# Lecture 3. Формы Django

## Определение форм

Django предоставляет специальные возможности по работе с формами, которые позволяют определять функциональноть форм в одном месте и использовать многократно в разных местах, упрощают валидацию данных, помогают связывать формы с моделями и многое другое.

Каждая форма определяется в виде отдельного класса, который расширяет класс forms.Form. Классы размещаются внутри проекта, где они используются. Нередко они помещаются в отдельный файл, который называется, к примеру, forms.py. Однако также формы могут размещаться внутри уже имеющихся в приложении файлов, например, в views.py или models.py.

Например, создадим в приложении новый файл forms.py и поместим в него следующий код:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField()  age = forms.IntegerField() |

Класс формы называется UserForm. Он определяет два поля. Поле name представляет тип forms.CharField и будет генерировать поле input type="text". Поле age представляет тип forms.IntegerField и будет генерировать поле input type="number". То есть первое поле для ввода текста, а второе для ввода чисел.

Далее в файле views.py определим следующее представление:

|  |
| --- |
| **from** django.shortcuts **import** render **from** .forms **import** UserForm  **def** index(request):  userform = UserForm()  **return** render(request, **"index.html"**, {**"form"**: userform}) |

Здесь объект формы передается в шаблон index.html в виде переменной form.

И определим следующий шаблон index.html:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Django Forms</**title**> </**head**> <**body**>  <**table**>  {{ **form** }}  </**table**> </**body**> </**html**> |

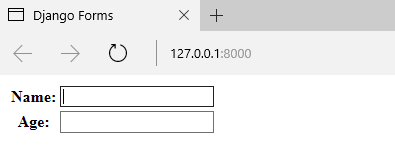
Файл urls.py

|  |
| --- |
| **from** django.urls **import** path **from** django.views.generic **import** TemplateView **from** myApplication **import** views  urlpatterns = [  path(**''**, views.index, name=**"homepage"**),  ] |

В итоге из переменной form будет генерироваться следующий код html:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"**/>  <**title**>Django Forms</**title**> </**head**> <**body**> <**table**>  <**tr**>  <**th**><**label for="id\_name"**>Name:</**label**></**th**>  <**td**>  <**input type="text" name="name" required id="id\_name"**>  </**td**>  </**tr**>  <**tr**>  <**th**><**label for="id\_age"**>Age:</**label**></**th**>  <**td**>  <**input type="number" name="age" required id="id\_age"**>  </**td**>  </**tr**> </**table**> </**body**> </**html**> |

Результат:



**Получение POST-запроса**

Теперь с помощью форм Django определим полнофункциональную форму, с которой можно отправлять данные на сервер.

Вначале изменим шаблон index.html:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"**/>  <**title**>Django Forms</**title**> </**head**> <**body**> <**form method="POST"**>  {% **csrf\_token** %}  <**table**>  {{ **form** }}  </**table**>  <**input type="submit" value="Send"**> </**form**> </**body**> </**html**> |

Для создания формы здесь использован стандартный элемент html <form>. В начале формы помещен встроенный тег Django {% csrf\_token %}, который позволяет защитить приложение от CSRF-атак, добавляя в форму в виде скрытого поля csrf-токен.

[CSRF](http://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%B4%D0%B5%D0%BB%D0%BA%D0%B0_%D0%BC%D0%B5%D0%B6%D1%81%D0%B0%D0%B9%D1%82%D0%BE%D0%B2%D1%8B%D1%85_%D0%B7%D0%B0%D0%BF%D1%80%D0%BE%D1%81%D0%BE%D0%B2) (Cross-Site Request Forgery, также XSRF) – опаснейшая атака, которая приводит к тому, что хакер может выполнить на неподготовленном сайте массу различных действий от имени других, зарегистрированных посетителей.

Какие это действия – отправка ли сообщений, перевод денег со счёта на счёт или смена паролей – зависят от сайта, но в любом случае эта атака входит в образовательный минимум веб-разработчика.

