

## Тема 2.4.7. Формы.

Форма – это объект, включающий набор полей (виджетов) на веб-странице для получения данных от пользователя для последующей передачи их на сервер.

Виджет – представление элемента ввода на HTML-странице.  
Отвечает за внешний вид поля.

Каждая форма определяется в виде отдельного класса, который расширяет класс `forms.Form`. Классы размещаются внутри проекта, где они используются. Формы могут быть помещены в отдельный файл, например, `forms.py`. Также формы могут размещаться внутри уже имеющихся в приложении файлов, например, в `views.py` или `models.py`.

Создадим в приложении `firstapp` проекта `hello` новый файл `forms.py` и поместим в него следующий код:

```
from django import forms

class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента")
    age = forms.IntegerField(label="Возраст клиента")
```

Далее в файле **`views.py`** отредактируем представление **`index`**:

```
from django.shortcuts import render
from django.http import HttpResponse
from .forms import UserForm

def index(request):
    """
    :param request:
    :return:
    """
    userform = UserForm()
    return render(request, "firstapp/index.html", {"form": userform})
```

Объект формы передается здесь в шаблон index.html в виде переменной form.

Изменим шаблон index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title> Это Форма Django </title>
</head>
<body>
  <table>
    {{ form }}
  </table>
</body>
</html>
```

Страница на сервере отобразит:

<b>Имя клиента:</b>	<input type="text"/>
<b>Возраст клиента:</b>	<input type="text"/>

## Использование в формах POST-запросов

Создадим форму, в которой можно не только вводить данные, но и отправлять их на сервер.

Сначала изменим шаблон index.html:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title> Это Форма Django </title>
</head>
<body>
  <form method="POST">
    {% csrf_token %}
    <table>
      {{ form }}
    </table>
    <input type="submit" value="Отправить" >
  </form>
</body>
</html>
```

Тег Django `{% csrf_token %}` позволяет защитить приложение от CSRF (Cross-Site Request Forgery) атак, добавляя в форму в виде скрытого поля csrf-токен.

Далее изменим функцию `index()` в файле `views.py`:

```
from django.shortcuts import render
from django.http import *
from .forms import UserForm

def index(request):
    """
    :param request:
    :return:
    """
    if request.method == "POST":
        # получение значения поля Имя
        name = request.POST.get("name")
        # получение значения поля Возраст
        age = request.POST.get("age")
        output = "<h2>Пользователь</h2><h3>Имя - {0}, Возраст - {1}</h3>".format(name, age)
        return HttpResponse(output)
    else:
        userform = UserForm()
        return render(request, "firstapp/index.html", {"form": userform})
```

Представление обрабатывает сразу два типа запросов: GET и POST.

Для определения типа запроса делается проверка значения `request.method`.

Если запрос типа POST, то будет форма для отправки данных.

Если запрос представляет тип GET (ветка else), то будет форма для ввода данных.

Страница отобразит:

<b>Имя клиента:</b>	<input type="text"/>
<b>Возраст клиента:</b>	<input type="text"/>
<input type="button" value="Отправить"/>	

Введем в поля данные:

<b>Имя клиента:</b>	<input type="text" value="Иван"/>
<b>Возраст клиента:</b>	<input type="text" value="22"/>
<input type="button" value="Отправить"/>	

Если запрос будет типа POST (`request.method == "POST"`)), то данные будут отправлены по нажатию кнопки Отправить.

Отправляемые данные будут присвоены переменным `name` и `age`, а их значения переменной `output`. Затем значения из `output` будут отправлены через объект `HttpResponse`. В данном примере ответ отправляется пользователю на ту же HTML-страницу. Щелкаем по кнопке Отправить и получаем:

## Пользователь

**Имя - Иван, Возраст – 22**

### Виджеты Django

Поля формы 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 мы их можем применять:

**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

## Типы полей формы

В формах Django доступны следующие классы для создания полей форм:

**forms.BooleanField:** создает поле `<input type="checkbox" >` для выбора решения. Возвращает значение Boolean: True - если флажок отмечен и False - если флажок не отмечен. По умолчанию устанавливается виджет `CheckboxInput`.

### Пример

```
from Django import forms  
class UserForm(form.Form)  
    basket= forms.BooleanField(label="Положить товар в корзину")
```

**forms.NullBooleanField:** создает следующую разметку:

```
<select>  
<option value="1" selected="selected">Неизвестно</option>  
<option value="2">Да</option>  
<option value="3">Нет</option>  
</select>
```

По умолчанию использует виджет NullBooleanSelect.

