# Forms

1. A [**Form**](https://docs.djangoproject.com/en/2.1/ref/forms/api/#django.forms.Form) class describes a form and determines how it works and appears. A form class’s fields map to HTML form **<input>** elements.

A form’s fields are themselves classes, they manage form data and perform validation when a form is submitted.

A form field is represented to a user in the browser as an HTML “widget” - a piece of user interface machinery. Each field type has an appropriate default [Widget class](https://docs.djangoproject.com/en/2.1/ref/forms/widgets/), but these can be overridden as required.

Forms in djnago is written in forms.py file:



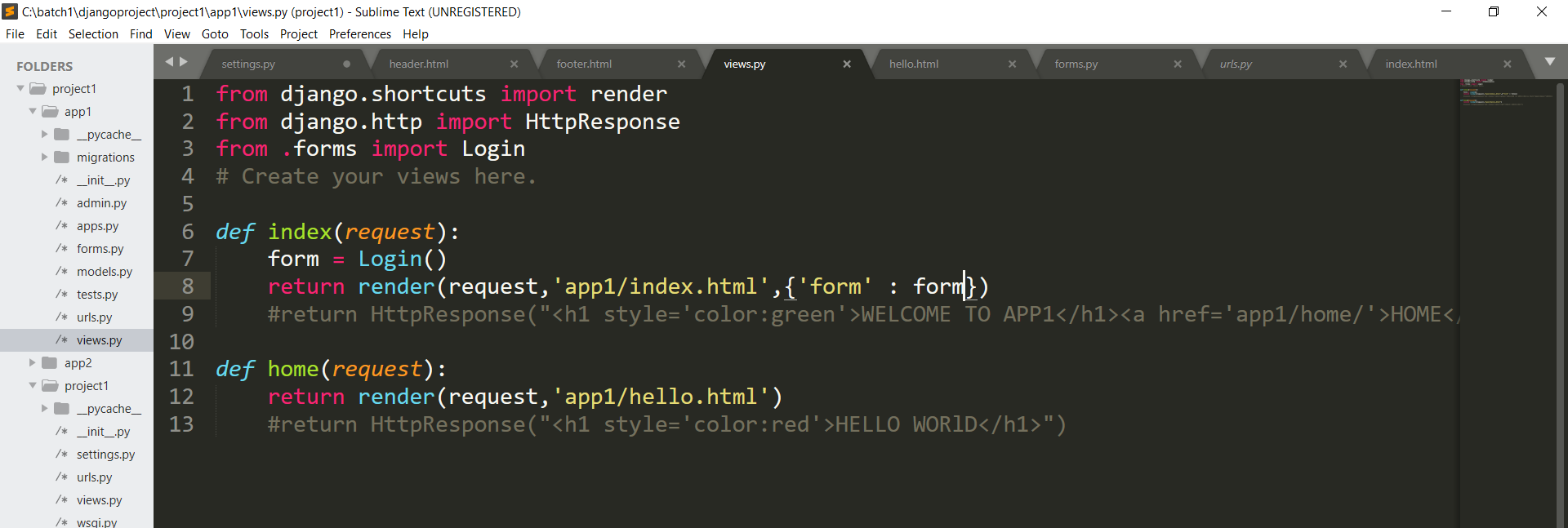
In this form we have defined ‘Login’ class for login form.

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.

The field used is CharField which is same as in html form <input type=”text”>

1. Form data sent back to a Django website is processed by a view i.e in views.py file.

First import the class ‘Login’ in views.py to create an instance of that class.



Then render it to a template in the form of a dictionary as a key value pair. Inn above code ‘form’(on left side) is key and ‘form’(on right side) is its value which is an instance of Login class.

1. 

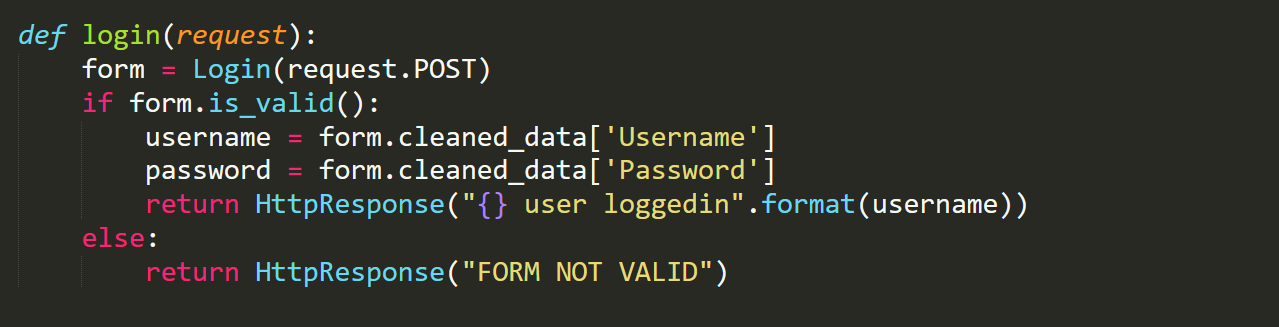
All the form’s fields and their attributes will be unpacked into HTML markup from that **{{ form }}** by Django’s template language.