Внизу формы определена кнопка для отправки данной формы на сервер.

Теперь изменим представление в файле views.py:

|  |
| --- |
| **from** django.shortcuts **import** render **from** django.http **import** HttpResponse **from** .forms **import** UserForm  **def** index(request):  **if** request.method == **"POST"**:  name = request.POST.get(**"name"**)  *# age = request.POST.get("age") # получение значения поля age* **return** HttpResponse(**"<h2>Hello, {0}</h2>"**.format(name))  **else**:  userform = UserForm()  **return** render(request, **"index.html"**, {**"form"**: userform}) |

Поскольку в шаблоне форма по умолчанию будет отправляться на тот же адрес, то представление обабатывает сразу да типа запросов GET и POST. Для определения типа запроса проверяем значение request.method.

Если запрос типа POST, то вначале создаем объект UserForm, наполняя его данными, которые пришли в запросе через request.POST. То есть это и будут отправленные данные формы. Затем мы можем получить эти данные по отдельности для каждого поля формы. После этого отправляем пользователю сообещние через объект HttpResponse. В прицние тут можно было бы сделать переадресацию или использовать другой шаблон для генерации ответа.

Если запрос представляет тип GET, то просто отправляем форму для ввода данных.

Таким образом, при обращении к приложению мы вначале увидим форму ввода. Введем в нее некоторые данные:

После нажатия на кнопку введенные данные в запросе POST опять же уйдут преставлению index, которое обработает их и в ответ отправить пользователю сообщение с введенным именем:

## Типы полей формы

В формах Django мы можем использовать следующие классы для создания полей форм:

* forms.BooleanField: создает поле <input type="checkbox" >. Возвращает значение Boolean: True - если флажок отмечен и False - если флажок не отмечен.
* forms.NullBooleanField: создает следующую разметку:

|  |
| --- |
| <select>  <option value="1" selected="selected">Unknown</option>  <option value="2">Yes</option>  <option value="3">No</option>  </select> |

* forms.CharField: предназначен для ввода текста и создает следующую разметку:

|  |
| --- |
| <input type="text"> |

* forms.EmailField: предназначен для ввода адреса электронной почты и создает следующую разметку:

|  |
| --- |
| <input type="email"> |

* forms.GenericIPAddressField: предназначен для ввода IP-адреса в формате IP4v или IP6v и создает следующую разметку:

|  |
| --- |
| <input type="text"> |

* forms.RegexField (regex="регулярное\_выражение"): предназначен для ввода текста, который должен соответствовать определенному регулярному выражению. Создает текстовое поле:

|  |
| --- |
| <input type="text"> |

* forms.SlugField(): предназначен для ввода текста, который условно называется "slug", то есть последовательность символов в нижнем регистре, чисел, дефисов и знаков подчеркивания. Создает текстовое поле:

|  |
| --- |
| <input type="text"> |

* forms.URLField(): предназначен для ввода ссылок. Создает следующее поле:

|  |
| --- |
| <input type="url"> |

* forms.UUIDField(): предназначен для ввода UUID (универсального уникального идентификатора). Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.ComboField(fields=[field1, field2,..]): аналогичен обычному текстовому полю за тем исключением, что требует, чтобы вводимый текст соответствовал требованиям тех полей, которые передаются через параметр fields. Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.MultiValueField(fields=[field1, field2,..]): предназначен для создания сложных компоновок, состоящих из нескольких полей.
* forms.FilePathField(path="каталог файлов"): создает список select, который содержит все папки и файлы в определенном каталоге:

|  |
| --- |
| <select>  <option value="folder/file1">folder/file1</option>  <option value="folder/file2">folder/file2</option>  <option value="folder/file3">folder/file3</option>  //.............................................  </select> |

