

# Лекция 4. Формы

## Часть 2



## 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. Например:

```
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, установленный 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 функции-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>
```





Django Forms

127.0.0.1:8000

as\_table

Name:

Age:

Send

as\_ul

Name:

Age:

Send

as\_p

Name:

Age:

Send



## 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 функцию-представление в файле views.py:

```
from django.shortcuts import render
from django.http import HttpResponseRedirect
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 HttpResponseRedirect("<h2>Hello, {0}</h2>".format(name))
        else:
            return HttpResponseRedirect("Invalid data")
    else:
        userform = UserForm()
        return render(request, "index.html", {"form": userform})
```



Если приходит POST-запрос, то в начале заполняем форму 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")
```



При тестировании формы можно установить у нее атрибут **novalidate**. Это логический атрибут. Если он присутствует, то указывает, что данные формы (ввод) не должны проверяться при отправке:

```
<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.



Возьмем нашу форму (файл forms.py):

```
forms.py
1 from django import forms
2
3
4 class UserForm(forms.Form):
5     name = forms.CharField(label="Имя клиента")
6     age = forms.IntegerField(label="Возраст клиента")
```

Создадим шаблон index.html :

```
index.html
6 </head>
7 <body>
8     <form method="POST" novalidate>
9         {% csrf_token %}
10        <div>
11            {% for field in form %}
12                <div class="form-group">
13                    {{field.label_tag}}
14                    <div>{{field}}</div>
15                    <div class="error">{{field.errors}}</div>
16                </div>
17            {% endfor %}
18        </div>
19        <input type="submit" value="Send" >
20    </form>
21 </body>
22 </html>
```

В функции-представлении передадим эту форму в шаблон:

```
views.py
11 from django.http import *
12 from .forms import UserForm
13
14
15 from django.shortcuts import render
16 from django.http import *
17 from .forms import UserForm
18
19
20 def index(request):
21     userform = UserForm()
22     if request.method == "POST":
23         userform = UserForm(request.POST)
24         if userform.is_valid():
25             name = userform.cleaned_data["name"]
26             return HttpResponse("<h2>Hello, {0}</h2>".format(name))
27     return render(request, "index.html", {"form": userform})
```



Django Forms x +

127.0.0.1:8000

Имя клиента:

Возраст клиента:

Send



Django Forms x +

127.0.0.1:8000

Имя клиента:

Возраст клиента:

Send

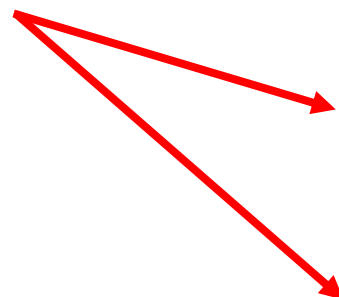


127.0.0.1:8000 x +

127.0.0.1:8000

**Hello, Петр**

При попытке отправить незаполненные поля  
получим сообщение:



Django Forms x +

127.0.0.1:8000

Имя клиента:

Возраст клиента:

Send

• This field is required.

• This field is required.

