

Django tutorial (Windows)

[Getting started with Django | Django \(djangoproject.com\)](https://www.djangoproject.com)

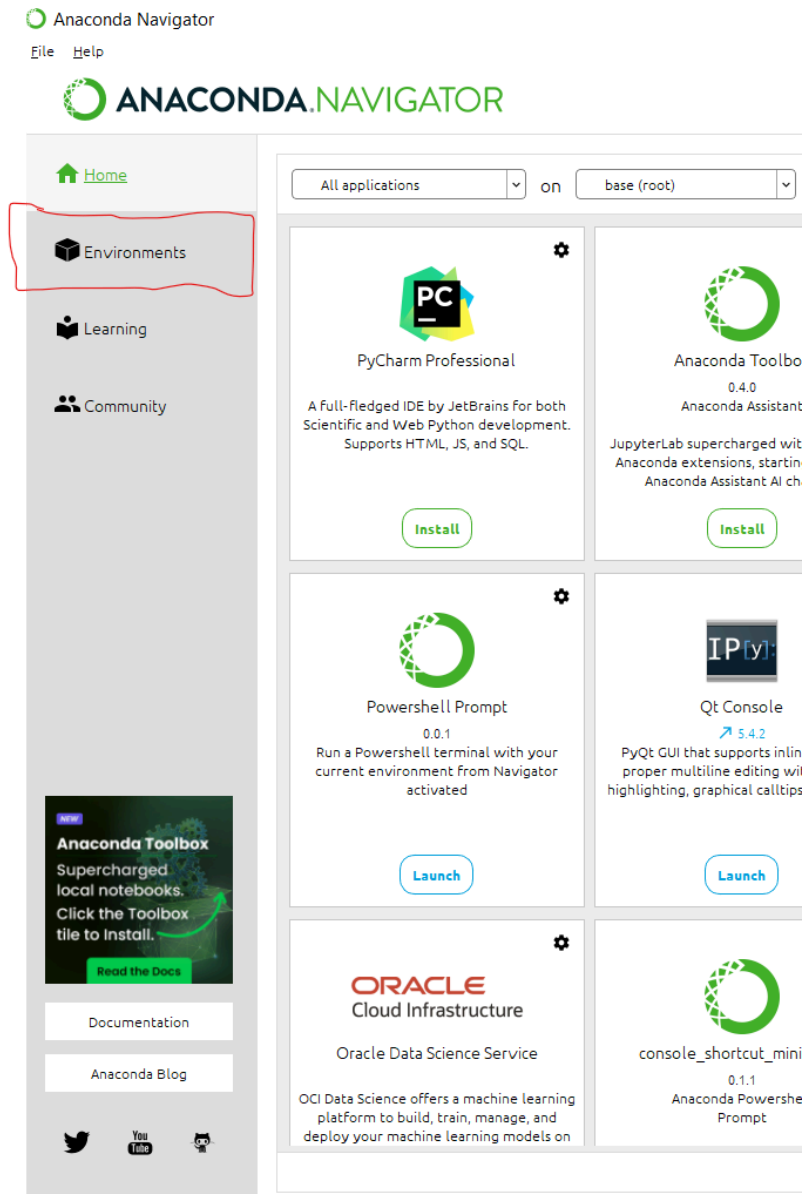
This Django tutorial is based on the official Django tutorial from [djangoproject.com](https://www.djangoproject.com) (goes up to part 2 of 'Writing your first Django app) but with a few slight differences for the sake of simplicity and convenience. The code and commands here were tested using Windows so Linux and Mac systems may or may not experience different results.

Table of contents

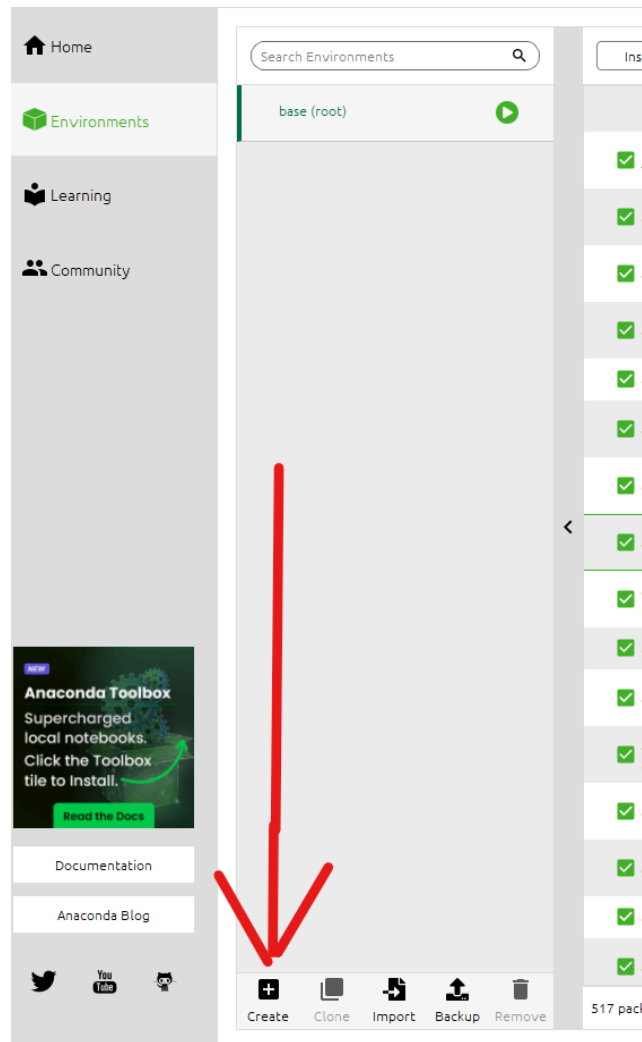
Table of contents	1
Anaconda Navigator (virtual environment setup)	2
Django setup	5
Django tutorial	8
Django tutorial Part 2	13

