод clean родителя (forms.Form.clean), оттуда получаем данные формы (уже приведенные к нужным типам). После чего либо возвращаем данные как есть, если все в порядке. Если же пользователь нажал галочку "Опубликовать пост", но не ввел никакого содержимого - поднимаем ошибку класса forms.ValidationError. В этом случае выполнение кода прерывается, и управление возвращается обратно к методу is_valid. Метод is_valid проверяет код поднятой ошибки, и если это forms.ValidationError - читает текст ошибки, кладет его в self.errors['__all__'] (ошибки валидации формы "в целом"), и возвращает False

Валидация отдельных полей

В ходе валидации метод is_valid пытается найти в классе формы методы с именем "clean_имя-поля". Это происходит до общего метода clean, но уже после составления словаря cleaned_data у формы. Если такой метод найден - он вызывается, и мы в нем достаем значение поля из self_cleaned.data и либо поднимаем ошибку, либо возвращаем значение поля. На примере:

```
class MyCoolForm...
    def clean_text(self):
        text = self.cleaned_data.get('text')
```

```
if u'Вася' in text:
    raise forms.ValidationError(u'Нельзя упоминать имя Васи всуе!!!')
return text
```

Почти аналогично предыдущему примеру - достали значение поля, проверили что-нибудь. Либо подняли ошибку, либо вернули значение. Если ошибка - то метод is_valid проверит ее и дополнит аттрибут self.errors['text'] (а в шаблоне он будет дополнительно доступен как {{ myform.text.errors }})

Общие атрибуты для всех типов полей в формах django

required

Определяет, должно ли поле быть заполнено для того, чтобы присланные в форму данные считались верными

label

Определяет человекочитаемый текст, который будет показан пользователю в теге label

initial

Можно предзаполнить чем-либо значение поля

widget

Если задан - должен содержать один из классов пакета forms, отвечающие за отрисовку полей (так называемые виджеты).
У каждого поля задан какой-либо widget по умолчанию, но можно переопределить.
В основном применяется для отрисовки текстовых полей через textarea:

```
text = forms.CharField(widget=forms.Textarea)
```

Типы полей

Типы полей можно смотреть в документации django, их довольно много. Отдельно стоит упомянуть лишь несколько:

forms.ChoiceField

Предоставляет выбор одного из значений из заранее определенного списка.
Параметры:
choices - обязательный параметр, должен содержать список пар "значение"-"текст".
На примере:

```
COLOR_CHOICES = (('red', u'Красный'), ('green', u'Зеленый'), ('blah', u'Абыр-абыр!'))
color_type = forms.ChoiceField(choices=COLOR_CHOICES, label=u'Выберите цвет')
```

Данный пример создаст поле, которое при валидации проверит, введено ли одно из значений red, green или blah.
В шаблоне будет нарисован соответствующий select
После валидации из cleaned_data мы достанем 'red', 'green' или 'blah'

forms.MultipleChoiceField

Аналогично предыдущему, но позволяет выбрать несколько вариантов. Соответственно, из cleaned_data мы достанем список, содержащий какие-либо комбинации из 'red', 'green' или 'blah'

forms.ModelChoiceField

Очень похоже на ChoiceField, но принимает параметр QuerySet, рисует select со списком объектов из выборки, а в cleaned_data возвращает сам экземпляр модели.
На примере:

blog = forms.ModelChoiceField(queryset=Blog.objects.all(), label=u'Выберите блог')

В этом случае мы из cleaned_data достанем сразу экземпляр модели Blog

forms.ModelMultipleChoiceField

Аналогично предыдущему, но с мультивыбором. Из cleaned_data достаем список объектов

Различные виджеты для полей выбора

По умолчанию четыре предыдущих типа полей рендерятся в тег select.
Это можно переопределить, передав им в параметре widget один из этих классов https://docs.djangoproject.com/en/1.9/ref/forms/widgets/#selector-and-checkbox-widgets
Например:

blog = forms.ModelMuplipleChoiceField(queryset=Blog.objects.all(), label=u'Выберите блог', widget=forms.CheckboxInput)