* forms.FileField(): предназначен для выбора файла. Создает следующее поле:

|  |
| --- |
| <input type="file"> |

* forms.ImageField(): предназначен также для выбора файла, но при этом добавляет ряд дополнительных возможностей. Создает следующее поле:

|  |
| --- |
| <input type="file"> |

* forms.DateField(): предназначен установки даты. В создаваемое поле вводится текст, который может быть сконвертирован в дату, например, 2017-12-25 или 11/25/17. Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.TimeField(): предназначен ввода времени, например, 14:30:59 или 14:30. Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.DateTimeField(): предназначен ввода даты и времени, например, 2017-12-25 14:30:59 или 11/25/17 14:30. Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.DurationField(): предназначен временного промежутка. Вводимый текст должен соответствовать формату "DD HH:MM:SS", например, 2 1:10:20 (2 дня 1 час 10 минут 20 секунд). Создает следующее поле:

|  |
| --- |
| <input type="text"> |

* forms.SplitDateTimeField(): создает два текстовых поля для ввода соответственно даты и времени:

|  |
| --- |
| <input type="text" name="\_0" >  <input type="text" name="\_1" > |

* forms.IntegerField(): предназначен для ввода чисел. Создает следующее поле:

|  |
| --- |
| <input type="number"> |

* forms.DecimalField(): предназначен для ввода чисел. Создает следующее поле:

|  |
| --- |
| <input type="number"> |

* forms.FloatField(): предназначен для ввода чисел. Создает следующее поле:

|  |
| --- |
| <input type="number"> |

* forms.ChoiceField(choises=кортеж\_кортежей): генерирует список select, каждый из его элементов формируется на основе отдельного кортежа. Например, следующее поле:

|  |
| --- |
| forms.ChoiceField(choices=((1, "English"), (2, "German"), (3, "French"))) |

будет генерировать следующую разметку:

|  |
| --- |
| <select>  <option value="1">English</option>  <option value="2">German</option>  <option value="3">French</option>  </select> |

* forms.TypedChoiceField(choises=кортеж\_кортежей, coerce=функция\_преобразования, empty\_value=None): также генерирует список select на основе кортежа. Однако дополнительно принимает функцию преобразования, которая преобразует каждый элемент. И также принимает параметр empty\_value, который указывает на значение по умолчанию.
* forms.MultipleChoiceField(choises=кортеж\_кортежей): также генерирует список select на основе кортежа, как и forms.ChoiceField, добавляя к создаваемому полю атрибут multiple="multiple". То есть список поддерживает множественный выбор.
* forms.TypedMultipleChoiceField(choises=кортеж\_кортежей, coerce=функция\_преобразования, empty\_value=None): аналог forms.TypedChoiceField для списка с множественным выбором.

**Виджеты Django**

Выше рассмотренные поля при генерации разметки используют определенные виджеты из пакета forms.widgets. Например, класс CharField использует виджет forms.widgets.TextInput, а ChoiceField использует forms.widgets.Select. Но есть ряд виджетов, которые по умолчанию не используются полями форм, но тем не менее мы их можем задействовать:

* PasswordInput: генерирует поле для ввода пароля <input type="password">
* HiddenInput: генерирует скрытое поле <input type="hidden" >
* MultipleHiddenInput: генерирует набор скрытых полей
* TextArea: генерирует многострочное текстовое поле <textarea></textarea>
* RadioSelect: генерирует список переключателей (радиокнопок) <input type="radio" >
* CheckboxSelectMultiple: генерирует список флажков <input type="checkbox" >
* TimeInput: генерирует поле для ввода времени (например, 12:41 или 12:41:32)
* SelectDateWidget: генерирует три поля select для выбора дня, месяца и года
* SplitHiddenDateTimeWidget: использует скрытое поле для хранения даты и времени
* FileInput: генерирует поле для выбора файла

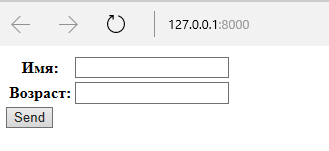
## Настройка формы и ее полей

**label**

Свойство label позволяет установить текстовую метку, которая отображается рядом с полей. По умолчанию она отображает название самого поля с большой буквы. Например:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(label=**"Имя"**)  age = forms.IntegerField(label=**"Возраст"**) |

