Django Form

Django’s form functionality can simplify and automate vast portions of work like data prepared for display in a form, rendered as HTML, edit using a convenient interface, returned to the server, validated and cleaned up etc and can also do it more securely than most programmers would be able to do in code they wrote themselves.

Django handles three distinct parts of the work involved in forms:

- preparing and restructuring data to make it ready for rendering
- creating HTML forms for the data
- receiving and processing submitted forms and data from the client

Bound and Unbound Forms

If it’s bound to a set of data, it’s capable of validating that data and rendering the form as HTML with the data displayed in the HTML.

If it’s unbound, it cannot do validation (because there’s no data to validate!), but it can still render the blank form as HTML.

Create Django Form using Form Class

To create Django form we have to create a new file inside application folder lets say file name is **forms.py**. Now we can write below code inside **forms.py** to create a form:-

Syntax:-

```
from django import forms
class FormClassName(forms.Form):
        label=forms.FieldType()
        label=forms.FieldType(label='display_label')
```

Example:-

```
from django import forms
class StudentRegistration(forms.Form):
        name=forms.CharField()              # Here length is not required
        email=forms.EmailField()
```

```
<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_email">Email:</label></th>
  <td><input type="email" name="email" required
            id="id_email"></td>
</tr>
```

Display Form to User

- Create an object of Form class in **views.py** then pass object to template files
- Use Form object in template file

Creating Form object in views.py

First of all create form object inside **views.py** file then pass this object to template file as a dict.

views.py

```
from .forms import StudentRegistration
def showformdata(request):
    fm = StudentRegistration()
    return render(request, ‘enroll/userregistration.html’, {‘form’:fm})
```

Get object from views.py in template file

templates/enroll / userregistration.html

```
<!DOCTYPE html>
<html>
	<body>
		{{form}}
	</body>
</html>
```

```
<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_email">Email:</label></th>
  <td><input type="email" name="email" required
            id="id_email"></td>
</tr>
```

Note - Form object won't provide form tag and button you have to write them manually in template file.

Get object from views.py in template file

{{form}} doesn’t add form tag and button so we have to add them manually.

templates/enroll / userregistration.html

```
<!DOCTYPE html>
<html>
	<body>
		<form action =“” method=“get”>
			{{form}}
			<input type=“submit” value=“Submit”>
		</form>
	</body>
</html>
```

```
<form action =“” method=“get”>
	<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_email">Email:</label></th>
	  <td><input type=“email" name="email" required
			id="id_email"></td>
	</tr>
	<input type=“submit” value=“Submit”>
</form>
```

- The output does not include the <table> and </table> tags, nor does it include the <form> and </form> tags or an <input type="submit"> tag. It’s your job to do that.
- Each field type has a default HTML representation. CharField is represented by an <input type="text"> and EmailField by an <input type="email">.
- The HTML *name* for each tag is taken directly from its attribute name in the StudentRegistration class.
- The text label for each field e.g. ‘Name:’ and ‘Email:' is generated from the field name by converting all underscores to spaces and upper-casing the first letter.
- Each text label is surrounded in an HTML <label> tag, which points to the appropriate form field via its id.
- Its id, in turn, is generated by prepending 'id_' to the field name. The id attributes and <label> tags are included in the output by default.
- The output uses HTML5 syntax, targeting <!DOCTYPE html>.

Form Rendering Options

- {{form}} will render them all
- {{ form.as_table }} will render them as table cells wrapped in <tr> tags
- {{ form.as_p }} will render them wrapped in <p> tags
- {{ form.as_ul }} will render them wrapped in <li> tags
- {{ form.name_of_field }} will render field manually as given

```
<form action =“” method=“get”>
        {{form}}
       <input type=“submit” value=“Submit”>
</form>
```

```
<form action =“” method=“get”>
        {{form.as_p}}
       <input type=“submit” value=“Submit”>
</form>
```

Configure id attribute

auto_id – The id attribute values are generated by prepending id_ to the form field names. This behavior is configurable, though, if you want to change the id convention or remove HTML id attributes and <label> tags entirely.

Use the auto_id argument to the Form constructor to control the id and label behavior. This argument must be True, False or a string. By default, auto_id is set to the string 'id_%s'.

Example:-

fm = StudentRegistration(auto_id=False)

fm = StudentRegistration(auto_id=‘geeky’)

- If auto_id is set to a string containing the format character '%s', then the form output will include <label> tags, and will generate id attributes based on the format string.
- If auto_id is set to True, then the form output will include <label> tags and will use the field name as its id for each form field.
- If auto_id is False, then the form output will not include <label> tags nor id attributes.
- If auto_id is set to any other true value – such as a string that doesn’t include %s – then the library will act as if auto_id is True.

Configure label tag

label_suffix - A translatable string (defaults to a colon (:) in English) that will be appended after any label name when a form is rendered.

It’s possible to customize that character, or omit it entirely, using the label_suffix parameter.

The label suffix is added only if the last character of the label isn’t a punctuation character (in English, those are ., !, ? or :)

fm = StudentRegistration(label_suffix=‘ ’)

Dynamic initial values

initial - initial is used to declare the initial value of form fields at runtime.

To accomplish this, use the initial argument to a Form. This argument, if given, should be a dictionary mapping field names to initial values. Only include the fields for which you’re specifying an initial value; it’s not necessary to include every field in your form.

If a Field defines initial and you include initial when instantiating the Form, then the latter initial will have precedence.

Ordering Form Field

order_fields(field_order) – This method is used to rearrange the fields any time with a list of field names as in field_order. By default Form.field_order=None, which retains the order in which you define the fields in your form class.

If field_order is a list of field names, the fields are ordered as specified by the list and remaining fields are appended according to the default order.

Unknown field names in the list are ignored.

fm = StudentRegistration()

fm.order_fields(field_order=[‘email’, ‘name’])