Anaconda Navigator (virtual environment setup)

1. Install Anaconda Navigator
2. Open it and navigate to environments



3. This is where we'll be creating a virtual environment that Django will use.
Create a new environment by clicking **'Create'**



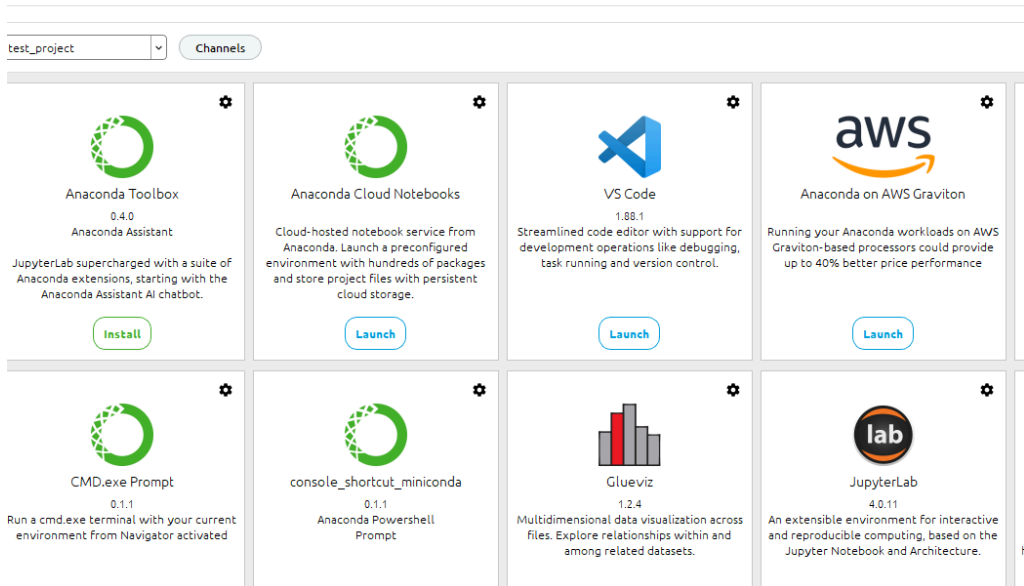
4. Name the environment 'test_project' , tick the python package and select the latest version of Python available. Then click 'Create'.

The image shows a 'Create new environment' dialog box. It has a title bar with a close button. Inside, there's a 'Name' field with 'test_project' entered. Below it is a 'Location' field showing a file path. Under the 'Packages' section, there are two rows: 'Python' with a checked checkbox and a dropdown menu showing '3.12.2', and 'R' with an unchecked checkbox and a dropdown menu showing '3.6.1'. At the bottom are 'Cancel' and 'Create' buttons.

5. Once it's finished loading, go back to the home page and make sure it's on 'test_project'



6. Find VS Code and click 'launch'.

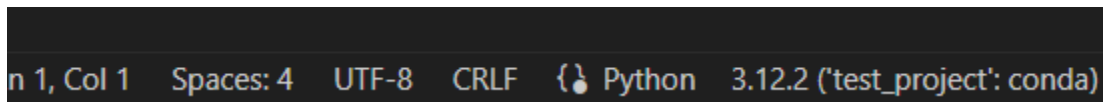


Django setup

7. Once VS Code is open, press 'Ctrl + N' or simply create a new file and change the programming language at the bottom right to Python. (Click on 'Plain Text' and type 'Python', then click on it) If you can't find Python then you'll need to install the Python extension in VS Code.

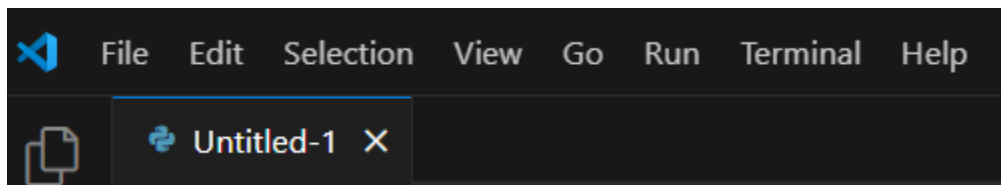


8. Doing so should display this,



which shows that you're currently working in the virtual environment. (Although VS Code can create virtual environments, using Anaconda to create one allows you to change the Python version to suit your needs if necessary.)