Эта форма сформирует следующую веб-страницу:

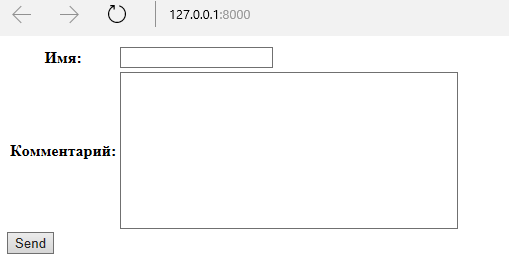


**widget**

Параметр widget позволяет задать виджет, который будет использоваться для генерации разметки html:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(label=**"Имя"**)  comment = forms.CharField(label=**"Комментарий"**, widget=forms.Textarea) |

По умолчанию поле CharField использует виджет forms.widgets.TextInput, который создает однострочное текстовое поле. Однако если нам надо создать многострочное текстовое поле, то необходимо воспользоваться виджетом forms.Textarea:



**Значения по умолчанию**

С помощью параметра initial можно установить значения по умолчанию.

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(initial=**"undefined"**)  age = forms.IntegerField(initial=18) |

**Порядок полей**

Поля ввода отображаются на веб-странице в том порядке, в котором они определены в классе формы. С помощью свойства **field\_order** можно переопределить порядок, как в классе формы:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField()  age = forms.IntegerField()  field\_order = [**"age"**, **"name"**] |

Так и при определении объекта формы в представлении:

|  |
| --- |
| **from** django.shortcuts **import** render **from** django.http **import** HttpResponse **from** .forms **import** UserForm  **def** index(request):  userform = UserForm(field\_order = [**"age"**, **"name"**])  **return** render(request, **"index.html"**, {**"form"**: userform}) |

**help\_text**

Параметр help\_text устанавливает подсказку рядом с полем ввода:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(help\_text=**"Введите свое имя"**)  age = forms.IntegerField(help\_text=**"Введите свой возраст"**) |

**Настройка вида формы**

С помощью специальных методов можно настроить обзее отображение формы:

* as\_table(): отображение в виде таблицы
* as\_ul(): отображение в виде списка
* as\_p(): каждое поле формы отобажается в отдельном параграфе

Применение методов Index.html:

|  |
| --- |
| <**h2**>as\_table</**h2**> <**form method="POST"**>  {% **csrf\_token** %}  <**table**>  {{ **form**.**as\_table** }}  </**table**>  <**input type="submit" value="Send"**> </**form**> <**h2**>as\_ul</**h2**> <**form method="POST"**>  {% **csrf\_token** %}  <**ul**>  {{ **form**.**as\_ul** }}  </**ul**>  <**input type="submit" value="Send"**> </**form**> <**h2**>as\_p</**h2**> <**form method="POST"**>  {% **csrf\_token** %}  <**div**>  {{ **form**.**as\_p** }}  </**div**>  <**input type="submit" value="Send"**> </**form**> |

## Валидация данных

Теоретически пользователь может ввести в форму какие угодно данные и отправить их. Однако не все данные бывают уместными или корректными. Например, в поле для возраста пользователь может ввести отрицательное число, что вряд ли может считаться корректным возрастом. В этой связи для проверки корректности вводимых данных используется механизм валидации.

**Правила валидации**

Основным элементом валидаии являются правила, которые задают параметры корректности вводимых данных. Например, для всех поле по умолчанию устанавливается обязательность ввода значения. И при генерации html-кода для поля ввода устанавливается атрибут required. И если мы попробуем отправить форму, если какое-то из ее полей не введено никакого значения, то мы получим ошибку:

И нам надо будет обязательно ввести какое-то значение в незаполненное поле. Однако это не всегда нужно. Допустим, одно поле может иметь, а может не иметь значение. В этом случае мы можем отключить атрибут required:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField()  age = forms.IntegerField(required=**False**)  email = forms.EmailField(required=**False**) |

**Длина текста**

Для полей, которые требуют ввода текста, например, CharField, EmailField и др., с помощью параметров max\_length и min\_length можно задать соответственно максимальную и минимальную длину вводимого текста в символах.