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

```
<input type="text">
```

По умолчанию использует виджет TextInput.

### Пример:

```
from Django import forms  
class UserForm(form.Form)  
    name= forms.CharField(label="Имя клиента")
```

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

```
<input type="email">
```

По умолчанию использует виджет `EmailInput`.

Пример:

```
from Django import forms

class UserForm(form.Form)

email= forms.EmailField(label="электронный адрес",
help_text="обязательный символ – @")
```

**forms.GenericIPAddressField:** предназначен для ввода IP-адреса в формате IPv4 или IPv6 и создает следующую разметку:

```
<input type="text">
```

По умолчанию использует виджет `TextInput`.

Пример:

```
from Django import forms

class UserForm(form.Form)

ip_adres= forms.GenericIPAddressField (label="IP адрес",
help_text="Пример формата 192.0.2.0")
```

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

```
<input type="text">
```

По умолчанию использует виджет `TextInput` и метод `RegexValidator`.

Пример:

```
from Django import forms  
class UserForm(form.Form)  
    reg_text= forms.RegexField(label="Текст", regex="^[0-9][A-F][0-9]$")
```

**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">
```

По умолчанию использует виджет TextInput.

Пример:

```
from Django import forms  
class UserForm(form.Form)  
    slug_text= forms.SlugField(label="Введите текст")
```

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

```
<input type="url">
```

По умолчанию использует виджет URLInput

Пример:

```
from Django import forms  
class UserForm(form.Form)  
    url_text= forms.URLField(label="Введите URL", help_text="Например,  
http://www.google.com")
```



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

```
<input type="text">
```

По умолчанию использует виджет TextInput

Пример:

```
from Django import forms

class UserForm(form.Form)

uuid_text= forms.UUIDField(label="Введите UUID",
help_text="Формат xxxxxxxx_xxxx_xxxx_xxxx_xxxxxxxxxxxxxx")
```

**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">
```

Пример:

```
from Django import forms

class UserForm(form.Form)

        combo_text= forms.ComboField(label="Введите
        URL",fields=[ forms.URLField(),
        forms.CharField(max_length=20)])
```

**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>
```

По умолчанию использует виджет Select.

Пример:

```
from Django import forms
class UserForm(form.Form)
    file_path= forms.FilePathField(label="Выберите файл:",
path="C:/Python/")
```

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

```
<input type="file">
```

По умолчанию использует виджет ClearableFileInput.

Пример:

```
from Django import forms
class UserForm(form.Form)
    file= forms.FileField(label="Файл")
```

**forms.ImageField()**: предназначен также для выбора файла изображения. Создает следующее поле:

```
<input type="file">
```

По умолчанию использует виджет ClearableFileInput.

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    file= forms.ImageField (label="Изображение")
```

**forms.DateField():** предназначено для ввода 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">
```

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    date= forms.DateField(label="Введите дату:")
```

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

```
<input type="text">
```

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    time= forms.TimeField (label="Введите время:")
```

**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">
```

Пример:

```
from Django import forms

class UserForm(form.Form)

    date_time= forms.DateTimeField (label="Введите дату и
время:")
```

**forms.DurationField():** предназначен для ввода промежутка времени. 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">
```

По умолчанию использует виджет TextInput.

Пример:

```
from Django import forms

class UserForm(form.Form)

    time_delta= forms.DurationField (label="Введите промежуток
времени")
```

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

```
<input type="text" name="_0" >
```

```
<input type="text" name="_1" >
```

По умолчанию использует виджет SplitDateTimeWidget.

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    time_delta= forms.SplitDateTimeField (label="Введите дату и  
время")
```

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

```
<input type="number">
```

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    num= forms.IntegerField("label="Введите целое число")
```

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

```
<input type="number">
```

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    num= forms.DecimalField ("label="Введите десятичное число",  
decimal_places=2)
```

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

```
<input type="number">
```

По умолчанию использует виджет `NumberInput`.

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    num= forms.FloatField ("label="Введите число с плавающей точкой")
```

**forms.ChoiceField(choices=кортеж\_кортежей):** выбор данных из списка, 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>
```

По умолчанию использует виджет Select.

Пример:

```
from Django import forms
```

```
class UserForm(form.Form)
```

```
    country= forms.ChoiceField (label="Выберите страну",  
    choices=((1,Россия),(2,Франция),(3,Италия))
```

**forms.TypeChoiceField(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. Создает на HTML -странице разметку:

```
<select>
    <option value="1">Date 1</option>
    <option value="2">Date 2</option>
    <option value="3">Date 3</option>
</select>
```

По умолчанию использует виджет Select.