9. Start by opening a new terminal (top left corner).



10. Once it's open, enter `py -m pip install --upgrade pip` and run it.

```
PS C:\Users\PSC\mysite>PS C:\Users\PSC\mysite> py -m pip install --upgrade pip
Requirement already satisfied: pip in
c:\users\psc\AppData\Local\Programs\Python\Python312\lib\site-packages
(24.0)
```

11. Once it's done updating, we'll now start to install Django. While you're in the terminal, type `py -m pip install Django` and run it. Ensure the spelling and capitalisation are all correct.

```
PS C:\Users\PSC\mysite> py -m pip install Django
Collecting Django
  Using cached Django-5.0.4-py3-none-any.whl.metadata (4.1 kB)
Requirement already satisfied: asgiref<4,>=3.7.0 in
c:\users\psc\appdata\local\programs\python\python312\lib\site-packages (from
Django) (3.8.1)
Requirement already satisfied: sqlparse>=0.3.1 in
c:\users\psc\appdata\local\programs\python\python312\lib\site-packages (from
Django) (0.4.4)
Requirement already satisfied: tzdata in
c:\users\psc\appdata\local\programs\python\python312\lib\site-packages (from
Django) (2024.1)
Using cached Django-5.0.4-py3-none-any.whl (8.2 MB)
Installing collected packages: Django
Successfully installed Django-5.0.4
```

12. After Django is done installing, we can check to make sure it's installed by running 'py -m django --version' in the terminal.

```
PS C:\Users\PSC\mysite> py -m django --version
5.0.4
```

If Django is installed, a version number will appear. For this tutorial Django needs to be at least version 5.0 or above.

13. Now that Django is installed in Python, we now need to install Django into our virtual environments. Run 'pip install Django' in the terminal:

```
PS C:\Users\PSC\mysite> pip install Django
Collecting Django
  Using cached Django-5.0.4-py3-none-any.whl.metadata (4.1 kB)
Collecting asgiref<4,>=3.7.0 (from Django)
  Using cached asgiref-3.8.1-py3-none-any.whl.metadata (9.3 kB)
Collecting sqlparse>=0.3.1 (from Django)
  Using cached sqlparse-0.5.0-py3-none-any.whl.metadata (3.9 kB)
Collecting tzdata (from Django)
  Using cached tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 kB)
Using cached Django-5.0.4-py3-none-any.whl (8.2 MB)
Using cached asgiref-3.8.1-py3-none-any.whl (23 kB)
```

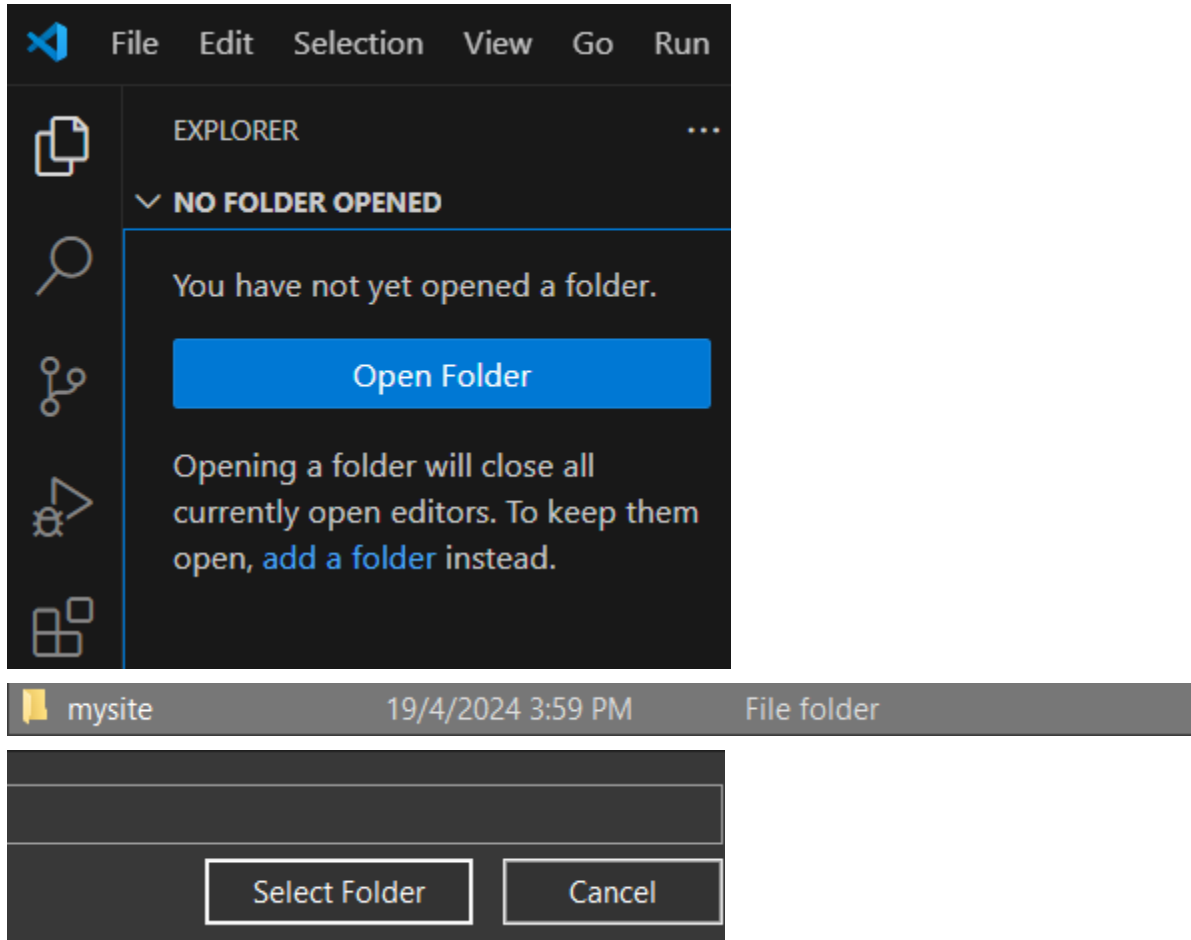
```
Using cached sqlparse-0.5.0-py3-none-any.whl (43 kB)
Using cached tzdata-2024.1-py2.py3-none-any.whl (345 kB)
Installing collected packages: tzdata, sqlparse, asgiref, Django
Successfully installed Django-5.0.4 asgiref-3.8.1 sqlparse-0.5.0 tzdata-2024.1
```

14. To create our Django project, run 'django-admin startproject mysite' in the terminal. Doing this will tell Django to create a project named 'mysite', the name can be modified if you wish.

```
PS C:\Users\PSC\mysite> django-admin startproject mysite
```

Django tutorial

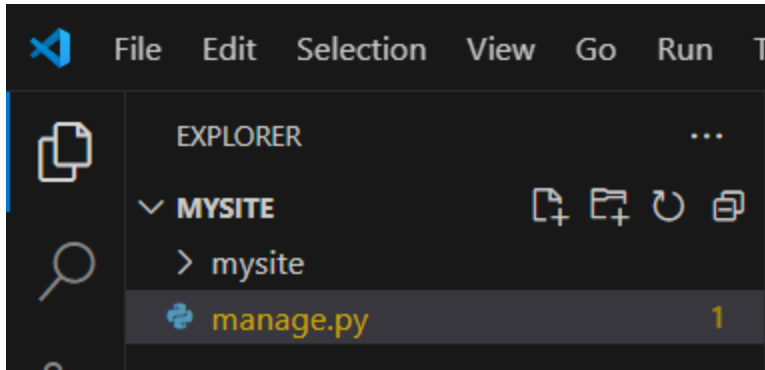
15. To make navigating through our different files easier, we'll need to create a workspace in VS Code. Click on explorer and click on 'Open Folder'. Now find the file that contains your Django project and select it.



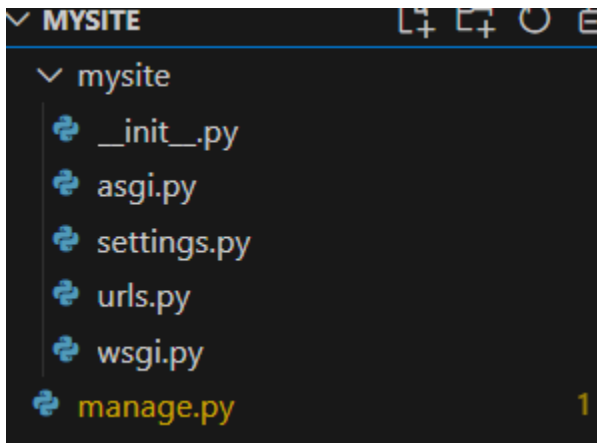
16. Alternatively, if you want to store your project elsewhere, simply type 'cd' and add a space, followed by the file path of your destination of choice.

```
PS C:\Users\PSC\mysite> cd C:\Users\PSC
PS C:\Users\PSC>
```

17. Upon doing so, you should see a directory towards the left which allows us to access our project's different files and folders more easily.



18. Now, let's take a look at the files that came with our project:



- manage.py: This file is a command-line utility that allows you to interact with the project in numerous ways including checking what changes have been made to your project or running your website.
- __init__.py: This file tells Python that the inner 'mysite' file should be considered a Python package.
- settings.py: This file acts as the settings/configuration for your project and can be modified.
- urls.py: This file contains the URLs that your project will use, it can also be reused in other projects.
- asgi.py: This file acts as an entry point for ASGI-compatible web servers to host your project.
- wegi.py: This file acts as an entry point for ASGI-compatible web servers to host your project.

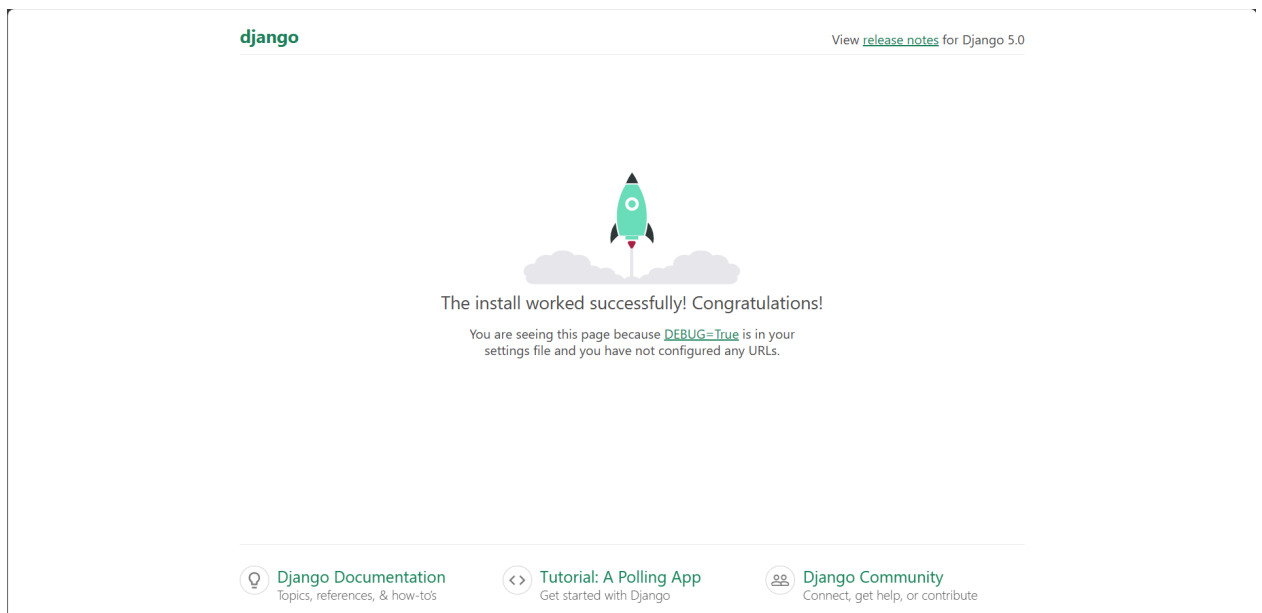
19. Try running `py manage.py runserver` in your VS Code terminal, you should then see something like this:

```
PS C:\Users\PSC\mysite> py manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 18 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.
April 23, 2024 - 11:37:44
Django version 5.0.4, using settings 'mysite.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

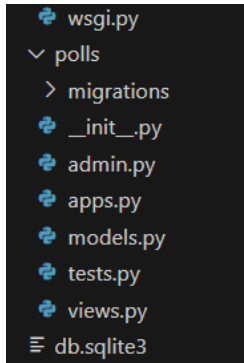
Once this is done, you can visit the link shown in your terminal (in this case it's <http://127.0.0.1:8000/>). When you visit the page you should see this:



If you see this, it means that your Django server deployed successfully.

20. Now, stop the server by pressing `CTRL + C` at the same time while in the terminal.

21. By default, Django launches its server with a port number of 8000, this can be changed. To do so, type 'py manage.py runserver' but don't run it just yet. After 'runserver', add any 4 digit number and then run the code, it should look something like 'py manage.py runserver 9000'.
22. Now, we'll be creating our 'polls' app. Ensure your terminal is in the same directory as 'manage.py' and run 'py manage.py startapp polls'.
- This should then appear:



23. Open 'views.py' in the polls folder and type the following code(always save the file you're working on after making any changes):

```
from django.http import HttpResponse

def index(request):

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

24. This is what your server should look like:

You may have noticed that nothing has changed, this is because even though we just created a view, we still haven't told our project to navigate and find this view.

25. To fix this, create a python file named 'urls.py' in the polls folder (for future reference when opening files, we'll be using 'folder name'/'file name'; for example 'polls/urls.py', which is what we've opened here) and type the following code:

```
from django.urls import path
```

```
from . import views

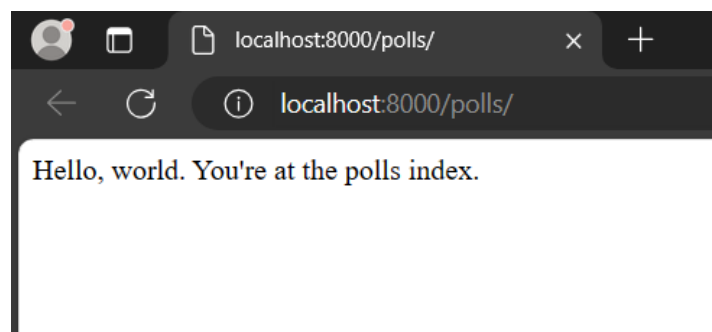
urlpatterns = [
    path("", views.index, name="index"),
]
```

26. Now, we'll be instructing our project to navigate through specific directories so that we can see for ourselves what our view actually looks like. To do so, open 'mysite/urls.py' and type in the following code (you can just leave the orange text above alone):

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

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

27. The 'include' function in Django allows the project to reference other URLs. Whenever Django sees 'include()', it removes whatever part of the URL leading up to a certain point and sends the rest through, which makes it easy to 'plug and play' numerous URLs. Now run 'py manage.py runserver' and visit <http://localhost:8000/polls/> or <http://localhost:8000/>, if the first link doesn't work. If you've used a different port number for your server, make sure to make that change in the link when opening it. Note that you'll encounter a 'Page not found' (error 404) if you try visiting the link given in your VS Code terminal. Your page should look like this:



Django tutorial Part 2

1. By default, Django uses SQLite as its database since it comes pre-installed with Python. In the future, you may want to switch to a more scalable database like PostgreSQL to avoid database-switching headaches down the road. Now, open 'mysite/settings.py' and scroll down until line 33 where you can see the 'INSTALLED_APPS':

```
INSTALLED_APPS = [  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
]
```

2. We'll go through briefly what each of these apps do:
 - django.contrib.admin: The admin site. We'll be using it soon
 - django.contrib.auth: An authentication system
 - django.contrib.contenttypes: A framework for different content types
 - django.contrib.sessions: A session framework
 - django.contrib.messages: A messaging framework
 - django.contrib.staticfiles: A framework for managing static files
3. Run 'py manage.py migrate' in the VS Code terminal, you should see something like this:

```
PS C:\Users\PSC\mysite> py manage.py migrate  
Operations to perform:  
  Apply all migrations: admin, auth, contenttypes, sessions  
Running migrations:  
  Applying contenttypes.0001_initial... OK
```

```
Applying auth.0001_initial... OK
Applying admin.0001_initial... OK
Applying admin.0002_logentry_remove_auto_add... OK
Applying admin.0003_logentry_add_action_flag_choices... OK
Applying contenttypes.0002_remove_content_type_name... OK
Applying auth.0002_alter_permission_name_max_length... OK
Applying auth.0003_alter_user_email_max_length... OK
Applying auth.0004_alter_user_username_opts... OK
Applying auth.0005_alter_user_last_login_null... OK
Applying auth.0006_require_contenttypes_0002... OK
Applying auth.0007_alter_validators_add_error_messages... OK
Applying auth.0008_alter_user_username_max_length... OK
Applying auth.0009_alter_user_last_name_max_length... OK
Applying auth.0010_alter_group_name_max_length... OK
Applying auth.0011_update_proxy_permissions... OK
Applying auth.0012_alter_user_first_name_max_length... OK
Applying sessions.0001_initial... OK
```

4. In our polls app, we'll be creating 2 models: 'Question' and 'Choice'. 'Question' has a question and publication date while 'Choice' has 2 fields, a text of choice and a vote tally. Each choice will be associated with a question. Open 'polls/models.py' and type the following code:

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)

    votes = models.IntegerField(default=0)
```

5. Here, each model (Question and Choice) are represented by subclasses (models.Model). Each model has a number of class variables, each of which represents a database field in the model. Each field is then represented by an instance of a 'Field' class, e.g. 'CharField' for character fields and

'DateTimeField' for datetimes. A relationship is then defined between the 2 models using 'ForeignKey', which tells Django that each 'Choice' is related to a single 'Question'.

6. We now need to inform our project that the 'polls' app is installed. To do this, open 'mysite/settings.py' and edit the 'INSTALLED_APPS' section by adding "polls.apps.PollsConfig" in there and add a comma at the end (the comma is very important). It should now look something like this:

```
INSTALLED_APPS = [  
    "polls.apps.PollsConfig",  
    "django.contrib.admin",  
    "django.contrib.auth",  
    "django.contrib.contenttypes",  
    "django.contrib.sessions",  
    "django.contrib.messages",  
    "django.contrib.staticfiles",  
]
```

7. Run 'py manage.py makemigrations polls' in the VS Code terminal and you should see this:

```
PS C:\Users\PSC\mysite> py manage.py makemigrations polls  
Migrations for 'polls':  
  polls\migrations\0001_initial.py  
    - Create model Question  
    - Create model Choice
```

By running 'makemigrations', we're telling Django that we've made some changes to our models (or made new ones) and that you'd like these changes to be stored as a migration. Migrations are how Django stores changes to your models, they're files on disk.

8. Run 'py manage.py sqlmigrate polls 0001' in your terminal and you should see something like this (note that the exact output may vary depending on what database you're using):

```

PS C:\Users\PSC\mysite> py manage.py sqlmigrate polls 0001
BEGIN;
--
-- Create model Question
--
CREATE TABLE "polls_question" ("id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "question_text" varchar(200) NOT NULL, "pub_date" datetime
NOT NULL);
--
-- Create model Choice
--
CREATE TABLE "polls_choice" ("id" integer NOT NULL PRIMARY KEY
AUTOINCREMENT, "choice_text" varchar(200) NOT NULL, "votes" integer NOT
NULL, "question_id" bigint NOT NULL REFERENCES "polls_question" ("id")
DEFERRABLE INITIALLY DEFERRED);
CREATE INDEX "polls_choice_question_id_c5b4b260" ON "polls_choice"
("question_id");
COMMIT;

```

9. The 'sqlmigrate' command doesn't actually make the migration happen, instead, it shows you what SQL Django thinks is needed.
Now run 'py manage.py migrate' in the terminal.

```

PS C:\Users\PSC\mysite> py manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, polls, sessions
Running migrations:
  Applying polls.0001_initial... OK

```

The 'migrate' command checks all the unapplied migrations, runs them against your database and synchronises the changes you made to the models with schema in your database. Migrations are very powerful since they allow you to change your models over time without the need to delete your databases/tables or make new ones - it specialises in upgrading your database live, without losing any data. To make future changes to your models, remember this three step process:

1. Change your models in 'models.py'
 2. Run 'py manage.py make migrations' to create migrations for those changes
 3. Run 'py manage.py migrate' to apply those changes to the database
10. Run 'py manage.py shell' in your terminal to invoke the Python shell and type in the following commands as shown (the greyed-out text in the picture consist only of informative notes or output from the terminal, you do not type those out in the terminal):

```
>>> from polls.models import Choice, Question  # Import the model
classes we just wrote.

# No questions are in the system yet.
>>> Question.objects.all()
<QuerySet []>

# Create a new Question.
# Support for time zones is enabled in the default settings file, so
# Django expects a datetime with tzinfo for pub_date. Use timezone.now()
# instead of datetime.datetime.now() and it will do the right thing.
>>> from django.utils import timezone
>>> q = Question(question_text="What's new?", pub_date=timezone.now())

# Save the object into the database. You have to call save() explicitly.
>>> q.save()

# Now it has an ID.
>>> q.id
1

# Access model field values via Python attributes.
>>> q.question_text
"What's new?"
>>> q.pub_date
datetime.datetime(2012, 2, 26, 13, 0, 0, 775217,
tzinfo=datetime.timezone.utc)

# Change values by changing the attributes, then calling save().
>>> q.question_text = "What's up?"
>>> q.save()

# objects.all() displays all the questions in the database.
>>> Question.objects.all()

<QuerySet [<Question: Question object (1)>]>
```

11. To exit the Python shell simply type 'exit()' or press 'CTRL + Z' and enter. Open 'polls/models.py' and add this code to it:

```
from django.db import models

class Question(models.Model):
    # ...
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    # ...
    def __str__(self):
        return self.choice_text
```

The '# ...' section tells us that we should leave the above section alone. Your code should now look like this:

```
from django.db import models

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")
    def __str__(self):
        return self.question_text

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    def __str__(self):
        return self.choice_text
```

12. The '__str__()' method makes it more convenient when using the Python shell. Now edit 'polls/models.py' with the following:

```

import datetime

from django.db import models
from django.utils import timezone

class Question(models.Model):
    # ...
    def was_published_recently(self):

        return self.pub_date >= timezone.now() -
            datetime.timedelta(days=1)

```

Your code should now look like this:

```

import datetime

from django.db import models
from django.utils import timezone

class Question(models.Model):
    question_text = models.CharField(max_length=200)
    pub_date = models.DateTimeField("date published")
    def __str__(self):
        return self.question_text
    def was_published_recently(self):
        return self.pub_date >= timezone.now() -
            datetime.timedelta(days=1)

class Choice(models.Model):
    question = models.ForeignKey(Question, on_delete=models.CASCADE)
    choice_text = models.CharField(max_length=200)
    votes = models.IntegerField(default=0)
    def __str__(self):
        return self.choice_text

```

The 'class Choice(models.Model)' section was left out because no changes were made to it.

13. The addition of 'import datetime' allows us to use Python's standard 'datetime' module while 'django.utils import timezone' allows us to use

Django's time-zone-related utilities. Now open the Python shell by running 'py manage.py shell' in your terminal again and type in the following commands (there are multiple pictures):

```
>>> from polls.models import Choice, Question

# Make sure our __str__() addition worked.
>>> Question.objects.all()
<QuerySet [<Question: What's up?>]>

# Django provides a rich database lookup API that's entirely driven by
# keyword arguments.
>>> Question.objects.filter(id=1)
<QuerySet [<Question: What's up?>]>
>>> Question.objects.filter(question_text__startswith="What")
<QuerySet [<Question: What's up?>]>

# Get the question that was published this year.
>>> from django.utils import timezone
>>> current_year = timezone.now().year
>>> Question.objects.get(pub_date__year=current_year)
<Question: What's up?>

# Request an ID that doesn't exist, this will raise an exception.
>>> Question.objects.get(id=2)
Traceback (most recent call last):
...
DoesNotExist: Question matching query does not exist.

# Lookup by a primary key is the most common case, so Django provides a
# shortcut for primary-key exact lookups.
# The following is identical to Question.objects.get(id=1).
>>> Question.objects.get(pk=1)
<Question: What's up?>

# Make sure our custom method worked.
>>> q = Question.objects.get(pk=1)
>>> q.was_published_recently()
True

# Give the Question a couple of Choices. The create call constructs a
new
# Choice object, does the INSERT statement, adds the choice to the set
# of available choices and returns the new Choice object. Django creates
# a set (defined as "choice_set") to hold the "other side" of a
ForeignKey
# relation (e.g. a question's choice) which can be accessed via the API.
>>> q = Question.objects.get(pk=1)

# Display any choices from the related object set -- none so far.
>>> q.choice_set.all()
```

```

<QuerySet []>

# Create three choices.
>>> q.choice_set.create(choice_text="Not much", votes=0)
<Choice: Not much>
>>> q.choice_set.create(choice_text="The sky", votes=0)
<Choice: The sky>
>>> c = q.choice_set.create(choice_text="Just hacking again", votes=0)

# Choice objects have API access to their related Question objects.
>>> c.question
<Question: What's up?>

# And vice versa: Question objects get access to Choice objects.
>>> q.choice_set.all()
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking
again>]>
>>> q.choice_set.count()
3

# The API automatically follows relationships as far as you need.
# Use double underscores to separate relationships.
# This works as many levels deep as you want; there's no limit.
# Find all Choices for any question whose pub_date is in this year
# (reusing the 'current_year' variable we created above).
>>> Choice.objects.filter(question__pub_date__year=current_year)
<QuerySet [<Choice: Not much>, <Choice: The sky>, <Choice: Just hacking
again>]>

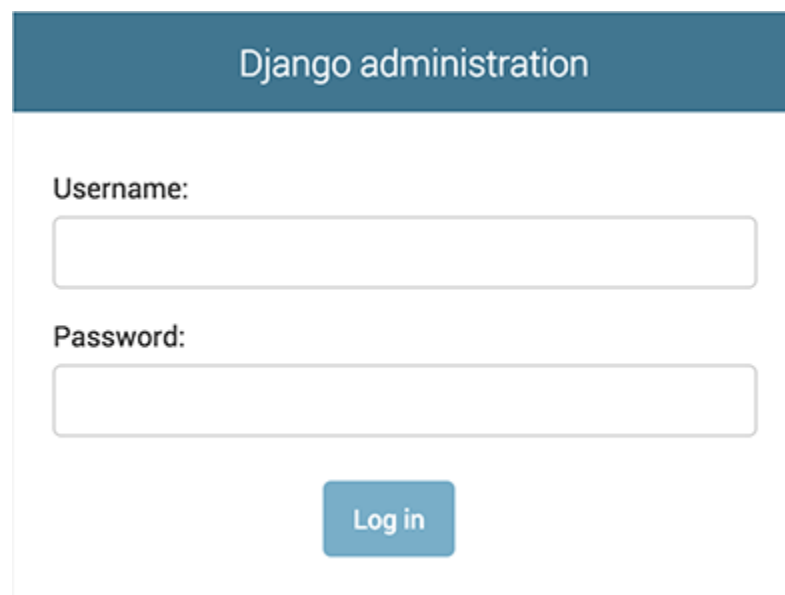
# Let's delete one of the choices. Use delete() for that.
>>> c = q.choice_set.filter(choice_text__startswith="Just hacking")

>>> c.delete()

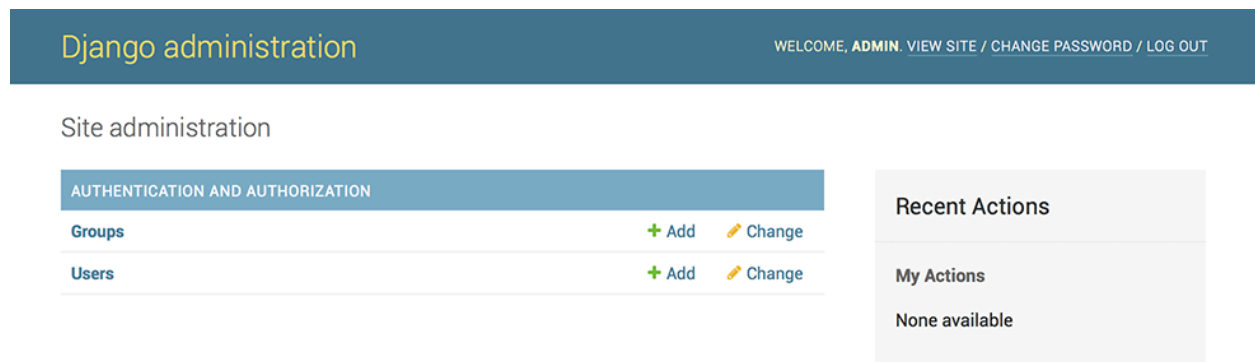
```

14. What we've done is essentially create several choices to choose from as well as learn how to filter between the different choices. Now, we'll be creating an admin user for our server. Run 'py manage.py createsuperuser' in the terminal. Once you've done this, you'll need to enter a username, your email, and a password (you'll need to type your password twice to confirm it):
15. Start the server by running 'py manage.py runserver' in the terminal and then visit <http://127.0.0.1:8000/admin/> (change the port number accordingly if you've used a different one):

```
PS C:\Users\PSC\mysite> py manage.py createsuperuser
Username (leave blank to use 'psc'): admin
Email address: admin@gmail.com
Password:
Password (again):
The password is too similar to the username.
This password is too short. It must contain at least 8 characters.
This password is too common.
Bypass password validation and create user anyway? [y/N]: y
Superuser created successfully.
```

The image shows the Django administration login interface. It features a dark blue header with the text "Django administration". Below the header, there is a white box containing two input fields: "Username:" and "Password:". A blue "Log in" button is positioned below the password field.

16. Enter your username and password to reach this screen:

The image displays the Django administration dashboard after a successful login. The top header is dark blue with "Django administration" on the left and "WELCOME, ADMIN. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)" on the right. Below the header, the page is titled "Site administration". On the left, there is a section titled "AUTHENTICATION AND AUTHORIZATION" with two rows: "Groups" and "Users". Each row has a green "+ Add" link and a yellow pencil icon followed by a "Change" link. On the right, there is a "Recent Actions" section with a "My Actions" link and the text "None available".

17. The 'AUTHENTICATION AND AUTHORIZATION' section is provided by 'django.contrib.auth' in the 'INSTALLED_APPS' section of

'mysite/setting.py'. However, our polls app is seemingly missing. To remedy this, open 'polls/admin.py' and edit the code such that it looks like this:

```
from django.contrib import admin
from .models import Question

admin.site.register(Question)
```

18. Once you've saved the changes, reload your page and you should find this:

Site administration

AUTHENTICATION AND AUTHORIZATION	
Groups	+ Add Change
Users	+ Add Change

POLLS	
Questions	+ Add Change

Recent Actions

My Actions

None available

19. Click on 'Questions' to reach this page:

Home > Polls > Questions

Select question to change [ADD QUESTION +](#)

Action: 0 of 1 selected

<input type="checkbox"/>	QUESTION
<input type="checkbox"/>	What's up?



1 question

20. Click on 'What's up?' and edit it however you like:

Change question

HISTORY

Question text:

Date published: Date: Today 
Time: Now 

Delete

Save and add another

Save and continue editing

SAVE

21. The form you see here was generated from the 'Question' model and the different field types ('DateTimeField' and 'CharField') each have their own unique HTML input widget. The 'Today' and 'Now' shortcuts that you see beside 'Date' and 'Time' are Javascript shortcuts.

22. If you made any changes and saved them, you can view said changes by clicking on the question and navigating to 'HISTORY' in the top right corner which'll show you this:

Change history: What's up?

DATE/TIME	USER	ACTION
Sept. 6, 2015, 9:21 p.m.	elky	Changed pub_date.