

Learn Python with Django



Django?

Django is a high-level Python web framework that enables rapid development of secure and maintainable websites. Built by experienced developers, Django takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It is free and open source, has a thriving and active community, great documentation, and many options for free and paid-for support.

Django helps you write software that is:

Complete

Django follows the "Batteries included" philosophy and provides almost everything developers might want to do "out of the box". Because everything you need is part of the one "product", it all works seamlessly together, follows consistent design principles, and has extensive and up-to-date documentation.

Versatile

Django can be (and has been) used to build almost any type of website — from content management systems and wikis, through to social networks and news sites. It can work with any client-side framework, and can deliver content in almost any format (including HTML, RSS feeds, JSON, XML, etc). The site you are currently reading is based on Django!

Internally, while it provides choices for almost any functionality you might want (e.g. several popular databases, templating engines, etc.), it can also be extended to use other components if needed.

Secure

Django helps developers avoid many common security mistakes by providing a framework that has been engineered to "do the right things" to protect the website automatically. For example, Django provides a secure way to manage user accounts and passwords, avoiding common mistakes like putting session information in cookies where it is vulnerable (instead cookies just contain a key, and the actual data is stored in the database) or directly storing passwords rather than a password hash.

A password hash is a fixed-length value created by sending the password through a cryptographic hash function. Django can check if an entered password is correct by running it through the hash function and comparing the output to the stored hash value. However due to the "one-way" nature of the function, even if a stored hash value is compromised it is hard for an attacker to work out the original password.

Django enables protection against many vulnerabilities by default, including SQL injection, cross-site scripting, cross-site request forgery and clickjacking (see Website security for more details of such attacks).

Scalable

Django uses a component-based "shared-nothing" architecture (each part of the architecture is independent of the others, and can hence be replaced or changed if needed). Having a clear separation between the different parts means that it can scale for increased traffic by adding hardware at any level: caching servers, database servers, or application servers. Some of the busiest sites have successfully scaled Django to meet their demands (e.g. Instagram and Disqus, to name just two).

Maintainable

Django code is written using design principles and patterns that encourage the creation of maintainable and reusable code. In particular, it makes use of the Don't Repeat Yourself (DRY) principle so there is no unnecessary duplication, reducing the amount of code. Django also promotes the grouping of related functionality into reusable "applications" and, at a lower level, groups related code into modules (along the lines of the Model View Controller (MVC) pattern).

Portable

Django is written in Python, which runs on many platforms. That means that you are not tied to any particular server platform, and can run your applications on many flavours of Linux, Windows, and Mac OS X. Furthermore, Django is well-supported by many web hosting providers, who often provide specific infrastructure and documentation for hosting Django sites.

Where did it come from?

Django was initially developed between 2003 and 2005 by a web team who were responsible for creating and maintaining newspaper websites. After creating a number of sites, the team began to factor out and reuse lot of common code and design patterns. This common code evolved into a generic web development framework, which was open-sourced as the "Django" project in July 2005.

Django has continued to grow and improve, from its first milestone release (1.0) in September 2008 through to the recently-released version 2.0 (2017). Each release has added new functionality and bug fixes, ranging from support for new types of databases, template engines, and caching, through to the addition of "generic" view functions and classes (which reduce the amount of code that developers have to write for a number of programming tasks).

Note: Check out the release notes on the Django website to see what has changed in recent versions, and how much work is going into making Django better.

Django is now a thriving, collaborative open source project, with many thousands of users and contributors. While it does still have some features that reflect its origin, Django has evolved into a versatile framework that is capable of developing any type of website.

How popular is Django?