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(min\_length=2, max\_length=20)  email = forms.EmailField(required=**False**, min\_length=7) |

При генерации разметки для полей ввода будут устанавливаться атрибуты maxlength и minlength.

**Минимальное и максимальное число**

Для объектов IntegerField, DecimalField и FloatField можно устанавливать параметры max\_value и min\_value, которые задают соответственно максимально допустимое и минимально допустимое значение.

DecimalField дополнительно может принимать еще параметр decimal\_places, который указывает на максимальное количество знаков после запятой.

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField()  age = forms.IntegerField(min\_value=1, max\_value=100)  weight = forms.DecimalField(min\_value=3, max\_value=200, decimal\_places=2) |

**is\_valid**

Выше рассмотренные атрибуты позволяют валидировать значения при вводе на стороне клиента. Однако практически пользователя, имея определенные навыки, могут все равно отправить форму с заведомо некорректными данными. Например, через инструменты для разработчиков в веб-браузере можно подправить исходный код формы, добавив ей атрибут novalidate, который отключает клиентскую валидацию. Поэтому проверку на валидность данных также надо определять на стороне сервера. Для этого у формы вызывается метод is\_valid(), который возвращает True, если данные корректны, и False - если данные некорректны. Чтобы использоать этот метод, надо создать объект формы и передать ей пришедшие из запроса данные.

Итак, определим следующее представление в файле views.py:

|  |
| --- |
| **from** django.shortcuts **import** render **from** django.http **import** HttpResponse **from** .forms **import** UserForm   **def** index(request):  **if** request.method == **"POST"**:  userform = UserForm(request.POST)  **if** userform.is\_valid():  name = userform.cleaned\_data[**"name"**]  **return** HttpResponse(**"<h2>Hello, {0}</h2>"**.format(name))  **else**:  **return** HttpResponse(**"Invalid data"**)  **else**:  userform = UserForm()  **return** render(request, **"index.html"**, {**"form"**: userform}) |

Если приходит POST-запрос, то в начале заполняем форму пришедшими данными:

|  |
| --- |
| userform = UserForm(request.POST) |

Потом проверяем их корректность:

|  |
| --- |
| if userform.is\_valid(): |

После проверки на валидность мы можем получить данные через объект cleaned\_data (если данные корректны):

|  |
| --- |
| name = userform.cleaned\_data["name"] |

Если данные некорректны, можно предусмотреть альтернативный вывод:

|  |
| --- |
| return HttpResponse("Invalid data") |

Для тестирования формы можно установить у ней атрибут novalidate:

|  |
| --- |
| <**form method="POST" novalidate**>  {% **csrf\_token** %}  <**table**>  {{ **form** }}  </**table**>  <**input type="submit" value="Send"**> </**form**> |

## Детальная настройка полей формы

Форма и поля допускают установку ряда параметров, которые позволяют частично кастомизировать отображение полей и формы. Тем не менее этого нередко бывает недостаточно. Например, необходимо применить стилизацию или добавить рядом с полем ввода какой-нибудь специальный текст. И Django позволяет нам коренным образом изменить всю композицию создаваемых полей.