Пример:

```
from Django import forms
class UserForm(form.Form)
    city= forms.TypeChoiceField (label="Выберите город",
    empty_value=None,
    choices=((1,Москва),(2, Париж),(3,Милан))
```

**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.

По умолчанию использует виджет SelectMultiple.

### Пример:

```
from Django import forms

class UserForm(form.Form)

    countries= forms.MultipleChoiceField (label="Выберите
    страны",
    choices=((1,Россия),(2,Франция),(3,Италия),(4,Испания))
```

**forms.TypedMultipleChoiceField(choises=кортеж\_кортежей, coerce=функция\_преобразования, empty\_value=None):** аналог forms.TypeChoiceField для списка с множественным выбором.

По умолчанию использует виджет SelectMultiple.

### Пример:

```
from Django import forms

class UserForm(form.Form)

    countries= forms.MultipleChoiceField (label="Выберите
    страны",
    empty_value=None,
    choices=((1,Россия),(2,Франция),(3,Италия),(4,Испания))
```

## **Настройка формы и полей формы.**

Внешний вид формы и полей формы можно изменять с помощью некоторых параметров.

**label**



Свойство `label` позволяет установить текстовую метку, которая отображается рядом с полем. 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")
```

## **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 `TextInput`, который создает однострочное текстовое поле. Однако если нам надо создать многострочное текстовое поле, то необходимо воспользоваться виджетом `forms.Textarea`:

## **Initial - 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)
```

### **field\_order - порядок следования 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:

```
class UserForm(forms.Form):  
    name = forms.CharField()  
    age = forms.IntegerField()  
    field_order = ["age", "name"]
```

так и при определении объекта формы в представлении:

```
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 общее отображение формы:

**as\_table():** отображение в виде таблицы

**as\_ul():** отображение в виде списка

**as\_p():** каждое поле формы отображается в отдельном параграфе

## Применение методов:

```
<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>

```



The screenshot shows a web browser window with the title 'Django Forms' and the address '127.0.0.1:8000'. The page displays three Django form examples side-by-side:

- as\_table:** A form with two input fields labeled 'Name:' and 'Age:', followed by a 'Send' button.
- as\_ul:** A form with two input fields labeled 'Name:' and 'Age:', followed by a 'Send' button. The fields are preceded by bullet points.
- as\_p:** A form with two input fields labeled 'Name:' and 'Age:', followed by a 'Send' button.

## Валидация данных. Правила валидации.

### Обязательность ввода значения

Правила валидации задают параметры корректности вводимых данных. Например, для всех полей по умолчанию устанавливается обязательность ввода значения. При генерации html-кода для поля ввода устанавливается атрибут **required**. При попытке отправить форму с незаполненным полем мы получим ошибку. Чтобы этого не произошло, нужно 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 значение

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**

Проверку на валидность данных надо определять и на стороне сервера. Для этого у формы вызывается метод `is_valid()`, который возвращает `True`, если данные корректны, и `False` - если данные некорректны. Чтобы использовать этот метод, надо создать объект формы и передать ей пришедшие из запроса данные.

Определим 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

Django позволяет произвести детальную настройку полей формы.

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_имя_поля`.

**`form.название_поля.auto_id`:** возвращает `id` для поля, которое по умолчанию создается по схеме `id_имя_поля`.

**`form.название_поля.label_tag`:** возвращает элемент `label`, который представляет метку рядом с полем.

**`form.название_поля.help_text`:** возвращает текст подсказки, 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, ассоциированный с полем.

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

Возьмем простейшую форму (файл `forms.py`):

```
from django import forms

class UserForm(forms.Form):
    name = forms.CharField(label="Имя клиента")
    age = forms.IntegerField(label="Возраст клиента")
```

В представлении передадим эту форму в шаблон:

```
from django.shortcuts import render
from django.http import *
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, "firstapp/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>
```

Страница отобразит:

Имя клиента:

Возраст клиента:

Send

Введем значения в поля:

Имя клиента:

Возраст клиента:

Send

Нажимаем Send, получаем:

**Hello, Иван**

С помощью выражения `{% for field in form %}` мы можем пробегать по всем полям в форме и настраивать как само поле, так и

значения связанных с ним атрибутов - ошибок, текста подсказки, метки и т.д.

## Применение собственных стилей к полям форм

Кроме стилей, применяемых по умолчанию, Django содержит механизмы для применения к полям форм собственных стилей.

Например, каждое поле можно выводить вручную и определять правила стилизации для этого поля или окружающих его блоков.

Напишем форму:

```
from django import forms

class UserForm(forms.Form):

    name = forms.CharField(min_length=3)
    age = forms.IntegerField(min_value=1, max_value=100)
```

и создадим шаблон ее использования:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django Forms</title>
    <style>
        .alert{color:red}
        .form-group{margin: 10px 0;}
        .form-group input{width:250px;height: 25px;border-radius:3px;}
    </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>
```

Страница отобразит:

Name:

Age:

Send

При попытке отправить незаполненные поля получим сообщение:

Name:


This field is required.

