The nice thing about Django forms is that we can either define one from scratch or create a ModelFormwhich will save the result of the form to the model.

This is exactly what we want to do: we will create a form for our Post model.

Like every important part of Django, forms have their own file: forms.py.

We need to create a file with this name in the blog directory.

We need to import Django forms first (from django import forms) and our Post model (from .models import Post).

PostForm, as you probably suspect, is the name of our form. We need to tell Django that this form is a ModelForm (so Django will do some magic for us) – forms.ModelForm is responsible for that.

Next, we have class Meta, where we tell Django which model should be used to create this form (model = Post).

Finally, we can say which field(s) should end up in our form. In this scenario we want only title and text to be exposed – author should be the person who is currently logged in (you!) and created\_date should be automatically set when we create a post (i.e. in the code), right?

And that's it! All we need to do now is use the form in a view and display it in a template.

So once again we will create a link to the page, a URL, a view and a template.

**Building a form in Django[¶](https://docs.djangoproject.com/en/2.1/topics/forms/" \l "building-a-form-in-django" \o "Permalink to this headline)**

**The**[**Form**](https://docs.djangoproject.com/en/2.1/ref/forms/api/#django.forms.Form)**class**[**¶**](https://docs.djangoproject.com/en/2.1/topics/forms/#the-form-class)

We already know what we want our HTML form to look like. Our starting point for it in Django is this:

forms.py[**¶**](https://docs.djangoproject.com/en/2.1/topics/forms/#id1)

**from** **django** **import** forms

**class** **NameForm**(forms.Form):

your\_name = forms.CharField(label='Your name', max\_length=100)

This defines a [**Form**](https://docs.djangoproject.com/en/2.1/ref/forms/api/#django.forms.Form) class with a single field (**your\_name**). We’ve applied a human-friendly label to the field, which will appear in the **<label>** when it’s rendered (although in this case, the [**label**](https://docs.djangoproject.com/en/2.1/ref/forms/fields/#django.forms.Field.label) we specified is actually the same one that would be generated automatically if we had omitted it).

The field’s maximum allowable length is defined by **[max\_length](https://docs.djangoproject.com/en/2.1/ref/forms/fields/" \l "django.forms.CharField.max_length" \o "django.forms.CharField.max_length)**. This does two things. It puts a **maxlength="100"** on the HTML **<input>** (so the browser should prevent the user from entering more than that number of characters in the first place). It also means that when Django receives the form back from the browser, it will validate the length of the data.

A [**Form**](https://docs.djangoproject.com/en/2.1/ref/forms/api/#django.forms.Form) instance has an **[is\_valid()](https://docs.djangoproject.com/en/2.1/ref/forms/api/" \l "django.forms.Form.is_valid" \o "django.forms.Form.is_valid)** method, which runs validation routines for all its fields. When this method is called, if all fields contain valid data, it will:

* return **True**
* place the form’s data in its **[cleaned\_data](https://docs.djangoproject.com/en/2.1/ref/forms/api/" \l "django.forms.Form.cleaned_data" \o "django.forms.Form.cleaned_data)** attribute.

The whole form, when rendered for the first time, will look like:

<**label** for="your\_name">Your name: </**label**>

<**input** id="your\_name" type="text" name="your\_name" maxlength="100" required>

Note that it **does not** include the **<form>** tags, or a submit button. We’ll have to provide those ourselves in the template.

**The view**[**¶**](https://docs.djangoproject.com/en/2.1/topics/forms/#the-view)

Form data sent back to a Django website is processed by a view, generally the same view which published the form. This allows us to reuse some of the same logic.

To handle the form we need to instantiate it in the view for the URL where we want it to be published:

views.py[**¶**](https://docs.djangoproject.com/en/2.1/topics/forms/#id2)

**from** **django.http** **import** HttpResponseRedirect

**from** **django.shortcuts** **import** render

**from** **.forms** **import** NameForm

**def** get\_name(request):

*# if this is a POST request we need to process the form data*

**if** request.method == 'POST':

*# create a form instance and populate it with data from the request:*

form = NameForm(request.POST)

*# check whether it's valid:*

**if** form.is\_valid():

*# process the data in form.cleaned\_data as required*

*# ...*

*# redirect to a new URL:*

**return** HttpResponseRedirect('/thanks/')

*# if a GET (or any other method) we'll create a blank form*

**else**:

form = NameForm()

**return** render(request, 'name.html', {'form': form})

If we arrive at this view with a **GET** request, it will create an empty form instance and place it in the template context to be rendered. This is what we can expect to happen the first time we visit the URL.

If the form is submitted using a **POST** request, the view will once again create a form instance and populate it with data from the request: **form = NameForm(request.POST)** This is called “binding data to the form” (it is now a *bound* form).

We call the form’s **is\_valid()** method; if it’s not **True**, we go back to the template with the form. This time the form is no longer empty (*unbound*) so the HTML form will be populated with the data previously submitted, where it can be edited and corrected as required.

If **is\_valid()** is **True**, we’ll now be able to find all the validated form data in its **cleaned\_data** attribute. We can use this data to update the database or do other processing before sending an HTTP redirect to the browser telling it where to go next.

**The template**[**¶**](https://docs.djangoproject.com/en/2.1/topics/forms/#the-template)

We don’t need to do much in our **name.html** template. The simplest example is:

<**form** action="/your-name/" method="post">

{% **csrf\_token** %}

{{ form }}

<**input** type="submit" value="Submit">

</**form**>

All the form’s fields and their attributes will be unpacked into HTML markup from that **{{ form }}** by Django’s template language.

There are other output options though for the **<label>**/**<input>** pairs:

* **{{ 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

Note that you’ll have to provide the surrounding **<table>** or **<ul>** elements yourself.

Here’s the output of **{{ form.as\_p }}** for our **ContactForm** instance: