



Tutorial: Get started with the Django web framework in Visual Studio

Article • 09/08/2023

Applies to:  Visual Studio  Visual Studio for Mac  Visual Studio Code

Django is a high-level Python framework designed for rapid, secure, and scalable web development. This tutorial explores the Django framework in the context of the project templates. Visual Studio provides the project templates to streamline the creation of Django-based web apps.

In this tutorial, you learn how to:

- Create a basic Django project in a Git repository using the "Blank Django Web Project" template (step 1).
- Create a Django app with one page and render that page using a template (step 2).
- Serve static files, add pages, and use template inheritance (step 3).
- Use the Django Web Project template to create an app with multiple pages and responsive design (step 4).
- Authenticate users (step 5).

Prerequisites

- Visual Studio 2022 on Windows with the following options:
 - The **Python development** workload (**Workload** tab in the installer). For more instructions, see [Install Python support in Visual Studio](#).
 - **Git for Windows** on the **Individual components** tab under **Code tools**.

Django project templates also include earlier versions of Python Tools for Visual Studio. The template details might differ from what's discussed in this tutorial (especially different with earlier versions of the Django framework).

Python development isn't presently supported in Visual Studio for Mac. On Mac and Linux, use the [Python extension in Visual Studio Code](#) .

"Visual Studio projects" and "Django projects"

In Django terminology, a "Django project" has several site-level configuration files along with one or more "apps." To create a full web application, you can deploy these apps to a web host. A Django project can contain multiple apps, and the same app can be in multiple Django projects.

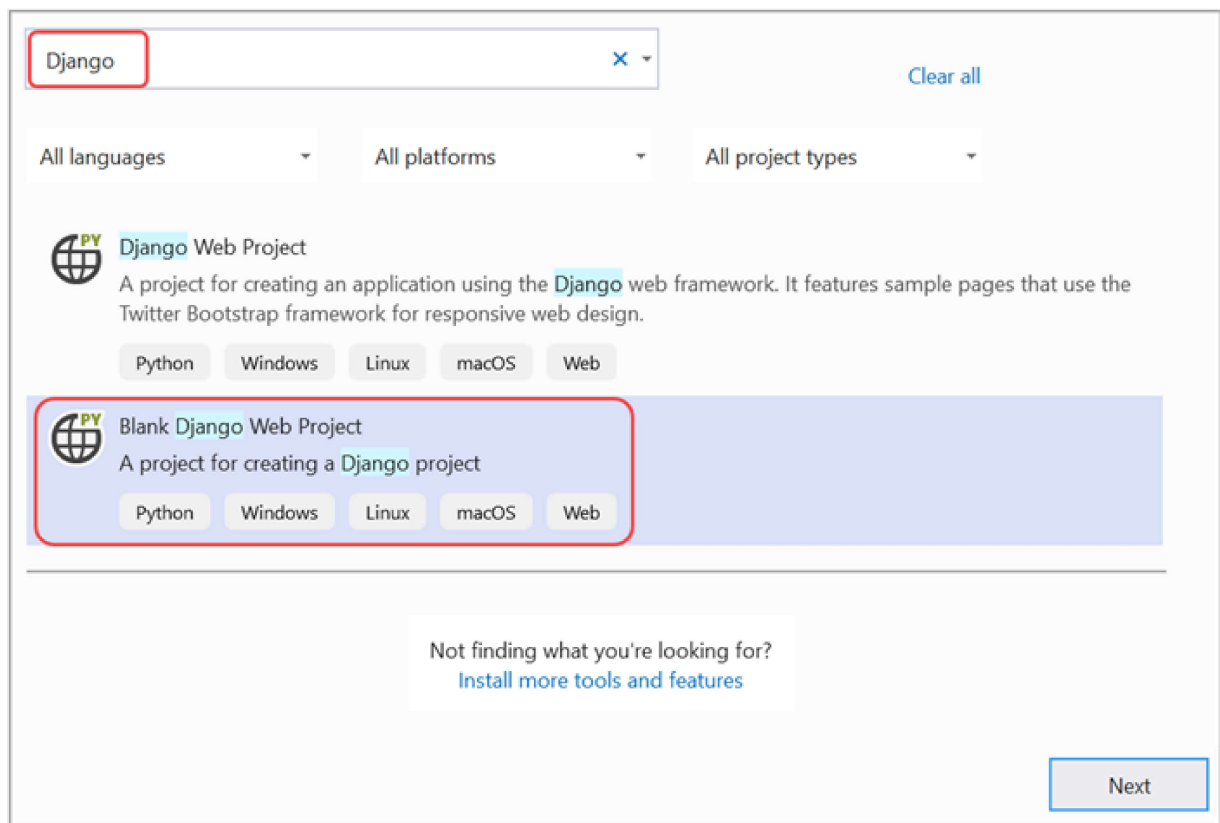
A Visual Studio project can contain the Django project along with multiple apps. Whenever this tutorial refers to just a "project," it's referring to the Visual Studio project. When it refers to the "Django project" portion of the web application, it's referring to a "Django project" specifically.

Over the course of this tutorial, you'll create a single Visual Studio solution that contains three separate Django projects. Each project contains a single Django app. You can easily switch between different files for comparison, by keeping the projects in the same solution.

Step 1-1: Create a Visual Studio project and solution

When working with Django from the command line, you usually start a project by running the `django-admin startproject <project_name>` command. In Visual Studio, the "Blank Django Web Project" template provides the same structure within a Visual Studio project and solution.

1. In Visual Studio, select **File > New > Project**, search for "Django", and select the **Blank Django Web Project** template, then select **Next**.



2. Enter the following information and then select **Create**:

- **Project Name:** Set the name of the Visual Studio project to **BasicProject**. This name is also used for the Django project.
- **Location:** Specify a location in which to create the Visual Studio solution and project.
- **Solution:** Leave this control set to default **Create new solution** option.
- **Solution name:** Set to **LearningDjango**, which is appropriate for the solution as a container for multiple projects in this tutorial.

Step 1-2: Examine the Git controls and publish to a remote repository

In this step, you'll familiarize yourself with Visual Studio's Git controls and **Team Explorer**. With the **Team Explorer** window, you'll work with the source control.

1. To commit the project to your local source control:
 - a. Select **Add to Source Control** command at the bottom corner of the Visual Studio main window.
 - b. Then, select the **Git** option.

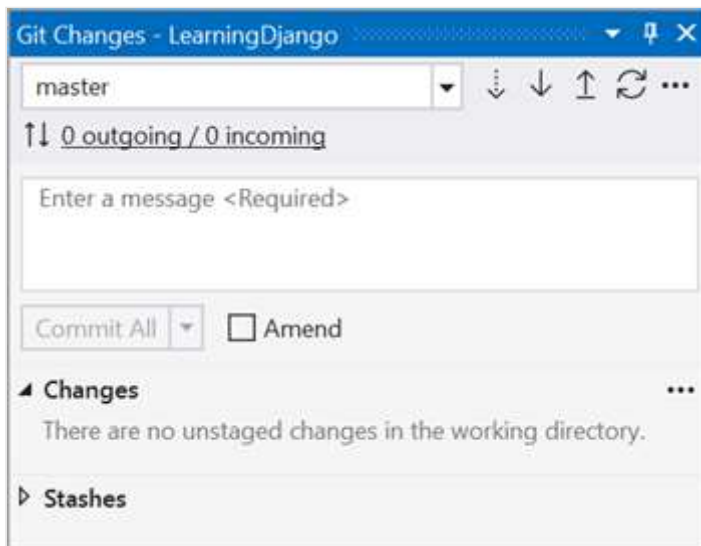
c. Now, you're taken to the **Create Git repository** window, where you can create and push a new repository.



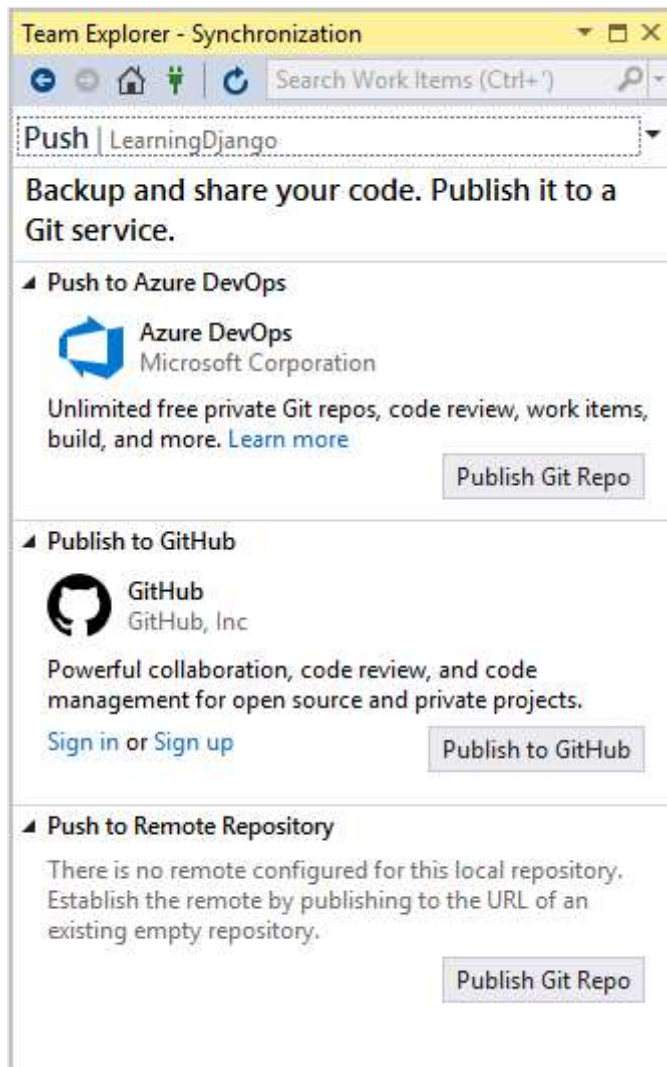
2. After creating a repository, a set of new Git controls appears at the bottom. From left to right, these controls show unpushed commits, uncommitted changes, the current branch, and the name of the repository.



3. Select the **Git changes** button. Visual Studio then opens the **Git Changes** page in **Team Explorer**. You don't see any pending changes as the newly created project is already committed to source control automatically.

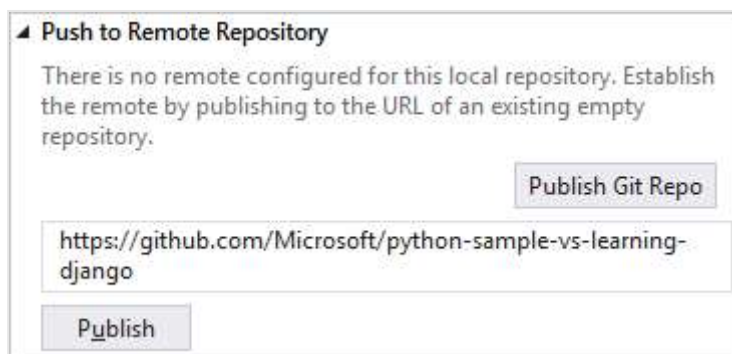


4. On the Visual Studio status bar, select the unpushed commits button (the up arrow with 2) to open the **Synchronization** page in **Team Explorer**. The **Synchronization** page provides easy options to publish the local repository to different remote repositories.



You can choose any service you want for your projects. This tutorial shows the use of GitHub, where the completed sample code for the tutorial is maintained in the [Microsoft/python-sample-vs-learning-django](https://github.com/Microsoft/python-sample-vs-learning-django) repository.

5. When selecting any of the **Publish** controls, **Team Explorer** prompts you for more information. For example, when publishing the sample for this tutorial, the repository itself has to be created first, in which case, the **Push to Remote Repository** option was used with the repository's URL.



If you don't have an existing repository, the **Publish to GitHub** and **Push to Azure DevOps** options let you create one directly from within Visual Studio.

6. As you work through this tutorial, get into the habit of periodically using the controls in Visual Studio to commit and push changes. This tutorial reminds you at appropriate points.

Tip

To quickly navigate within **Team Explorer**, select the header (that reads **Changes** or **Push** in the images above) to see a pop-up menu of the available pages.

Question: What are some advantages of using source control from the beginning of a project?

Answer: Source control from the start, especially if you also use a remote repository, provides a regular offsite backup of your project. Unlike maintaining a project on a local file system, source control provides a complete change history and the easy ability to revert a single file or the whole project to its previous state. The change history helps determine the cause of regressions (test failures). When multiple people are working on a project, the source control manages the overwrite and provides conflict resolution.