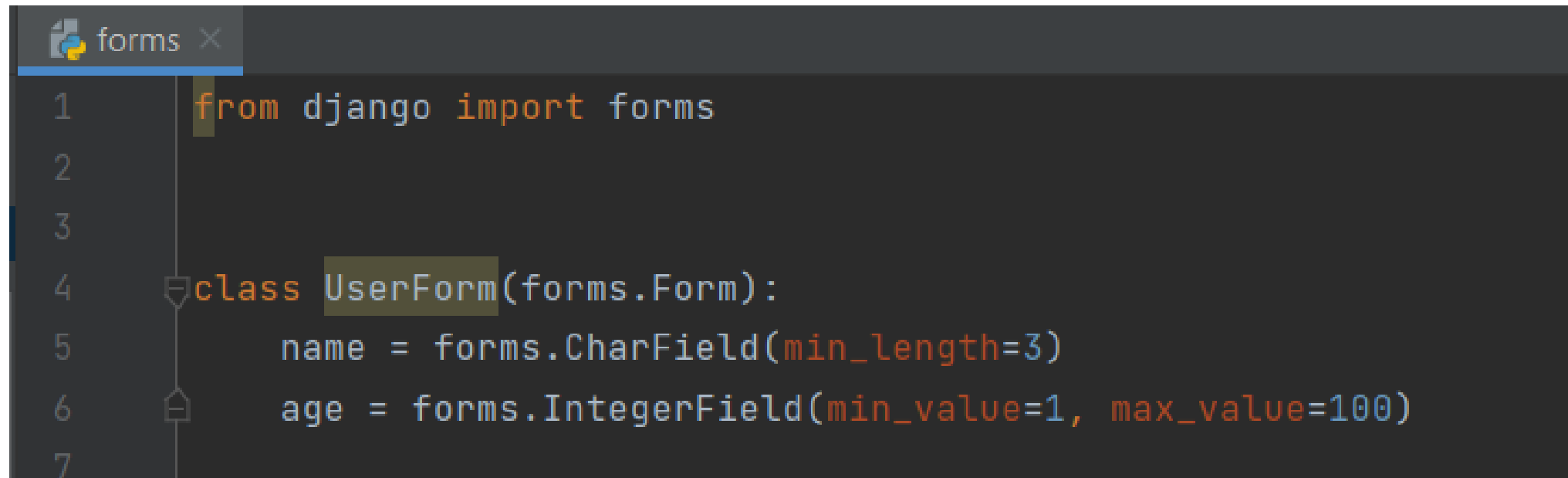


## Применение собственных стилей к полям форм

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.

Создадим форму:



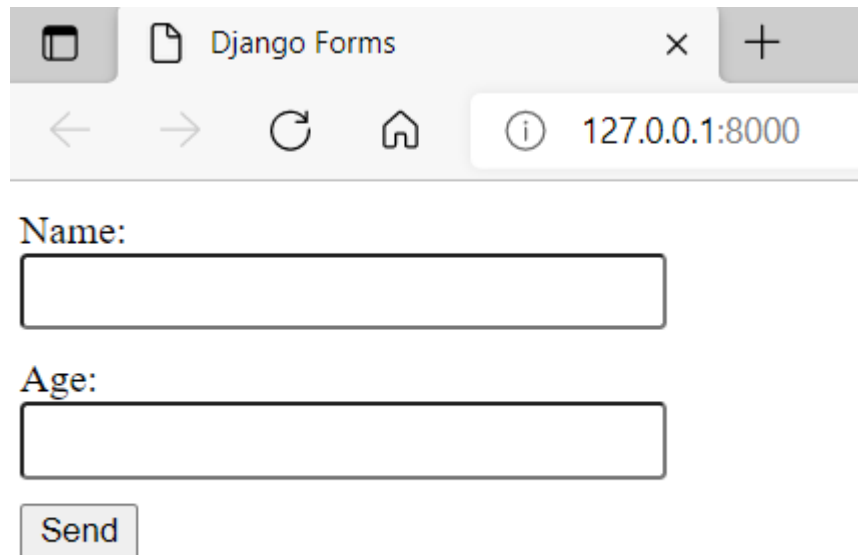
```
forms x
1  from django import forms
2
3
4  class UserForm(forms.Form):
5      name = forms.CharField(min_length=3)
6      age = forms.IntegerField(min_value=1, max_value=100)
7
```



создадим шаблон ее использования

Index.html.

Перейдем на сервер:



Browser window showing the Django Forms application running on 127.0.0.1:8000. The page displays a form with two input fields: "Name:" and "Age:", and a "Send" button.

```
<!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>
```



Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

При попытке отправить незаполненные поля  
получим сообщение:

127.0.0.1:8000 x +

127.0.0.1:8000

**Hello, Ангела**

Django Forms x +

127.0.0.1:8000

Name:

This field is required.

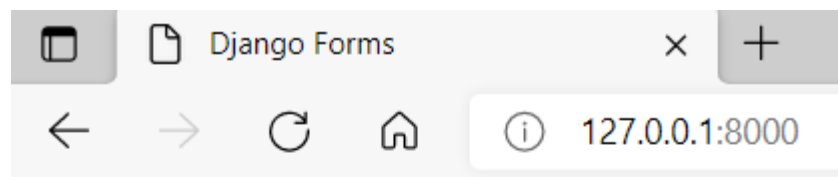
Age:

This field is required.

Send



Заполним поля и повторим отправку.



Name:

Илья

This field is required.