Csrf\_token stands for cross site verification token.

1. When the form is submitted the action goes to ‘login/’.The url is defined in app1.urls.



1. In app1.views the ‘login’ function is defined as follows:



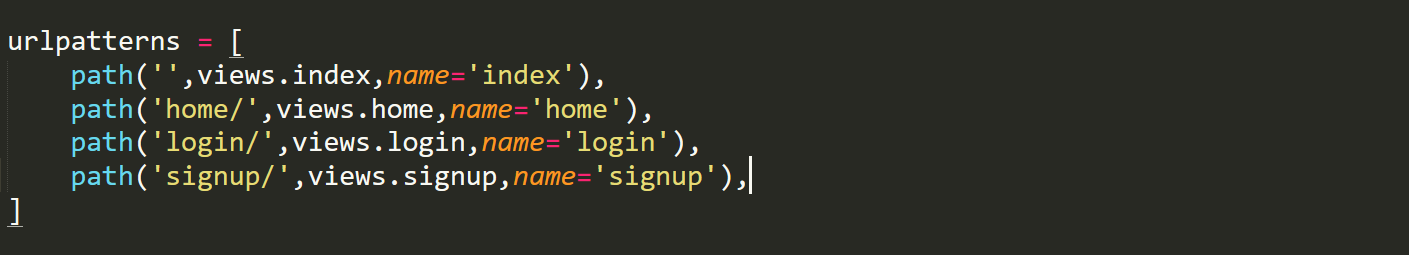
If we arrive at this view with a **GET** request, it will create an empty form instance and place it in the template context to be rendered. This is what we can expect to happen the first time we visit the URL.

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.

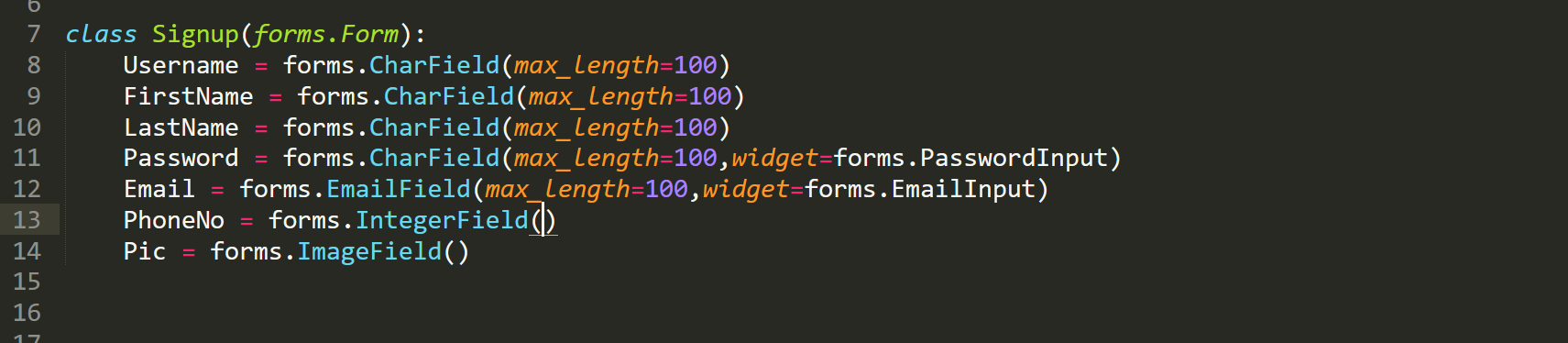
If **is\_valid()** is **True**, we’ll now be able to find all the validated form data in its **cleaned\_data** attribute. We can use this data to update the database or do other processing before sending an HTTP redirect to the browser telling it where to go next.