Lastly, source control, which is fundamentally a form of automation, sets you up for automating builds, testing, and release management. It's the first step in using DevOps for a project. There's really no reason to not use source control from the beginning as the barriers to entry are low.

For further discussion on source control as automation, see [The Source of Truth: The Role of Repositories in DevOps](#), an article in MSDN Magazine written for mobile apps that also applies to web apps.

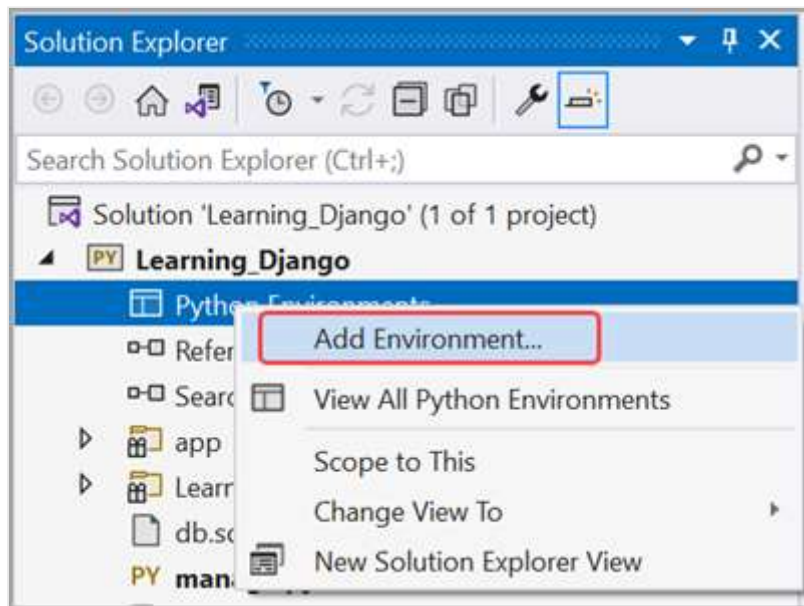
Question: Can I prevent Visual Studio from autocommitting a new project?

Answer: Yes. To disable autocommit, go to the **Settings** page in **Team Explorer**. Select **Git > Global settings**, clear the option labeled **Commit changes after merge by default**, and then select **Update**.

Step 1-3: Create the virtual environment and exclude it from source control

Now that you've configured source control for your project, you can create the virtual environment that contains the necessary Django packages for the project. You can then use **Team Explorer** to exclude the environment's folder from source control.

1. In **Solution Explorer**, right-click the **Python Environments** node and select **Add Environment**.



2. Select **Create** to accept the defaults, in Add Virtual Environment dialog. (You can change the name of the virtual environment if you want, which just changes the name of its subfolder, but `env` is a standard convention.)

Project
LearningDjango

Name
env

Base interpreter
Python 3.9 (64-bit)

Location
C:\Users\user-name\source\repos\LearningDjango\
[Change virtual environment location](#)

Install packages from file (optional)
C:\Users\user-name\source\repos\LearningDjango\LearningDjango\requirements.txt ...

☒ Set as current environment
☐ Set as default environment for new projects
☐ View in Python environments window
☐ Make this environment available globally