Age:

This field is required.

Send

Заполним поля:

 Django Forms

×

+

←

→

↺

🏠

ⓘ

127.0.0.1:8000

Name:

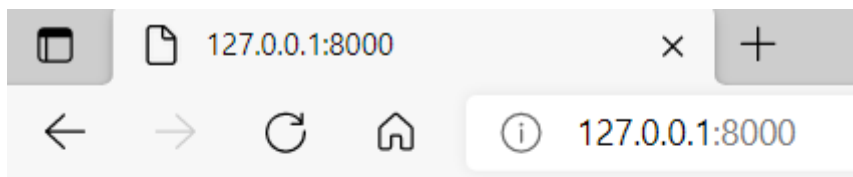
This field is required.

Age:

This field is required.

Send

И повторим отправку (нажмем Send). Получим:



**Hello, Илья**

Другой механизм представляют свойства формы **required\_css\_class** и **error\_css\_class**, которые 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 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>
```

Результат: страница отобразит:

**Name:**

**Age:**

При попытке ввести пустые значения получим:

**Name:** • This field is required.

**Age:** • This field is required.

Заполним поля:

**Name:** • This field is required.

**Age:** • This field is required.

и нажмем Send. Получим:

# Hello, Денис

Оба способа можно комбинировать. Изменим шаблон index.html:

```
<!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>
```

Результат:

**Name:**

**Age:**

Send

При попытке отправить пустые поля:

**Name:**

- This field is required.

**Age:**

- This field is required.

Send

Внесем данные:

**Name:**

- This field is required.

**Age:**

- This field is required.

Send

Отправим (Send). Результат:



# Hello, Мария

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`. Класс `myfield` будет определен в шаблоне:

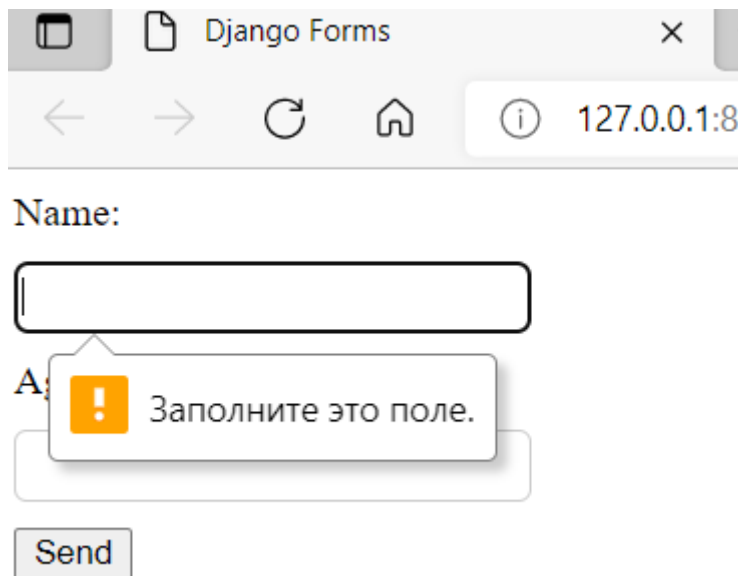
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <title>Django Forms</title>
    <style>
        .myfield{
            border:1px solid #ccc;
            border-radius:5px;
            height:25px;
            width:200px;
            margin: 10px 10px 10px 0;
        }
    </style>
</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>
```

Name:

Age:

Send

При попытке отправить пустые поля будет выведено сообщение:  
“Заполните это поле”.



The screenshot shows a web browser window titled "Django Forms" with the address bar displaying "127.0.0.1:8000". The form contains two input fields, "Name:" and "Age:", both of which are empty. A red border highlights the "Name:" field, and a red error message box with an exclamation mark icon appears below it, stating "Заполните это поле." (Fill in this field). The "Send" button is visible at the bottom of the form.

Внесем данные и отправим:

Name:

Алексей

Age:

32

Send

Результат:

**Hello, Алексей**