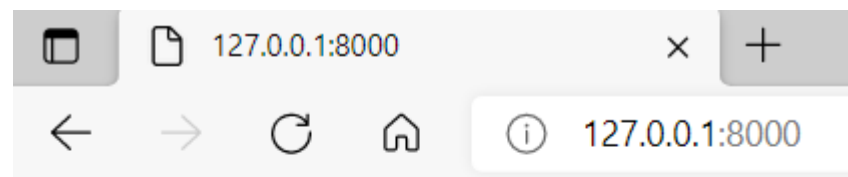
Age:

23

This field is required.

Send

Получим:



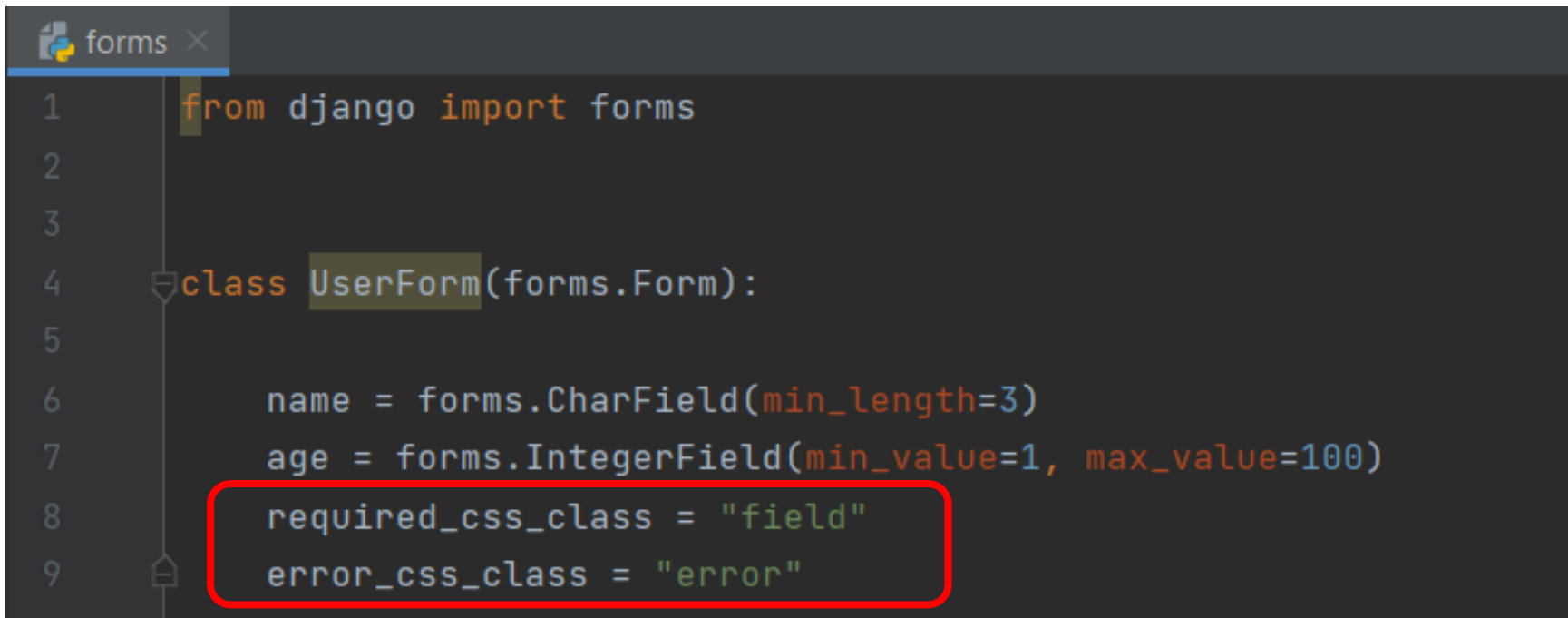
**Hello, Илья**





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:



```
forms ×
1  from django import forms
2
3
4  class UserForm(forms.Form):
5
6      name = forms.CharField(min_length=3)
7      age = forms.IntegerField(min_value=1, max_value=100)
8      required_css_class = "field"
9      error_css_class = "error"
```



В этом случае в шаблоне **index.html** должны быть определены или подключены классы **"field"** и **"error"**:

```
index x
3  <head>
4      <meta charset="utf-8" />
5      <title>Django Forms</title>
6      <style>
7          .field{font-weight:bold;}
8          .error{color:red;}
9      </style>
10 </head>
11 <body class="container">
12     <form method="POST" novalidate>
13         {% csrf_token %}
14         <table>
15             {{form}}
16         </table>
17         <input type="submit" value="Send" >
18     </form>
19 </body>
20 </html>
```



Страница отобразит:

Django Forms

127.0.0.1:8000

Name:

Age:

Send

Django Forms

127.0.0.1:8000

Name:

Age:

Send

127.0.0.1:8000

Hello, Анна

При попытке отправить пустые поля получим:      Заполним поля и отправим:

Django Forms

127.0.0.1:8000

Name:

• This field is required.

Age:

• This field is required.

Send

Django Forms

127.0.0.1:8000

Name:

• This field is required.

Age:

• This field is required.

Send

127.0.0.1:8000

Hello, Гаврила



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:

Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

127.0.0.1:8000 x +

127.0.0.1:8000

Hello, Светлана

Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

127.0.0.1:8000 x +

127.0.0.1:8000

Hello, Елена

• This field is required.

Age:

• This field is required.

Send

• This field is required.

Age:

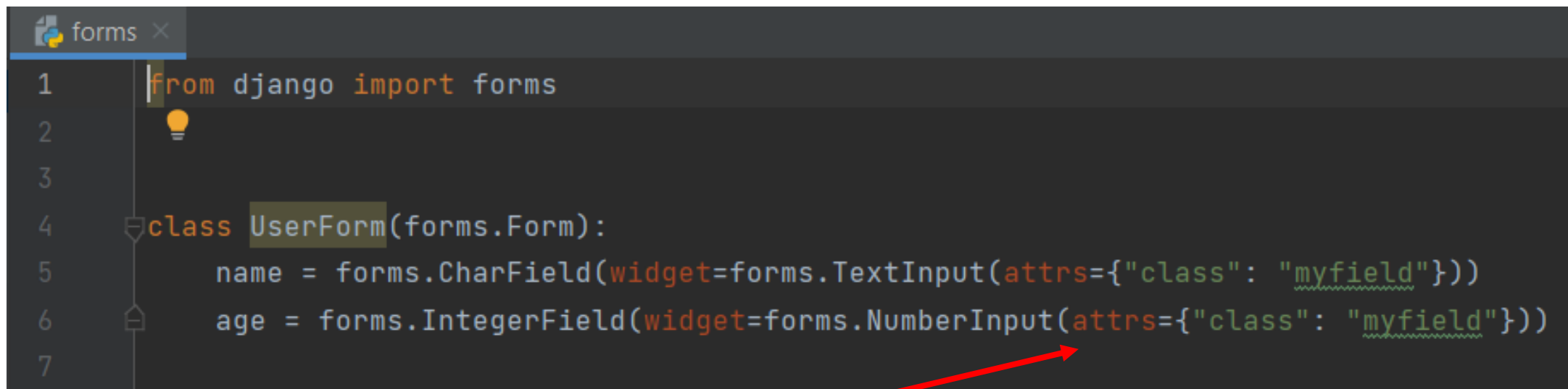
• This field is required.

Send



## Третий механизм стилизации представляет установка классов и стилей через виджеты

Редактируем форму:



```
1 from django import forms
2
3
4 class UserForm(forms.Form):
5     name = forms.CharField(widget=forms.TextInput(attrs={"class": "myfield"}))
6     age = forms.IntegerField(widget=forms.NumberInput(attrs={"class": "myfield"}))
7
```

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** определим 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>
```

Шаблон index.html

Определение класса  
**myfield** в шаблоне



Django Forms x +

127.0.0.1:8000

Name:

Age:

Send

Django Forms x +

127.0.0.1:8000

Name:

Николай

Age:

37

Send

127.0.0.1:8000 x +

127.0.0.1:8000

**Hello, Николай**

Django Forms x +

127.0.0.1:8000

Name:

Age:



Заполните это поле.

Send

Django Forms x +

127.0.0.1:8000

Name:

Александр

Age:

36

Send

127.0.0.1:8000 x +

127.0.0.1:8000

**Hello, Александр**





Спасибо за внимание!