Description for this environment

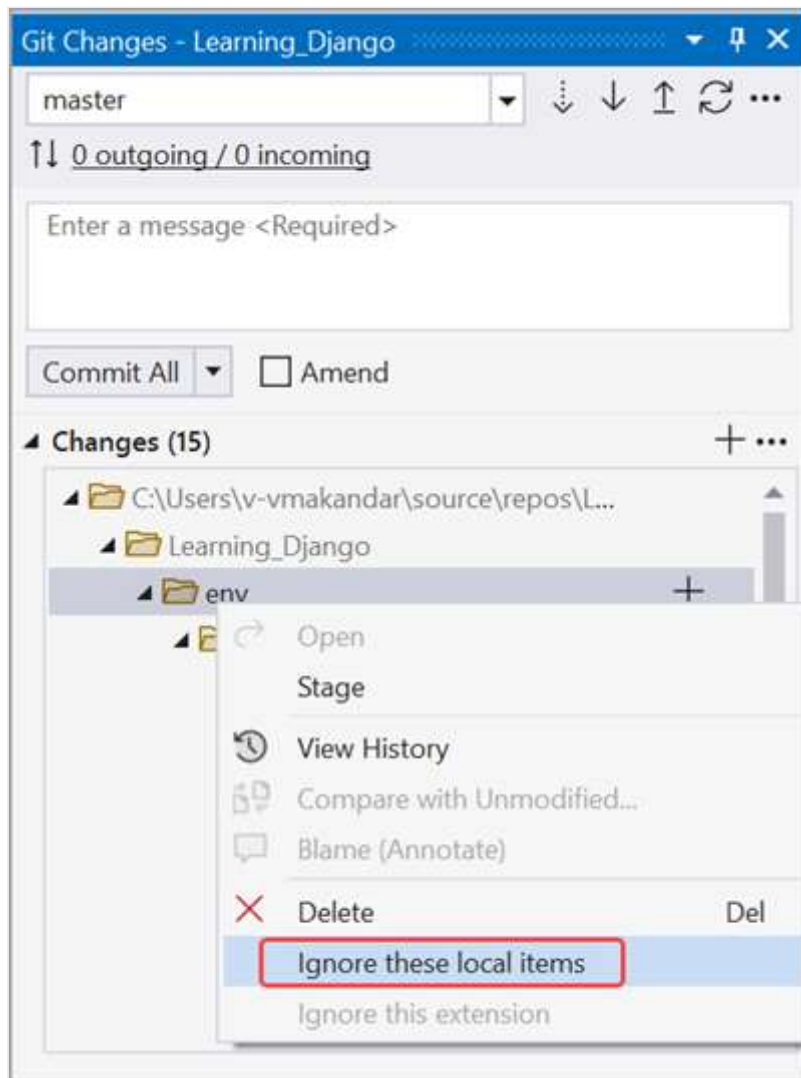
[ts?](#) **Create** Cancel

3. Consent to administrator privileges if prompted, then wait a few minutes while Visual Studio downloads and installs packages. During this time, thousands of files are transferred to many subfolders. You can see the progress in the Visual Studio **Output** window. While you're waiting, ponder the Question sections that follow.

4. On the Visual Studio Git controls (on the status bar), select the changes indicator (that shows **99***) which opens the **Changes** page in **Team Explorer**.

Creation of the virtual environment brought in thousands of changes, but you don't need to include any of them in source control because you (or anyone else cloning the project) can always recreate the environment from *requirements.txt*.

5. To exclude the virtual environment, right-click the **env** folder and select **Ignore these local items**.



6. After excluding the virtual environment, the only remaining changes are to the project file and *.gitignore* file. The *.gitignore* file contains an added entry for the virtual environment folder. You can double-click the file to see a diff.
7. Enter a commit message and select the **Commit All** button, then push the commits to your remote repository.

Question: Why do I want to create a virtual environment?

Answer: A virtual environment is a great way to isolate your app's exact dependencies. Such isolation avoids conflicts within a global Python environment, and aids both testing and collaboration. Over time, as you develop an app, you invariably bring in many helpful Python packages. You can easily update the project's *requirements.txt* file by keeping packages in a project-specific virtual environment. The *requirements.txt* file describes the environment, which is included in the source control. When the project is copied to any other computers, including build servers, deployment servers, and other development

computers, it's easy to recreate the environment using only *requirements.txt* (which is why the environment doesn't need to be in source control). For more information, see [Use virtual environments](#).

Question: How do I remove a virtual environment that's already committed to source control?

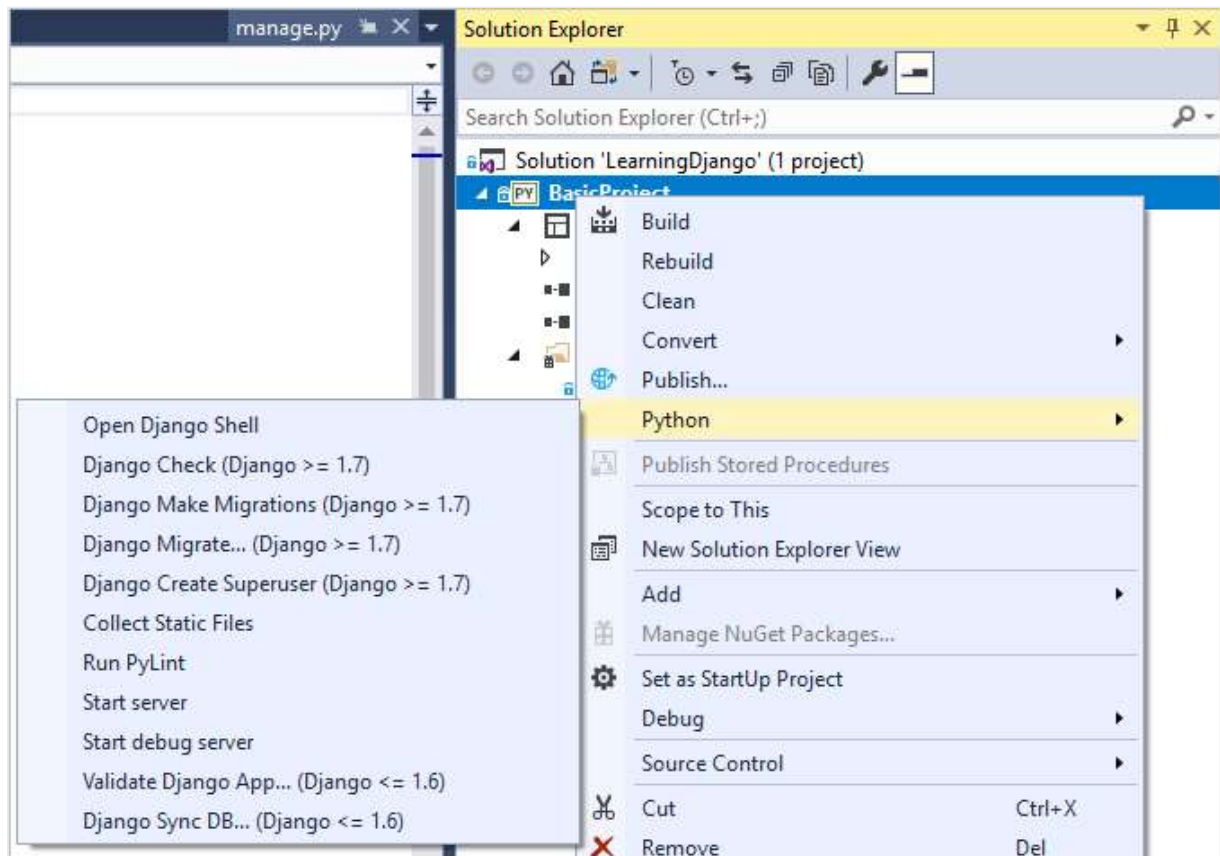
Answer: First, edit your *.gitignore* file to exclude the folder. Find the section at the end with the comment `# Python Tools for Visual Studio (PTVS)` and add a new line for the virtual environment folder, such as `/BasicProject/env`. (Visual Studio doesn't show the file in **Solution Explorer**. To open the file directly, go to **File > Open > File**. You can also open the file from **Team Explorer**. Go to the **Settings** page and select **Repository Settings**. Now, navigate to the **Ignore & Attributes Files** section and select the **Edit** link next to *.gitignore*.)

Second, open a command window, navigate to a folder such as *BasicProject*. The *BasicProject* folder contains the virtual environment folder such as *env*, and run `git rm -r env`. Then commit those changes from the command line (`git commit -m 'Remove venv'`) or commit from the **Changes** page of **Team Explorer**.

Step 1-4: Examine the boilerplate code

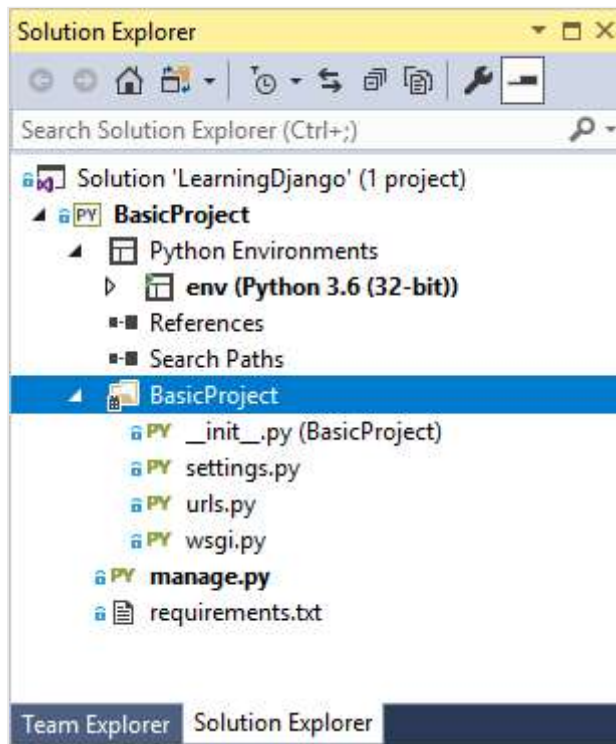
Once project creation completes, examine the boilerplate Django project code (which is again the same as generated by the CLI command `django-admin startproject <project_name>`).