В частности, в шаблоне компонента мы можем обратиться к каждому отдельному полю формы через название формы: form.название\_поля. По названию поля мы можем получить непосредственно генерируемый им элемент-html без внешних надписей и какого-то дополнительного кода. Кроме того, каждое поле имеет ряд ассоциированных с ним значений:

* form.название\_поля.name: возвращает название поля
* form.название\_поля.value: возвращает значение поля, которое ему было передано по умолчанию
* form.название\_поля.label: возвращает текст метки, которая генерируется рядом с полем
* form.название\_поля.id\_for\_label: возвращает id для поля, которое по умолчанию создается по схеме id\_имяполя.
* form.название\_поля.auto\_id: возвращает id для поля, которое по умолчанию создается по схеме id\_имяполя.
* form.название\_поля.label\_tag: возвращает элемент label, который представляет метку рядом с полем
* form.название\_поля.help\_text: возвращает текст подказки, ассоциированный с полем
* form.название\_поля.errors: возвращает ошибки валидации, связанные с полем
* form.название\_поля.css\_classes: возвращает css-классы поля
* form.название\_поля.as\_hidden: генерирует для поля разметку в виде скрытого поля <input type="hidden">
* form.название\_поля.is\_hidden: возвращает True или False в зависимости от того, является ли поле скрытым
* form.название\_поля.as\_text: генерирует для поля разметку в виде текстового поля <input type="text">
* form.название\_поля.as\_textarea: генерирует для поля разметку в виде <textarea></textarea>
* form.название\_поля.as\_widget: возвращает виджет Django, ассоциированны с полем

Так, чтобы получить текст на метке поля, которое называется age, нам надо использовать выражение form.age.label.

Например, возмьмем простейшую форму:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField()  age = forms.IntegerField() |

В представлении передадим эут форму в шаблон:

|  |
| --- |
| **from** django.shortcuts **import** render **from** django.http **import** HttpResponse **from** .forms **import** UserForm  **def** index(request):  userform = UserForm()  **if** request.method == **"POST"**:  userform = UserForm(request.POST)  **if** userform.is\_valid():  name = userform.cleaned\_data[**"name"**]  **return** HttpResponse(**"<h2>Hello, {0}</h2>"**.format(name))  **return** render(request, **"index.html"**, {**"form"**: userform}) |

И в шаблоне index.html пропишем использование полей формы:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"**/>  <**title**>Django Forms</**title**> </**head**> <**body**> <**form method="POST" novalidate**>  {% **csrf\_token** %}  <**div**>  {% **for field in form** %}  <**div class="form-group"**>  {{ **field**.**label\_tag** }}  <**div**>{{ **field** }}</**div**>  <**div class="error"**>{{ **field**.**errors** }}</**div**>  </**div**>  {% **endfor** %}  </**div**>  <**input type="submit" value="Send"**> </**form**> </**body**> </**html**> |

Фактически форма представляет набор полей, и с помощью выражения {% for field in form %} мы пробегаемся по каждому полю на форме и можем управлять его отображением - отображением собственно поля и связанных с ним атрибутов - ошибок, текста подсказки, метки и т.д.

Например, после отправки некорректных данных мы получим следующую веб-страницу:



Одно поле может содержать несколько ошибок. В этом случае можно использовать тег for для их последовательного вывода:

|  |
| --- |
| {% **for error in field**.**errors** %}  <**div class="alert alert-danger"**>{{ **error** }}</**div**> {% **endfor** %} |

## Стилизация полей форм

Поля формы применяют некоторые стили по умолчанию. Если же мы хотим применить к ним какие-то собственные стили и классы, то нам надо использовать ряд механизмов.

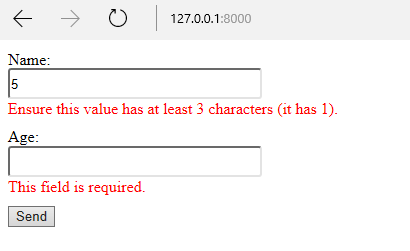
Прежде всего мы можем вручную выводить каждое поле и определять правила стилизации для этого поля или окружающих его блоков. Возьмем простейшую форму:

|  |
| --- |
| from django import forms    class UserForm(forms.Form):  name = forms.CharField(min\_length=3)  age = forms.IntegerField(min\_value=1, max\_value=100) |