1. Similarly the signup form can be made and can be rendered.

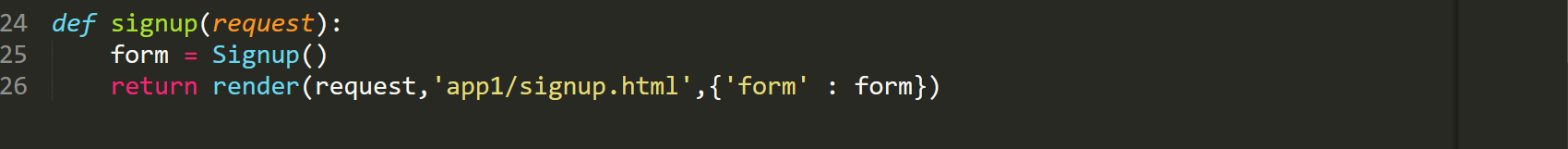


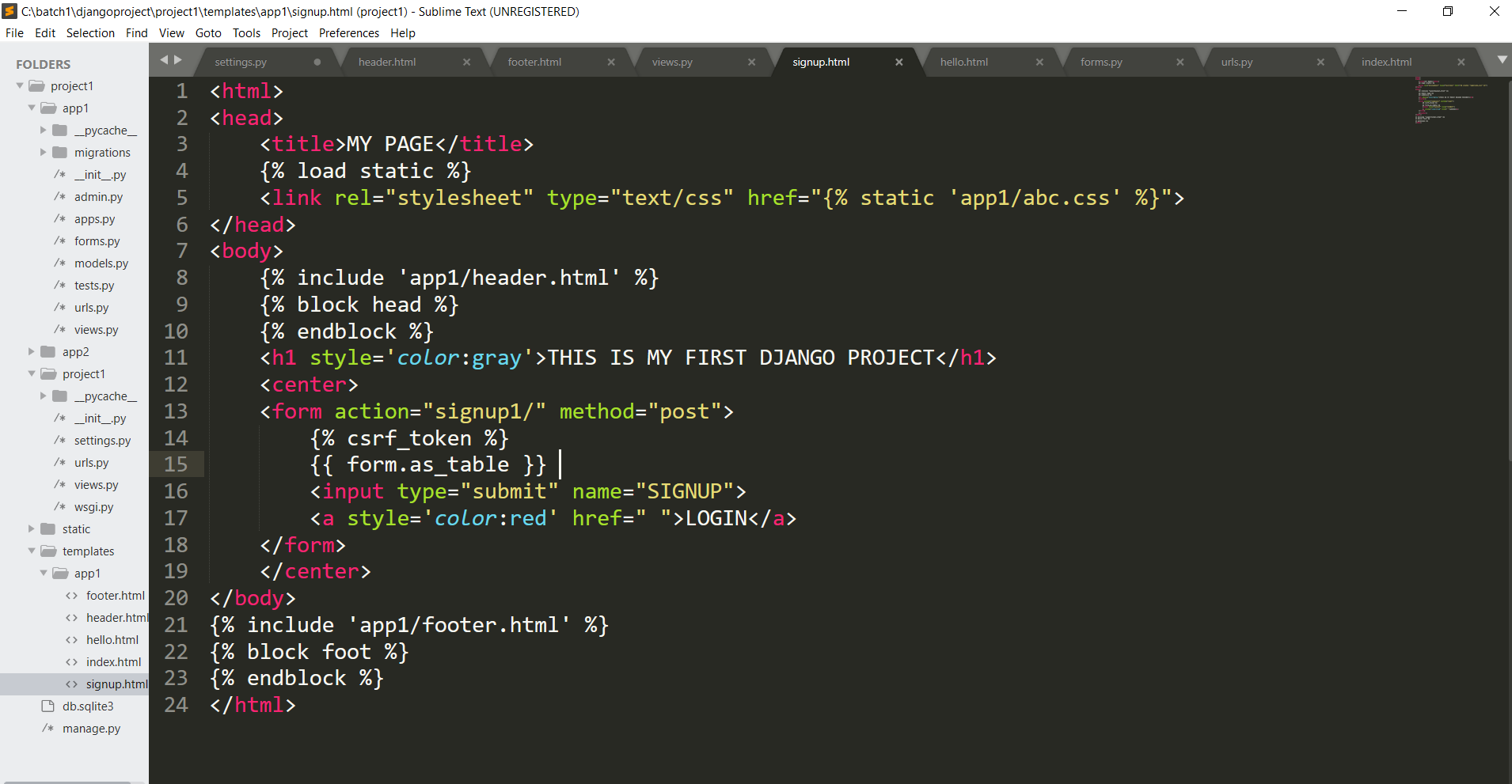
When user will request for ‘signup/’ url the signup form will be rendered.

In app1.urls the url for ‘signup/’ is defined.

1. 

In app1.forms ‘signup’ class is defined.





In the above file {{ form.as\_table }} will display the form like in html <table> tag is used.

Similarly {{ form.as\_p }} is used to display it in paragraph form.

And

{{ form.as\_ul }} display the form like in html <ul> tag is used(unordered list).