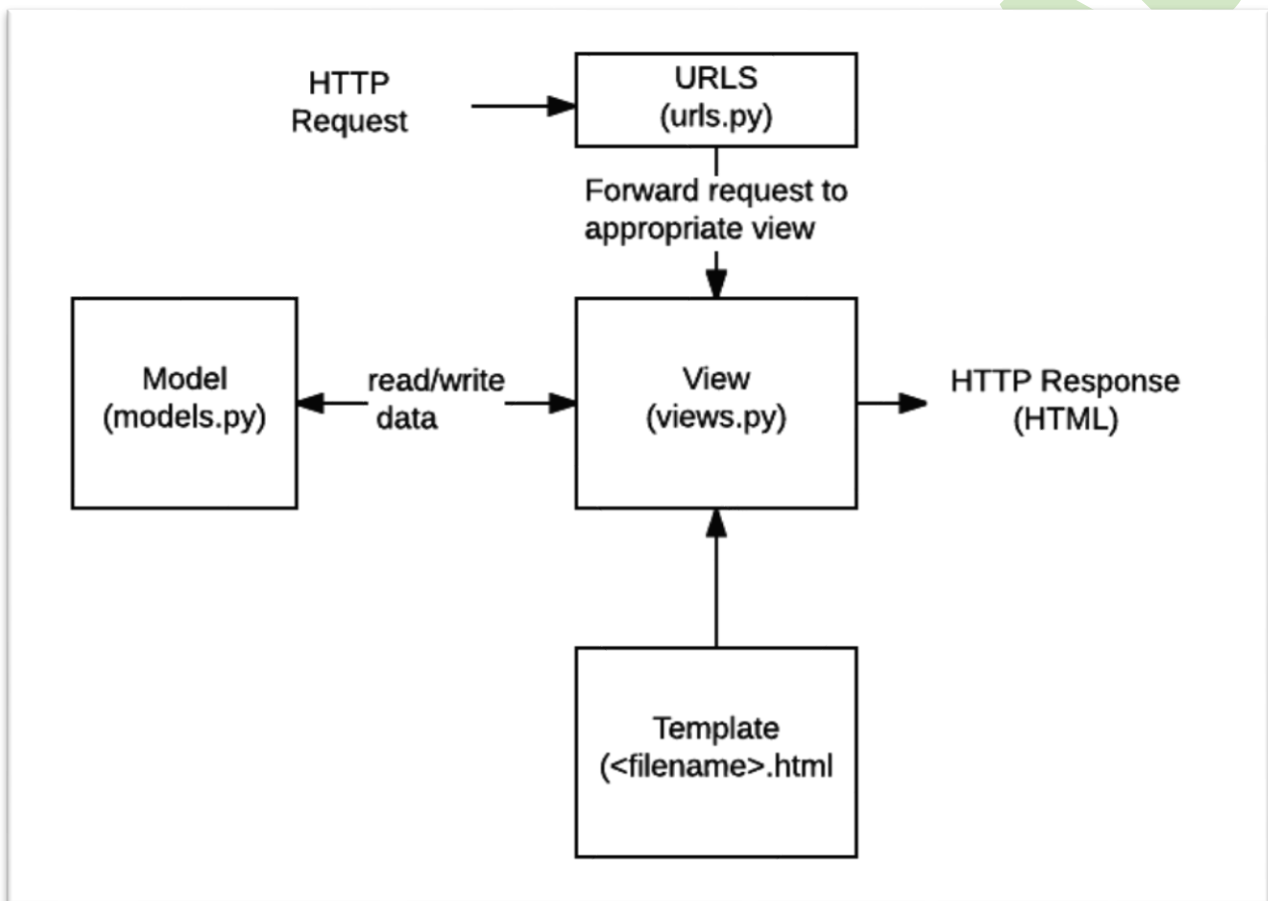
There isn't any readily-available and definitive measurement of popularity of server-side frameworks (although sites like Hot Frameworks attempt to assess popularity using mechanisms like counting the number of GitHub projects and Stack Overflow questions for each platform). A better question is whether Django is "popular enough" to avoid the problems of unpopular platforms. Is it continuing to evolve? Can you get help if you need it? Is there an opportunity for you to get paid work if you learn Django?

Based on the number of high profile sites that use Django, the number of people contributing to the codebase, and the number of people providing both free and paid for support, then yes, Django is a popular framework!

High-profile sites that use Django include: Discus, Instagram, Knight Foundation, MacArthur Foundation, Mozilla, National Geographic, Open Knowledge Foundation, Pinterest, and Open Stack (source: Django home page).

What does Django code look like?

In a traditional data-driven website, a web application waits for HTTP requests from the web browser (or other client). When a request is received the application works out what is needed based on the URL and possibly information in POST data or GET data. Depending on what is required it may then read or write information from a database or perform other tasks required to satisfy the request. The application will then return a response to the web browser, often dynamically creating an HTML page for the browser to display by inserting the retrieved data into placeholders in an HTML template. Django web applications typically group the code that handles each of these steps into separate files:



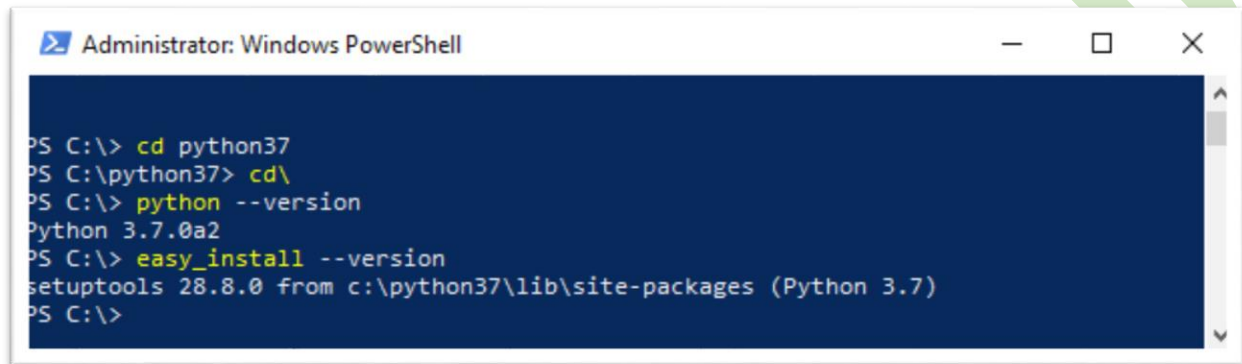
- **URLs:** While it is possible to process requests from every single URL via a single function, it is much more maintainable to write a separate view function to handle each resource. A URL mapper is used to redirect HTTP requests to the appropriate view based on the request URL. The URL mapper can also match particular patterns of strings or digits that appear in an URL, and pass these to a view function as data.
- **View:** A view is a request handler function, which receives HTTP requests and returns HTTP responses. Views access the data needed to satisfy requests via *models*, and delegate the formatting of the response to *templates*.
- **Models:** Models are Python objects that define the structure of an application's data, and provide mechanisms to manage (add, modify, delete) and query records in the database.
- **Templates:** A template is a text file defining the structure or layout of a file (such as an HTML page), with placeholders used to represent actual content. A *view* can dynamically create an HTML page using

an HTML template, populating it with data from a *model*. A template can be used to define the structure of any type of file; it doesn't have to be HTML!

Setting Up:

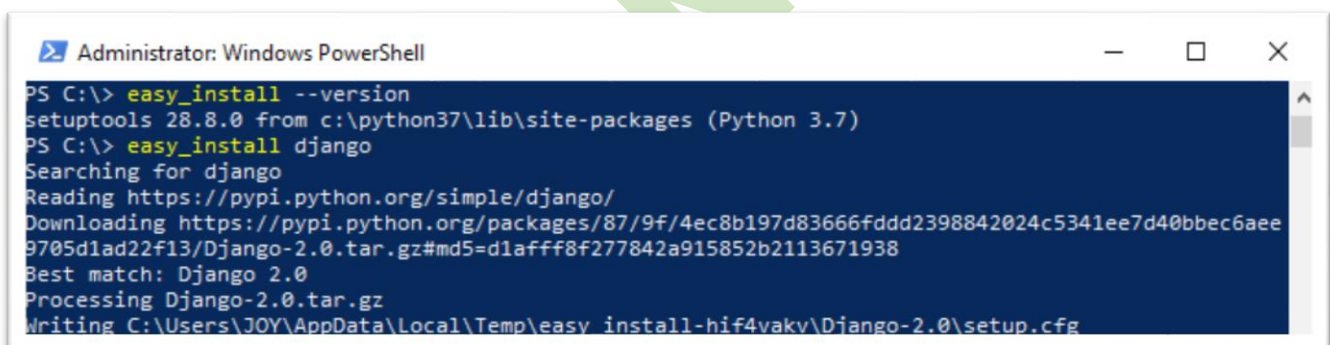
First check you pre requisites for Django:

Open Powershell from win search and check the version of python and installer



```
Administrator: Windows PowerShell
PS C:\> cd python37
PS C:\python37> cd\
PS C:\> python --version
Python 3.7.0a2
PS C:\> easy_install --version
setuptools 28.8.0 from c:\python37\lib\site-packages (Python 3.7)
PS C:\>
```