1. The root of your project has *manage.py*, the Django command-line administrative utility that Visual Studio automatically sets as the project startup file. You run the utility on the command line using `python manage.py <command> [options]`. For common Django tasks, Visual Studio provides convenient menu commands. Right-click the project in **Solution Explorer** and select **Python** to see the list. You'll come across some of these commands in the course of this tutorial.



2. In your project, there's a folder with the same name as the project. It contains the basic Django project files:

- *__init.py*: An empty file that tells Python that this folder is a Python package.
- *settings.py*: Contains settings for Django project, which you'll modify in the course of developing a web app.
- *urls.py*: Contains a table of contents for the Django project, which you'll also modify in the course of development.
- *wsgi.py*: An entry point for WSGI-compatible web servers to serve your project. You usually leave this file as-is, as it provides the hooks for production web servers.



3. As noted earlier, the Visual Studio template also adds a *requirements.txt* file to your project specifying the Django package dependency. The presence of this file is what invites you to create a virtual environment when first creating the project.

Question: Can Visual Studio generate a *requirements.txt* file from a virtual environment after I install other packages?

Answer: Yes. Expand the **Python Environments** node, right-click your virtual environment, and select the **Generate requirements.txt** command. It's good to use this command periodically as you modify the environment, and commit changes to *requirements.txt* to source control along with any other code changes that depend on that environment. If you set up continuous integration on a build server, you should generate the file and commit changes whenever you modify the environment.

Step 1-5: Run the empty Django project

1. In Visual Studio, select **Debug > Start Debugging (F5)** or use the **Web Server** button on the toolbar (the browser you see might vary):



2. Running the server means to run the command `manage.py runserver <port>`, which starts Django's built-in development server. If Visual Studio says **Failed to start debugger** with a message about having no startup file, right-click **manage.py** in **Solution Explorer** and select **Set as Startup File**.
3. When you start the server, you see a console window opens that also displays the server log. Visual Studio automatically opens a browser to `http://localhost:<port>`. Since the Django project has no apps, Django shows only a default page to confirm that what you have so far is working fine.



4. When you're done, stop the server by closing the console window, or by using the **Debug > Stop Debugging** command in Visual Studio.

Question: Is Django a web server and a framework?

Answer: Yes and no. Django has a built-in web server that's used for development purposes. This web server is used when you run the web app locally, such as when debugging in Visual Studio. When you deploy to a web host, however, Django uses the host's web server instead. The `wsgi.py` module in the Django project takes care of hooking into the production servers.

Question: What's the difference between using the Debug menu commands and the server commands on the project's Python submenu?

Answer: In addition to the **Debug** menu commands and toolbar buttons, you can also launch the server using the **Python > Run server** or **Python > Run debug server** commands on the project's context menu. Both commands open a console window in which you'll see the local URL (`localhost:port`) for the running server. However, you must manually open a browser with that URL, and running the debug server doesn't automatically start the Visual Studio debugger. If you want, you can attach a debugger to the running process by using the **Debug > Attach to Process** command.

Next steps

At this point, the basic Django project doesn't contain any apps. You'll create an app in the next step. Since you'll work with Django apps more than the Django project, you won't need to know more about the boilerplate files for now.

Create a Django app with views and page templates

Go deeper

- Django project code: [Writing your first Django app, part 1](https://docs.djangoproject.com/en/4.2/topics/newapp/) (docs.djangoproject.com)
- Administrative utility: [django-admin and manage.py](https://docs.djangoproject.com/en/4.2/topics/admin/) (docs.djangoproject.com)
- Tutorial source code on GitHub: [Microsoft/python-sample-vs-learning-django](https://github.com/microsoft/python-sample-vs-learning-django)