Writing your first Django app, part 1?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.If it isn't, you'll get an error telling "No module named django". This tutorial is written for Django 5.1, which supports Python 3.10 and later. If the Django version doesn't match, you can refer to the tutorial for your version of Django by using the version switcher at the bottom right corner of this page, or update Django to the newest version. If you're using an older version of Python, check What Python version can I use with Django? to find a compatible version of Django. See How to install Django for advice on how to remove older versions of Django and install a newer one. Where to get help: If you're having trouble going through this tutorial, please head over to the Getting Help section of the FAQ. Creating a project? If this is your first time using Django, you'll have to take care of some initial setup. Namely, you'll need to auto-generate some code that establishes a Django project – a collection of settings for an instance of Django, including database configuration, Django-specific options and application-specific settings. From the command line, cd into a directory where you'd like to store your code, then run the following command: ?/? ? $ django-admin startproject mysite This will create a mysite directory in your current directory. If it didn't work, see Problems running django-admin. Note You'll need to avoid naming projects after built-in Python or Django components. In particular, this means you should avoid using names like django (which will conflict with Django itself) or test (which conflicts with a built-in Python package). Let's look at what startproject created: mysite/ manage.py mysite/ __init__.py settings.py urls.py asgi.py wsgi.py These files are: The outer mysite/ root directory is a container for your project. Its name doesn't matter to Django; you can rename it to anything you like. manage.py: A command-line utility that lets you interact with this Django project in various ways. You can read all the details about manage.py in django-admin and manage.py. The inner mysite/ directory is the actual Python package for your project. Its name is the Python package name you'll need to use to import anything inside it (e.g. mysite.urls). mysite/__init__.py: An empty file that tells Python that this directory should be considered a Python package. If you're a Python beginner, read more about packages in the official Python docs. mysite/settings.py: Settings/configuration for this Django project. Django settings will tell you all about how settings work. mysite/urls.py: The URL declarations for this Django project; a "table of contents" of your Django-powered site.You'll see a "Congratulations!" page, with a rocket taking off. It worked! 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.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? 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.from django.contrib import admin from django.urls import include, path urlpatterns = [path("polls/", include("polls.urls")), path("admin/", admin.site.urls),] 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.mysite/asgi.py: An entry-point for ?????ASGI-compatible web servers to serve your project.The development server