After all setup shows fine, type Django install command:



```
Administrator: Windows PowerShell
PS C:\> easy_install --version
setuptools 28.8.0 from c:\python37\lib\site-packages (Python 3.7)
PS C:\> easy_install django
Searching for django
Reading https://pypi.python.org/simple/django/
Downloading https://pypi.python.org/packages/87/9f/4ec8b197d83666fddd2398842024c5341ee7d40bbec6aee9705d1ad22f13/Django-2.0.tar.gz#md5=d1afff8f277842a915852b2113671938
Best match: Django 2.0
Processing Django-2.0.tar.gz
Writing C:\Users\JOY\AppData\Local\Temp\easy_install-hif4vakv\Django-2.0\setup.cfg
```

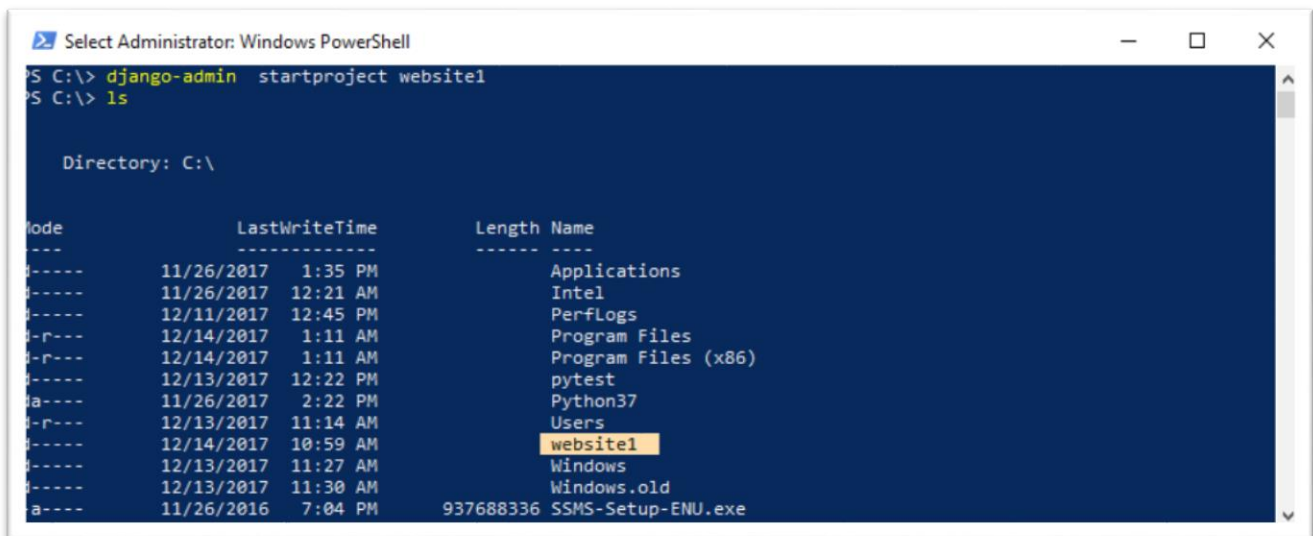
After install check the Django installed version:



```
Administrator: Windows PowerShell
PS C:\> django-admin --version
2.0
PS C:\>
```

Creating Django 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.



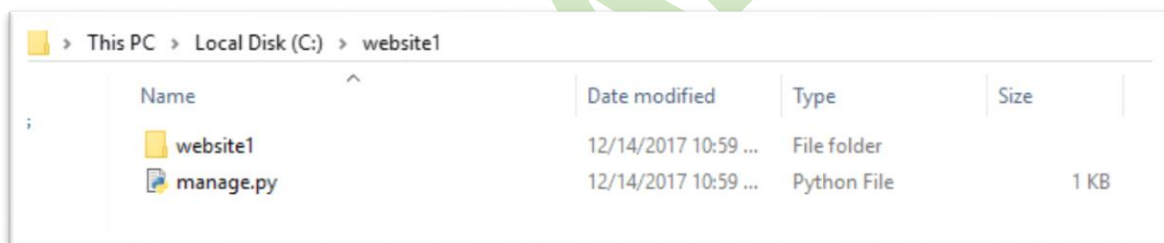
```
Select Administrator: Windows PowerShell

C:\> django-admin startproject website1
C:\> ls

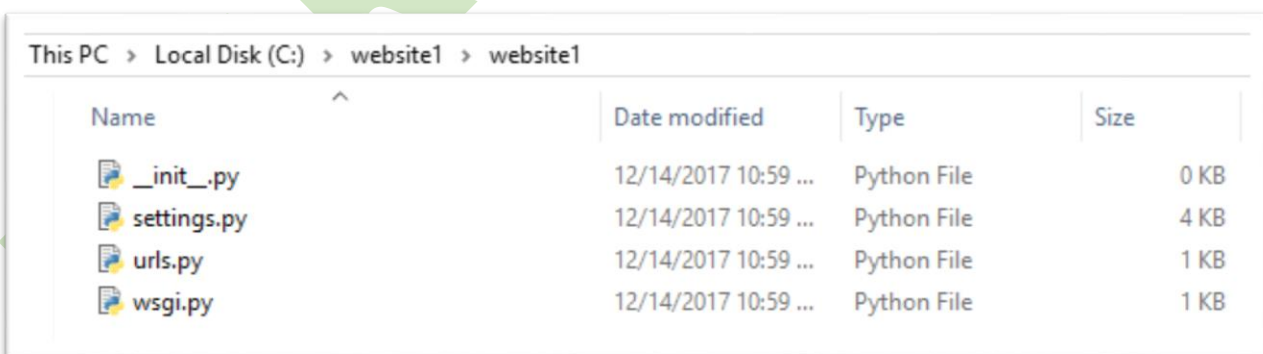
Directory: C:\

Mode                LastWriteTime         Length Name
----                -
d-----          11/26/2017    1:35 PM        Applications
d-----          11/26/2017   12:21 AM            Intel
d-----          12/11/2017   12:45 PM        PerfLogs
d-r-----        12/14/2017    1:11 AM      Program Files
d-r-----        12/14/2017    1:11 AM  Program Files (x86)
d-----          12/13/2017   12:22 PM        pytest
d-a-----        11/26/2017    2:22 PM      Python37
d-r-----        12/13/2017   11:14 AM        Users
d-----          12/14/2017   10:59 AM      website1
d-----          12/13/2017   11:27 AM        Windows
d-----          12/13/2017   11:30 AM    Windows.old
d-a-----        11/26/2016    7:04 PM 937688336 SSMS-Setup-ENU.exe
```

Now you can see your project:



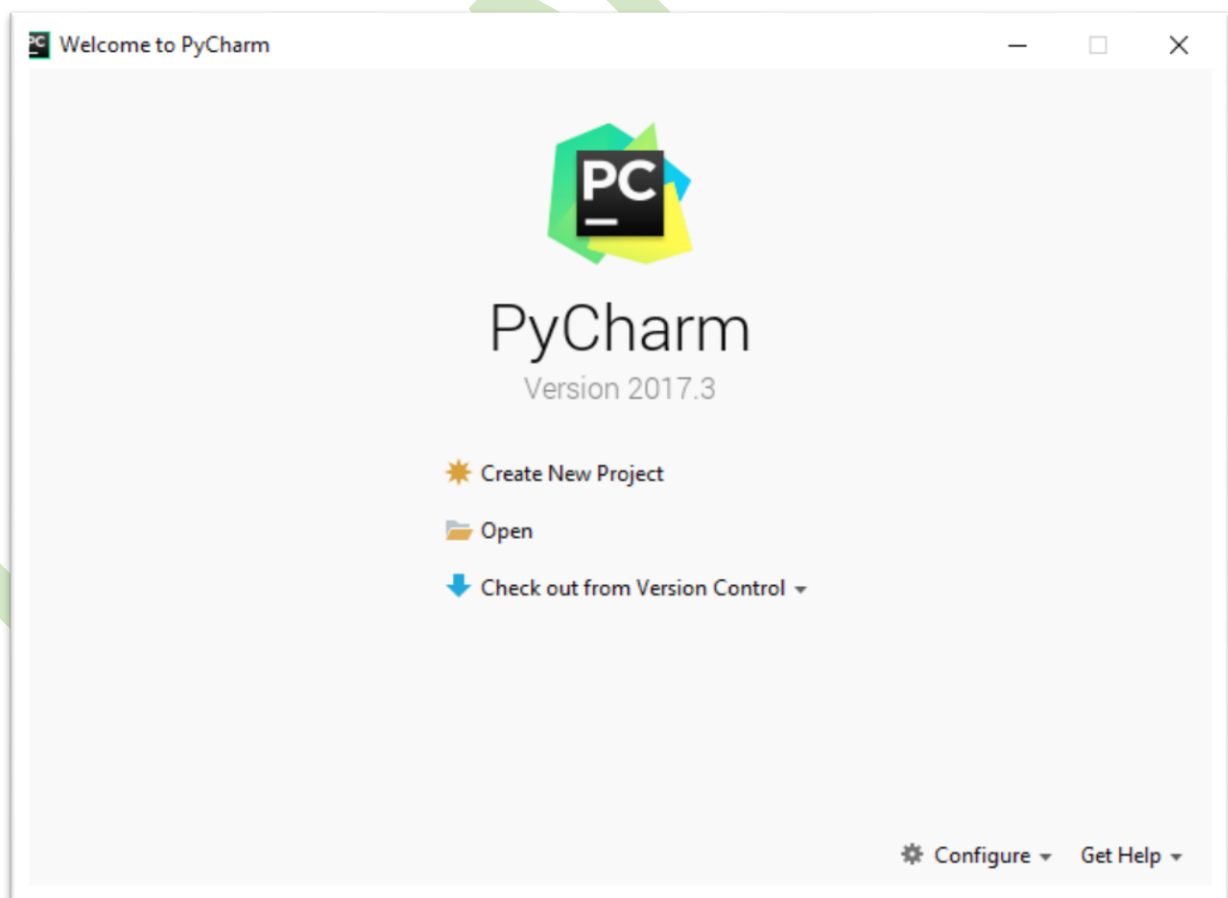
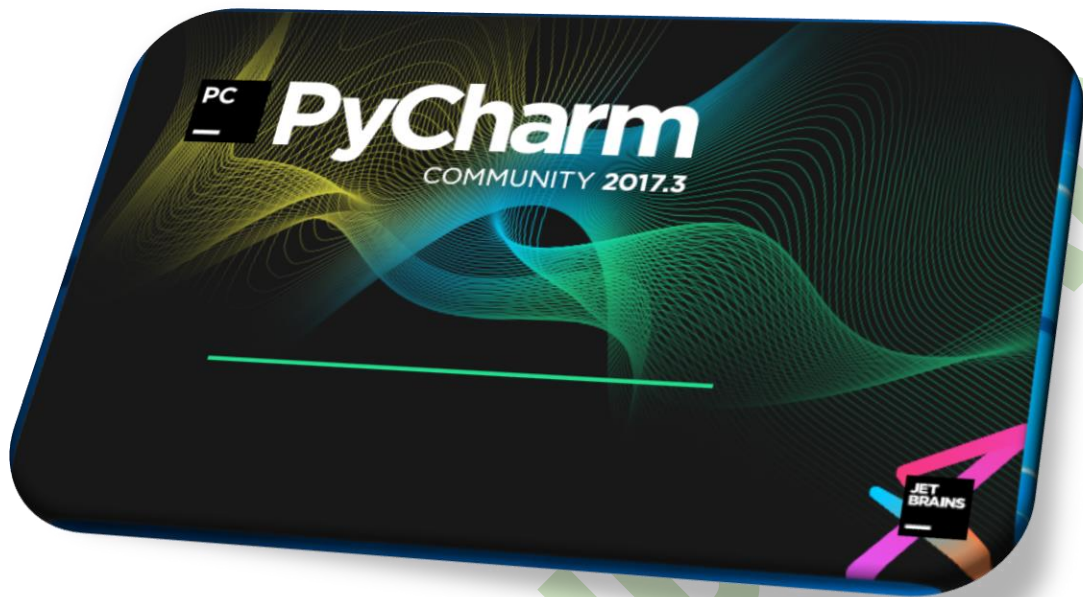
Name	Date modified	Type	Size
website1	12/14/2017 10:59 ...	File folder	
manage.py	12/14/2017 10:59 ...	Python File	1 KB

