You've started the Django development server, a lightweight web server written purely in Python. We've included this with Django so you can develop things rapidly, without having to deal with configuring a production server – such as Apache – until you're ready for production.

Now's a good time to note: **don't** use this server in anything resembling a production environment. It's intended only for use while developing. (We're in the business of making web frameworks, not web servers.)

(To serve the site on a different port, see the **`runserver`** reference.)

Automatic reloading of `runserver`

The development server automatically reloads Python code for each request as needed. You don't need to restart the server for code changes to take effect. However, some actions like adding files don't trigger a restart, so you'll have to restart the server in these cases.

Creating the Polls app ¶

Now that your environment – a "project" – is set up, you're set to start doing work.

Each application you write in Django consists of a Python package that follows a certain convention. Django comes with a utility that automatically generates the basic directory structure of an app, so you can focus on writing code rather than creating directories.

Projects vs. apps

What's the difference between a project and an app? An app is a web application that does something – e.g., a blog system, a database of public records or a small poll app. A project is a collection of configuration and apps for a particular website. A project can contain multiple apps. An app can be in multiple projects.

Your apps can live anywhere in your Python path. In this tutorial, we'll create our poll app inside the **`djangotutorial`** folder.

To create your app, make sure you're in the same directory as **`manage.py`** and type this command:

```
$ python manage.py startapp polls
```

That'll create a directory **`polls`**, which is laid out like this:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    views.py
```

This directory structure will house the poll application.

Write your first view ¶

Let's write the first view. Open the file **`polls/views.py`** and put the following Python code in it:

polls/views.py ¶

```python
from django.http import HttpResponse


def index(request):
    return HttpResponse("Hello, world. You're at the polls index.")
```

This is the most basic view possible in Django. To access it in a browser, we need to map it to a URL - and for this we need to define a URL configuration, or "URLconf" for short. These URL configurations are defined inside each Django app, and they are Python files named **`urls.py`**.

To define a URLconf for the **`polls`** app, create a file **`polls/urls.py`** with the following content:

polls/urls.py ¶

```python
from django.urls import path

from . import views

urlpatterns = [
    path("", views.index, name="index"),
]
```

Your app directory should now look like:

```
polls/
    __init__.py
    admin.py
    apps.py
    migrations/
        __init__.py
    models.py
    tests.py
    urls.py
    views.py
```

The next step is to configure the global URLconf in the **`mysite`** project to include the URLconf defined in **`polls.urls`**. To do this, add an import for **`django.urls.include`** in **`mysite/urls.py`** and insert an **`include()`** in the **`urlpatterns`** list, so you have:

mysite/urls.py ¶

```python
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path("polls/", include("polls.urls")),
    path("admin/", admin.site.urls),
]
```

The **`path()`** function expects at least two arguments: **`route`** and **`view`**. The **`include()`** function allows referencing other URLconfs. Whenever Django encounters **`include()`**, it chops off whatever part of the URL matched up to that point and sends the remaining string to the included URLconf for further processing.

The idea behind **`include()`** is to make it easy to plug-and-play URLs. Since polls are in their own URLconf (**`polls/urls.py`**), they can be placed under "/polls/", or under "/fun_polls/", or under "/content/polls/", or any other path root, and the app will still work.

When to use `include()`

You should always use **`include()`** when you include other URL patterns. The only exception is **`admin.site.urls`**, which is a pre-built URLconf provided by Django for the default admin site.

You have now wired an **`index`** view into the URLconf. Verify it's working with the following command:

```
$ python manage.py runserver
```

Go to http://localhost:8000/polls/ in your browser, and you should see the text *"Hello, world. You're at the polls index."*, which you defined in the **`index`** view.

Page not found?

If you get an error page here, check that you're going to http://localhost:8000/polls/ and not http://localhost:8000/.