В шаблоне пропишем ее использование:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"**/>  <**title**>Django Forms</**title**>  <**style**>  .**alert** {  **color**: **red** }   .**form-group** {  **margin**: 10**px** 0;  }   .**form-group input** {  **width**: 250**px**;  **height**: 25**px**;  **border-radius**: 3**px**;  }  </**style**> </**head**> <**body class="container"**> <**form method="POST" novalidate**>  {% **csrf\_token** %}  <**div**>  {% **for field in form** %}  <**div class="form-group"**>  {{ **field**.**label\_tag** }}  <**div**>{{ **field** }}</**div**>  {% **if field**.**errors** %}  {% **for error in field**.**errors** %}  <**div class="alert alert-danger"**>  {{ **error** }}  </**div**>  {% **endfor** %}  {% **endif** %}  </**div**>  {% **endfor** %}  </**div**>  <**input type="submit" value="Send"**> </**form**> </**body**> </**html**> |

Результа при отправке формы с ошибками:



Второй механизм представляют свойства формы required\_css\_class и error\_css\_class, который соответственно применяют класс css к метке, создаваемой для поля формы, и к блоку ассоциированных с ним ошибок.

Например, определим следующую форму:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(min\_length=3)  age = forms.IntegerField(min\_value=1, max\_value=100)  required\_css\_class = **"field"** error\_css\_class = **"error"** |

В этом случае в шаблоне у нас должны быть определены или подключены классы "field" и "error":

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Django Forms</**title**>  <**style**>  .**field**{**font-weight**:**bold**;}  .**error**{**color**:**red**;}  </**style**> </**head**> <**body class="container"**>  <**form method="POST" novalidate**>  {% **csrf\_token** %}  <**table**>  {{**form**}}  </**table**>  <**input type="submit" value="Send"** >  </**form**> </**body**> </**html**> |

Но также можно было бы комбинировать оба способа:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"** />  <**title**>Django Forms</**title**>  <**style**>  .**field**{**font-weight**:**bold**;}  .**error**{**color**:**red**;}  </**style**> </**head**> <**body**>  <**form method="POST" novalidate**>  {% **csrf\_token** %}  <**div**>  {% **for field in form** %}  <**div class="row"**>  {{**field**.**label\_tag**}}  <**div class="col-md-10"**>{{**field**}}</**div**>  {% **if field**.**errors**%}  <**div class="error"**>{{**field**.**errors**}}</**div**>  {% **endif** %}  </**div**>  {% **endfor** %}  </**div**>  <**input type="submit" value="Send"** >  </**form**> </**body**> </**html**> |

Третий механизм стилизации представляет установка классов и стилей через виджеты:

|  |
| --- |
| **from** django **import** forms  **class** UserForm(forms.Form):  name = forms.CharField(widget=forms.TextInput(attrs={**"class"**: **"myfield"**}))  age = forms.IntegerField(widget=forms.NumberInput(attrs={**"class"**: **"myfield"**})) |

В данном случае через параметр виджетов attrs устанавливаются атрибуты того элемента html, который будет генерироваться. В частности, здесь для обоих полей устанавливается атрибут class, который представляет класс myfield.

И, допустим, в шаблоне будет определен класс myfield:

|  |
| --- |
| <!DOCTYPE **html**> <**html**> <**head**>  <**meta charset="utf-8"**/>  <**title**>Django Forms</**title**>  <**style**>  .**myfield** {  **border**: 1**px solid #ccc**;  **border-radius**: 5**px**;  **height**: 25**px**;  **width**: 200**px**;  **margin**: 10**px** 10**px** 10**px** 0;  }  </**style**> </**head**> <**body**> <**form method="POST"**>  {% **csrf\_token** %}  <**div**>  {% **for field in form** %}  <**div class="row"**>  {{ **field**.**label\_tag** }}  <**div class="col-md-10"**>{{ **field** }}</**div**>  </**div**>  {% **endfor** %}  </**div**>  <**input type="submit" value="Send"**> </**form**> </**body**> </**html**> |