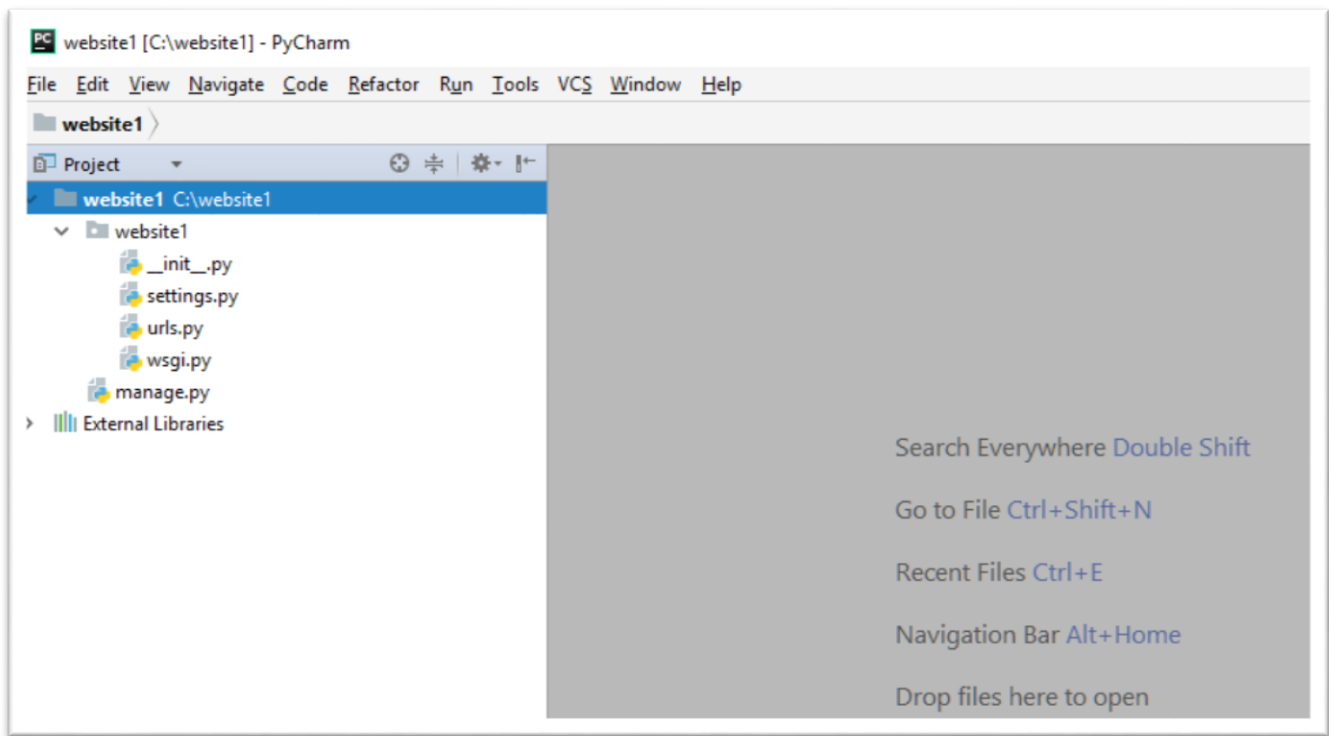


Name	Date modified	Type	Size
__init__.py	12/14/2017 10:59 ...	Python File	0 KB
settings.py	12/14/2017 10:59 ...	Python File	4 KB
urls.py	12/14/2017 10:59 ...	Python File	1 KB
wsgi.py	12/14/2017 10:59 ...	Python File	1 KB

Install PYCHARM IDE (community edition for free) Django web dev or use any other IDE

Open Pycharm:





About project files:

These files are:

- The outer **mysite/** root directory is just 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.
- 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.
- **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.
- **mysite/wsgi.py**: An entry-point for WSGI-compatible web servers to serve your project.

Now test your website:

The development server

Let's verify your Django project works. Change into the outer **mysite** directory, if you haven't already, and run the following commands:

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

You'll see the following output on the command line:

Performing system checks...

System check identified no issues (0 silenced).

You have unapplied migrations; your app may not work properly until they are applied.

Run 'python manage.py migrate' to apply them.

December 13, 2017 - 15:50:53

Django version 2.0, using settings 'mysite.settings'

Starting development server at <http://127.0.0.1:8000/>

Quit the server with CONTROL-C.

```
Select Administrator: Windows PowerShell

PS C:\> cd website1
PS C:\website1> python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).

You have 14 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
December 14, 2017 - 11:25:49
Django version 2.0, using settings 'website1.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Open any browser: type the URL that provide from run runserver:

<http://127.0.0.1:8000/>