Получим то же поведение, но в шаблоне отрисуется список input type="checkbox", где мы галочками отметим нужные блоги

forms.ModelForm

Внимание, **УБЕРШТУКОВИНА**!
Вы наверное уже заметили, что описывать полностью типы полей, валидацию и все прочее для формы в django довольно однообразно и утомительно. Мне кажется, даже читать про это вам будет довольно утомительно. При этом прослеживается явная корреляция между типами полей модели и типами полей формы для этой модели.
Теоретически, саму модель можно описать достаточно подробно для того, чтобы

заставить django генерить для нее форму автоматически.
Этот функционал реализован в классе forms.ModelForm. Это - наследник класса forms.Form, который умеет генерить формы для редактирования и создания объектов.
Работает это следующим образом - вы создаете наследника класса ModelForm, внутри которого создаете класс Meta, и в нем описываете минимум два параметра - model и fields.

model

Модель, по которой будет создана форма. Например, Post

fields

Список полей, которые нужно использовать. Например: 'title', 'text'

Пример

```
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text')
```

Всё, готово. Мы получили форму, которая умеет всё, чтобы обращаться с экземплярами модели Post, в том числе создавать и редактировать их. А параметр fields ограничивает поля, которые эта форма вообще имеет право трогать (например, мы врядли захотим дать пользователю редактировать поле "Автор поста", так что поле author мы в fields не включаем)

Инициализация ModelForm

instance

При создании экземпляра класса ModelForm мы можем передать все те же параметры, что и обычной forms.Form.
Но. Дополнительно мы передаем параметр instance, в который складываем объект той модели, с которой должна работать наша форма. Форма же самостоятельно возьмет всю нужную информацию из этого объекта, и заполнит поля значениями. Например:

```
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text')

post = Post.objects.get(id=10)
post_form = PostForm(instance=post)
```

Всё, после этого нам остается только отрендерить эту форму в шаблоне.
А в форме будет заполнен атрибут save.instance

метод save

Мало того, ModelForm умеет сохранять объекты (будучи вызвана с параметром

data). Для этого используется метод save, которые принимает параметр commit (True/False) и возвращает нам экземпляр модели с полями, уже заполненными данными, который ввел пользователь.
Если commit=True, то объект уже будет сохранен в базе данных.
Если commit=False, то сохранения объекта не произойдет, мы дозаполнить недостающие поля у объекта и вызвать у него метод save самостоятельно.
Пример:

```
post = Post.objects.get(id=10)
post_form = PostForm(data=request.POST, instance=post)
if post_form.is_valid():
    post = post_form.save(commit=True)
```

Еще пример:

```
post = Post()
post_form = PostForm(data=request.POST, instance=post)
if post_form.is_valid():
    post = post_form.save(commit=False)
    post.author = request.user
    post.save()
```

Использование форм в class-based-views

Прямо сейчас пишется небольшой пост на тему class-based-views, там я и освещу эту тему. Пока всю инфу можно найти тут
https://docs.djangoproject.com/en/1.9/topics/class-based-views/generic-editing/

Что почитать

Относительно форм в html - всё есть на http://htmlbook.ru/
Относительно форм в django: можно стартануть отсюда
https://docs.djangoproject.com/en/1.9/topics/forms/ , там есть все ссылки и на список полей и на список виджетов, и все-все-все. Аналог на русском на http://djbook.ru/

Вопросы

Приветствуются. Я понимаю, что публикую все эти посты довольно поздно, у вас остался один день на подготовку ДЗ и всё такое. Так что можете мучить меня по полной. Я думаю, желательно делать это в комментариях, т.к. тогда ответы увидят все и возможно я отвечу один раз, а не десять, на один и тот же вопрос :)